BACHELOR OF COMPUTER APPLICATIONS

ADVANCED PROGRAMMING IN C

PREVIOUS YEARS QUESTIONS

UNIT-I

प्रश्न 1 – डायनामिक ऐरे अलोकेशन क्या है? उदाहरण देकर समझाएँ।

What is Dynamic Array Allocation ? Explain with examples. *(June 2010; Jan. 2016)*

अथवा (Or)

C में डायनामिक ऐरे अलोकेशन की प्रक्रिया का उल्लेख करें।

Write the process of Dynamic Array Allocation in 'C'. *(June 2010; Jan. 2016)*

अथवा (Or)

निम्नलिखित को समझाइए:–

Explain the following :–

(i) sizeof() *(Jan. 2010, 12, 13, 15; June 2013)*

(ii) malloc() *(Jan. 2010, 12, 13, 15, 16; June 2012, 13, 14, 15)*

(iii) free() *(Jan. 2010, 12, 13, 16; June 2013, 14, 15)*

(iv) calloc() *(Jan. 2010, 16; June 2012, 13, 14, 15)*

(v) realloc() *(June 2015; Jan. 2010, 13)*

अथवा (Or)

malloc(), free() और sizeof() का प्रयोग करते हुए coding C-Language में लिखिए

With the use of functions malloc(), free() and sizeof() write coding inc. *(June 2011)*

(Jan. 2011, 16)

उत्तर–डायनामिक मेमोरी अलोकेशन (Dynamic Memory Allocation)

कम्प्यूटर में मेन मेमोरी एक संसाधन है जिसकी मात्रा सीमित है। अतः इसका व्यवस्थित प्रबंधन तथा इसके उपयोग पर नियंत्रण आवश्यक है, क्योंकि सभी सक्रिय प्रोग्राम जिनका क्रियान्वयन किया जाना है मेन मेमोरी में लोड होते हैं जिसमें प्रोग्राम का कोड तथा उससे सम्बंधित डाटा आदि सभी चीजें शामिल होती हैं। इसलिए कई बार प्रोग्रामों को मेन मेमोरी में लोड करते समय मेमोरी कम पड़ने की स्थिति में मेमोरी ओवरफ्लो की समस्या आती है।

सामान्यतः जब हमारा प्रोग्राम कम्पाइल होकर क्रियान्वयन के लिए मेन मेमोरी में लोड होता है तब ही उसके कोड तथा सभी वेरिएबलों के लिए आवश्यक मेमोरी का आवंटन अपने आप ही हो जाता है, इस प्रकार प्रोग्राम के कम्पाइल होते समय उसे आवंटित मेमोरी की मात्रा निश्चित रहती है जिसे हम स्टैटिक मेमोरी अलोकेशन कहते हैं।

लेकिन अनेक बार प्रोग्राम को कम्पाइल करते समय यह निश्चित नहीं हो पाता कि उसे कितनी मेमोरी की आवश्यकता है? ऐसी स्थिति में प्रोग्राम के रन होते समय उसे जितनी भी मेमोरी की आवश्यकता होती है वह लेता जाता है। इस प्रकार प्रोग्राम के रन होते समय उसे होने वाला अनिश्चित मेमोरी आवंटन डायनामिक मेमोरी अलोकेशन कहलाता है। इस प्रकार प्रोग्राम के रन होते समय आवंटित होने वाली मेमोरी की मात्रा हर बार (जब भी प्रेग्राम रन हो) अलग-अलग होती है।

डायनामिक ऐरे अलोकेशन (Dynamic Array Allocation)

डायनामिक मेमोरी अलोकेशन की आवश्यकता विशेष रूप से तब पड़ती है जब हम एक ऐसे ऐरे का उपयोग कर रहे हों, जिसका आकार निश्चित नहीं है। इस प्रकार एक अनिश्चित आकार के ऐरे प्रोग्राम रन होते समय किया जाने वाला मेमोरी आवंटन डायनामिक

Scanned by CamScanner

ऐरे अलोकेशन कहलाता है। इसमें ऐरे को आवंटित होने वाली मेमोरी की मात्रा पहले से निश्चित नहीं होती तथा हर बार अलग-अलग होती रहती है।

डायनामिक मेमोरी अलोकेशन फंक्शन (Dynamic Memory Allocation Function) *(Jan. 2011)*

डायनामिक मेमोरी अलोकेशन के लिए C में कुछ इनबिल्ट (Inbuilt) फंक्शनों की आवश्यकता पड़ती है, जो निम्नानुसार हैं–

(i) sizeof()–यह malloc() फंक्शन में पैरामीटर के रूप में उपयोग किया जाता है जो यह निर्धारित करता है कि malloc() फंक्शन द्वारा कितनी मेमोरी आवंटित की जानी है? यह फंक्शन पोर्टेबिलिटी को सुनिश्चित करता है जिसके द्वारा वापस की जाने वाली वैल्यू उस डाटा टाइप पर निर्भर करती है जो इसे आरग्यूमेन्ट रन के रूप में दिया जा रहा है; जैसे अगर हम sizeof(int) दें तो इसके द्वारा वापस की जाने वाली वेल्यू एक इन्टीजर वेरिएबल के आकार के अनुसार होगी, वहीं अगर हम sizeof(float) दें तो वापस की जाने वाली वैल्यू एक फ्लोट वेरिएबल के आकार के अनुसार होगी।

(ii) malloc()–malloc() का अर्थ है Memory Allocate अर्थात् यह फंक्शन वास्तव में भौतिक रूप से मेमोरी का एक ब्लॉक आवंटित करता है। malloc() फंक्शन एक निश्चित आकार का मेमोरी ब्लॉक आवंटित करता है जिसका उपयोग रन टाइम पर आवश्यकतानुसार मेमोरी ब्लॉक आवंटन के लिए किया जाता है। इस फंक्शन द्वारा आवश्यकतानुसार एक से अधिक ब्लॉक आवंटित किये जाने पर यह आवंटन कंटिन्यू ब्लॉक के रूप में होता है। इस फंक्शन में आरग्यूमेंट (Argument) के रूप में एक वेरिएवल तथा sizeof() फंक्शन दिया जाता है, जहाँ वेरिएबल का मान कुल आवंटित किये जाने वाले ब्लॉक्स की संख्या तथा sizeof() में दिया गया डाटा टाइप प्रत्येक ब्लॉक का आकार निर्धारित करता है; जैसे–मान लीजिए हम malloc (nun, sizeof(int) देते हैं जहां num का मान 3 है अतः इस स्थिति में यह फंक्शन 3 मेमारी ब्लॉक आवंटित करेगा। अब चूंकि sizeof में हमने int दिया है और प्रत्येक integer वेरिएबल 2 byte मेमोरी स्पेस लेता है अतः यहां आवंटित किये गये तीनों मेमोरी ब्लॉक 2-2 byte के होंगे। अगर हम sizeof() में int के स्थान पर float देते हैं तो आवंटित होने वाले तीनों ब्लॉक 4-4 byte के होते हैं। अगली बार प्रोग्राम रन करने पर अगर हम num का मान 5 दे दें तो यह फंक्शन 5 मेमोरी ब्लॉक आवंटित करेगा। इस प्रकार यहां आवंटित किये जाने वाले ब्लॉक की संख्या num के मान पर निर्भर करती है जो हम प्रोग्राम रन होने पर इनपुट करते हैं इसीलिए malloc() द्वारा किया जाने वाला मेमोरी आवंटन रन टाइम मेमोरी अलोकेशन कहलाता है।

(iii) free()–यह फंक्शन malloc() या calloc() फंक्शन द्वारा आवंटित की गई मेमोरी को प्रोग्राम समाप्त होने के बाद खाली करवाने के लिए उपयोग किया जाता है ताकि उस मेमोरी का पुनः उपयोग किया जा सके। सामान्यतः किसी प्रोग्राम में उपयोग किये गये वेरिएबल प्रोग्राम समाप्त होने के बाद भी नष्ट नहीं होते तथा वे मेन मेमोरी में बने रहते हैं अतः उनके द्वारा किये गये मेमोरी ब्लॉक अनावश्यक रूप से बंधे रहते हैं जिनका पुनः उपयोग नहीं हो पाता, ये वेरिएबल मेमोरी से तभी समाप्त होते हैं जब हम कम्प्यूटर को ऑफ (OFF) कर देते हैं। अतः प्रोग्राम समाप्त होने के बाद उसमें उपयोग किये गये वेरिएबलों को नष्ट कर मेमोरी खाली करवाने के लिए free() फंक्शन का उपयोग किया जाता है।

उदाहरण–निम्न प्रोग्राम के द्वारा हम malloc(), sizeof() तथा free() फंक्शनों का उपयोग व्यावहारिक रूप से समझेंगे–

```
#include<stdio.h>
#include<conio.h>
int main()
  {
        int i;
        int *ptr;
        int num;
        printf("How many values you want to store:");
        scanf("%d", & num);
        ptr=malloc(num*sizeof(int)); /*memory allocation*/
        if(ptr!=NULL)
            {
                for(i=o, i<num; i++)
                    {
            *(ptr+1)i;
```

Scanned by CamScanner

```
            }
    }
    else
        {
    printf("Not enough memory \n");
                                    return1;
        }
    if(ptr !=NULL)
        {
    for(i=0, i<num; i++)
        {
    printf("%d\n", *(ptr+(i));
        }
        }
    free (ptr);         /*free allocated memory */
    retrun0;
```

Output :

```
How many values you want to store : 3
0
1
2
```

उपरोक्त प्रोग्राम में हम रन टाइम में num की वेल्यू यूजर से पूछ रहे हैं तथा num को आर्ग्यूमेन्ट के रूप में malloc() फंक्शन को दे रहे हैं, अतः इस बार हम num की जो भी Vlue इनपुट करेंगे malloc() फंक्शन उतने ही मेमोरी ब्लॉक आवंटित करेगा। malloc() फंक्शन में दूसरे आर्ग्यूमेन्ट के रूप में हमने sizeof() फंक्शन लिखा है जिसमें आर्ग्यूमेन्ट के रूप में int दिया है जिसका अर्थ यह है कि malloc() द्वारा आवंटित किये जाने वाले मेमोरी ब्लॉक int मानों को संग्रहित करेंगे अतः प्रत्येक ब्लॉक का आकार 2 byte का होगा।

यदि मेमोरी का आवंटन सफलतापूर्वक हो जाता है, तो malloc() एक void प्वॉइंटर रिटर्न (Return) करता है। इसे हम एक प्वॉइंटर Variable के द्वारा दिखा सकते हैं जो आवंटित मेमोरी का address संग्रहित करता है। यदि आवश्यक मेमोरी उपलब्ध न होने के कारण मेमोरी आवंटन विफल हो जाता है तो malloc() फंक्शन एक NULL प्वॉइंटर जारी करता है। दी गई if कन्डीशन में हमने यही जाँच की है कि मेमोरी आवंटन सफलतापूर्वक हुआ या नहीं? अंत में free() फंक्शन ptr को आवंटित मेमोरी को मुक्त करवा रहा है।

(iv) calloc()—calloc() फंक्शन malloc() के समान ही होता है, इन दोनों में अंतर केवल इतना है कि calloc() फंक्शन का प्रारम्भिक मान शून्य होता है, जबकि malloc() फंक्शन का कोई प्रारम्भिक मान नहीं होता है।

इसके अतिरिक्त बाकी सभी चीजें दोनों फंक्शनों में समान होती हैं अर्थात् calloc() फंक्शन में भी दो आर्ग्यूमेन्ट दिये जाते हैं जिनमें पहला आर्ग्यूमेन्ट एक वेरिएबल होता है जो आवंटित किये जाने वाले मेमोरी ब्लॉकों की संख्या निर्धारित करता है, जबकि दूसरा आर्ग्यूमेन्ट sizeof() फंक्शन होता है जो आवंटित किये जा रहे मेमोरी ब्लॉकों का आकार निर्धारित करता है। malloc() के समान ही calloc() फंक्शन भी मेमोरी आवंटित होने पर एक void प्वॉइंटर जारी करता है तथा मेमोरी आवंटित हो पाने की स्थिति में NULL प्वॉइंटर जारी करता है।

उदाहरण—malloc() तथा calloc() फंक्शनों के बीच के अन्तर को हम निम्न प्रोग्राम के द्वारा समझ सकते हैं। इस प्रोग्राम में दोनों फंक्शनों का उपयोग किया गया है—

```
#include<stdio.h>
#include<conio.h>
#include<stdlib.h>
```

Scanned by CamScanner

```
{
    int i;
    float *c1, *m1;
    clrscr();
    c1=calloc(2, sizeof(float));
    m1=malloc(2*sizeof(float);

if(c1!=NULL &&m1!=NULL)
    {
        for(i=0; i<2; i++);
    {
        printf("c1[%d] holds; % 07.5f", i, c1[i]);
        printf("m1[%d]holds %7.5f\n", i, m1[i]);
    }
    free(c1);
    free(m1);
    return 0;
}
else
    {
        printf("\n Not enough memory \n");
        return1;
    }
}
```

Output :

```
c1[0] holds 0.00000, m1[0] holds –561362081.00000
c1[1] holds 0.00000, m1 [0] holds –561362081.00000
```

उपरोक्त प्रोग्राम के आउटपुट से स्पष्ट है कि calloc () द्वारा आवंटित मेमोरी ब्लॉक का प्रारम्भिक मान शून्य होता है, जबकि malloc() द्वारा आवंटित मेमोरी ब्लॉक का कोई प्रारम्भिक मान नहीं होता इसलिए वह एक अर्थहीन अंक [Garbage Value] प्रदर्शित करता है।

(v) realloc()–realloc() फंक्शन का उपयोग ऐरे का आकार पुनः निर्धारित करने के लिए किया जाता है। मान लीजिए इस प्रोग्राम के आरम्भ में एक ऐरे का आकार 5 निर्धारित कर चुके हैं और हम उसमें 5 Value संग्रहित भी कर चुके हैं लेकिन अब हम ऐरे में संग्रहित मानों को नष्ट किये बिना ऐरे का आकार बढ़ाना चाहते हैं तो इसके लिए realloc() फंक्शन का उपयोग किया जाता है। realloc() फंक्शन भी दो आरग्यूमेंट लेता है जिनमें पहला आरग्यूमेंट मेमोरी को reference करने वाला प्वॉइंटर तथा दूसरा आरग्यूमेंट ऐरे का आकार बढ़ाने के बाद उसकी कुल bytes की संख्या को दर्शाता है।

इसे आरग्यूमेंट के रूप में शून्य देने पर realloc() फंक्शन free() फंक्शन की तरह कार्य करता है; अर्थात् ऐरे का आकार शून्य हो जाता है और ऐरे को आवंटित मेमोरी मुक्त हो जाती है। यदि realloc() द्वारा ऐरे का आकार बढ़ जाता है तो यह फंक्शन फिर एक बार void प्वॉइंटर जारी करता है और अगर ऐरे का आकार नहीं बढ़ पाता तो यह NULL प्वॉइंटर जारी करता है।

उदाहरण–निम्न उदाहरण में हमने calloc() फंक्शन के द्वारा एक इन्टीजर ऐरे का आकार 5 निर्धारित किया है तथा बाद में realloc फंक्शन के द्वारा उसका आकार बढ़ाकर 7 किया है–

```
#include<stdio.h>
#include<stdlib.h>
#include<conio.h>
int main()
```

Scanned by CamScanner

```
{
int *ptr;
int i; int n=0;
clrscr();
ptr=calloc(5, sizeof(int));
if(ptr!=NULL)
    {
        for(i=0; i<5; i++)
            {
            n=n+2;
            ptr[i]=n;
        }
ptr=realloc(ptr, 7*sizeof(int));
if(ptr !=NULL)
    {
    for(i=5; i<7; i++)
        {
            n=n+2;
            ptr[0]=n;
        }
for(i=0; i<7; i++)
    {
    printf("ptr[%d]=%d\n", i, ptr[i]);
    }
realloc(ptr, 0);
return0;
}
else
    {
    printf("\n Not enough memory\n")
    return1;
    }
}
    else
    {
    printf("\n Not enough memory\n");
    return;
    }
}
```

Output :

```
ptr[0]=1
ptr[1]=3
ptr[2]=5
ptr[3]=7
ptr[4]=9
ptr[5]=11
ptr[6]=13
```

उपरोक्त प्रोग्राम में हमने पहले ऐरे का आकार 5 रखा है तथा बाद में उसे बढ़ाकर 7 किया है। ❑

Scanned by CamScanner

प्रश्न 2 – प्वॉइंटर्स की क्या आवश्यकता है? इसके विशेष उपयोगिता को उदाहरण सहित समझाइए

What is the need of pointers? What are their special applications? Explain with example.

(Jan. 2014, 16; June 2014, 15)

अथवा (Or)

प्वॉइंटर्स क्या हैं? उचित C भाषा प्रोग्राम द्वारा समझाइए।

What are Pointers? Explain with suitable 'C' Language Program. **(June 2013, 15)**

अथवा (Or)

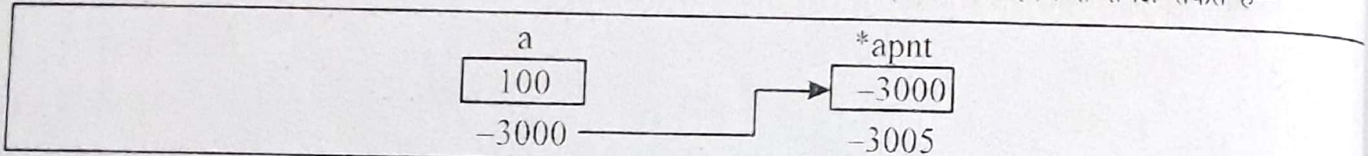
'C' में प्वॉइंटर क्या है? मेमोरी आवंटन में इसके उपयोग का उल्लेख करें।

What are pointers in 'C' ? Write it's usage in memory allotment.

(Dec. 2006; Jan. 2011, 16)

उत्तर–प्वॉइंटर एक बहुत महत्वपूर्ण वेरिएबल होता है जो हमें अनेक विशेष सुविधाएं देता है। खास तौर पर यह C को, बहुत शक्तिशा प्रोग्रामिंग लैंग्वेज बनाता है।

प्वॉइंटर एक ऐसा वेरिएबल होता है जो किसी अन्य वेरिएबल के ऐड्रेस को संग्रहित करता है तथा प्वॉइंटर के माध्यम से हम जो मान इनपुट करते हैं या जो भी गणनाएं करते हैं वे वास्तव में मुख्य वेरिएबल में ही संग्रहित होती हैं; अर्थात् प्वॉइंटर वेरिएबल उस वेरिए के प्रतिनिधि की तरह कार्य करता है जिसका ऐड्रेस प्वॉइंटर में रखा गया है। इसे हम निम्न उदाहरण से समझ सकते हैं–



वेरिएबलों के नाम	a	*apnt
वेरिएबलों में संग्रहित मान	100	–3000
वेरिएबलों के मेमोरी ऐड्रेस	–3000	–3005

यहाँ a एक सामान्य वेरिएबल है तथा *apnt एक प्वॉइंटर वेरिएबल है जो a को पॉइन्ट कर रहा है, अर्थात् a में एक मान 100 संग्रहि है, जबकि *apnt में वेरिएबल a का ऐड्रेस (–3000) संग्रहित है। अब अगर हम प्वॉइंटर वेरिएबल में 50 जोड़ें (*apnt=*apnt+50) त यह वास्तव में a वेरिएबल के मान में जुड़ेगा अर्थात् a का मान अब 150 हो जाएगा, क्योंकि *apnt यहाँ वेरिएबल a का ही प्रतिविधि अर्थात् *apnt में मेमोरी ऐड्रेस –3000 संग्रहित है, अतः *apnt में हम 50 जोड़ते हैं तो वह वास्तव में ऐड्रेस –3000 के मान अर्थात् 10 में ही जुड़ता है।

प्वॉइंटर के साथ दो ऑपरेटर * तथा & उपयोग किये जाते हैं। ऑपरेटर प्वॉइंटर वेरिएबल के साथ लगाया जाता है; जैसे–ऊपर हमने प्वॉइंटर वेरिएबल को *apnt लिखा है। यह ऑपरेटर प्वॉइंटर के माध्यम से मुख्य वेरिएबल को मान एक्सेस करने के लिए लगाया जान आवश्यक है; जैसे–ऊपर जब हमने *apnt के माध्यम से 50 जोड़ा है तब इसे हमने *apnt=*apnt+50 लिखा है, यही अगर हम बिन * के apnt=apnt+50 लिखें तो a के मान में 50 नहीं जुड़ेगा।

& ऑपरेटर एक ऐड्रेस ऑपरेटर है। अर्थात् मुख्य वेरिएबल का ऐड्रेस प्वॉइंटर में संग्रहित करने के लिए इसका उपयोग किया जात है।

सामानय वेरिएबल के समान ही प्वॉइंटर वेरिएबल को भी प्रयोग से पहले डिक्लेयर करना पड़ता है। इसका प्रारूप निम्नानुसार होता है–

```
data-type *Pointer name;
```

उदाहरण–

```
int *apnt;
```

मुख्य वेरिएबल का ऐड्रेस प्वॉइंटर वेरिएबल में निम्नानुसार डालते हैं–

```
int a;
int *apnt;
apnt= &a;
```

यहाँ a मुख्य वेरिएबल है जिसके साथ & ऑपरेटर लगाकर उसका ऐड्रेस एक्सेस किया गया है तथा प्वॉइंटर apnt में डाला गया है।

Scanned by CamScanner

प्वॉइंटर की उपरोक्त सम्पूर्ण प्रक्रिया तथा वेरिएबल का मान और ऐड्रेस करने की विधि निम्न प्रोग्राम में प्रस्तुत की गयी है–

```
#include<stdio.h>
#include<conio.h>
main()
{
      int a;
      int *p;
      p=&a;
      a=5;
      *p=*p+5;
      clrscr();
      printf("content of a:%d", a);
      printf("\n address of a:%d", &a);
      printf("\n content of p:%d", p);
      printf("\n content of a through p:%d", *p);
      printf("\n address of p:%d", &p);
```

Output :

```
content of a : 10
address of a : – 44
content of p : – 44
content of a through p : 10
address of p : – 42
```

स्पष्टीकरण–

(i) प्रोग्राम में हमने a में मूल रूप से 5 संग्रहित किया है लेकिन प्वॉइंटर p में +5 करने पर a का मान 10 हो गया है।

(ii) दूसरे printf में a के साथ & ऑपरेटर (&a) लेने पर a का मेमोरी ऐड्रेस & प्रदर्शित हो रहा है।

(iii) जिस प्रकार पहले printf में केवल a लेने पर a वेरिएबल में संग्रहित मान प्रदर्शित हो रहा है ठीक उसी प्रकार तीसरी printf में केवल p लेने पर p में संग्रहित मान अर्थात् a का ऐड्रेस (– 44) प्रदर्शित हो रहा है।

(iv) चौथे printf में a के साथ * ऑपरेटर लेने पर (*p), वह a के प्रतिनिधि के रूप में कार्य करता है और a का मान (10) प्रदर्शित करता है।

(v) पाँचवें printf में p के साथ & ऑपरेटर लेने पर (&p), वह अपना स्वयं का मेमोरी ऐड्रेस प्रदर्शित करता है।

प्वॉइंटर के लाभ (Advantages of Pointer)

(i) C लैंग्वेज में return स्टेटमेन्ट के द्वारा हम फंक्शन से केवल एक वैल्यू ही रिटर्न कर सकते हैं लेकिन प्वॉइंटर की सहायता से हम कॉलिंग फंक्शन के अनेक वेरिएबलों के मान बदल सकते हैं।

(ii) ऐरे को घुमाने की अपेक्षा प्वॉइंटर को घुमा कर ऐरे को मैनिप्यूलेशन करने पर मैनिप्यूलेशन की क्षमता तथा गति कई गुणा बढ़ जाती है।

(iii) वेरिएबलों के मेमोरी ऐड्रेसों को आसानी से एक्सेस किया जा सकता है।

(iv) पॉइन्टर्स का उपयोग करने से प्रोग्राम के क्रियान्वयन की गति बढ़ जाती है तथा प्वॉइंटर की सहायता से डायनामिक मेमोरी अलोकेशन भी किया जा सकता है।

(v) लिंक्ड लिस्ट संरचना तैयार करने के लिए प्वॉइंटर आवश्यक है, क्योंकि लिंक्ड लिस्ट के प्रत्येक नोड में अगले नोड का ऐड्रेस भी रखा जाता है। ❑

प्रश्न 3 – प्वॉइन्टर की धारणा को Function Argument की तरह उल्लेख करने के लिए 'C' में एक प्रोग्राम लिखें।

Write a C program to explain the concept of Pointer with respect to Function argument.

(Dec. 2006)

उत्तर–उपरोक्त प्रोग्राम अग्रांकित पृष्ठ पर समझाया गया है–

Scanned by CamScanner

```
#include<stdio.h>
#include<conio.h)
void main()
    {
    int a, b;
    void swp(int *,int *);
    clrscr();
    a=5;
    b=10;
    printf("values of variable before swap");
    printf("\n a=%d, b=%d", a, b);
    swp(&a, &b);
    printf("\n values of variables after swap");
    printf("\n a=%d, b=%d", a, b);
}
void swp(int *x, int *y)
    {
        int tmp;
        tmp=*x;
        *x=*y;
        *y=tmp;
    }
```

Output :

```
values of variables before swap
    a=5,    b=10
values of variables after swap
    a=10,   b=5
```

प्रश्न 4 – *GNU C Compiler के लाभ एवं हानियां Turbo C Compiler की तुलना में बताइए।*

What are the advantages and disadvantages of GNU C Compiler over Turbo C Compiler?

(June 2011)

अथवा (Or)

GNU C Compiler कें Salient Features बताइए ।

Explain salient features of GNU compiler. *(June 2011)*

उत्तर–C कम्पाइलर, C भाषा में लिखे गये सोर्स प्रोग्राम को मशीन भाषा में कनवर्ट करता है। C भाषा में प्रोग्राम Human Language में लिखते हैं। कम्प्यूटर Human Language को नहीं समझता है क्योंकि कम्प्यूटर केवल मशीन कोड 0 तथा 1 को ही समझता है, अतः Human Language में लिखे गये सोर्स प्रोग्राम तथा मशीन कोड के मध्य एक ऐसे Mediator का होना आवश्यक है, जो Human Language तथा मशीन लेंग्वेज दोनों को समझता हो। यह Mediator कम्प्यूटर सिस्टम में C कम्पाइलर के रूप में होता है। यह कम्पाइलर Human Language में लिखे गये सोर्स प्रोग्राम को मशीन कोड में कनवर्ट करता है तथा प्रोसेसिंग के पश्चात् आउटपुट को Human Language में स्क्रीन पर प्रदर्शित करता है; जैसे–दो व्यक्ति आपस में बातें करते हैं परन्तु पहला व्यक्ति सिर्फ अंग्रेजी भाषा को जानता है तथा दूसरा व्यक्ति हिन्दी भाषा को जानता है, ये दोनों व्यक्ति आपस में बातें करते हैं परन्तु एक-दूसरे की भाषा को नहीं समझते हैं, अतः एक ऐसे तीसरे व्यक्ति की आवश्यकता होगी जो हिन्दी तथा अंग्रेजी दोनों भाषाओं को जानता हो, तो इस तीसरे व्यक्ति के माध्यम से ये दोनों व्यक्ति आपस में बातें कर सकते हैं। यहां पर तीसरा व्यक्ति कम्पाइलर की तरह कार्य करता है। मुख्य रूप से प्रयोग किये जाने वाले कम्पाइलर दो प्रकार के होते हैं–

1. GNU C कम्पाइलर 2. Turbo C कम्पाइलर

(1) GNU C कम्पाइलर –GNU C कम्पाइलर को ANSI (American National Standards Institute) स्टैण्डर्ड के द्वारा सत्यापित किया गया है। GNU C कम्पाइलर में पहले से ही परिभाषित लाइब्रेरी फक्शन्स होते हैं, जिन्हें ANSI स्टैण्डर्ड ने सत्यापित किया है। इन फंक्शन्स का प्रयोग हम सोर्स प्रोग्राम में सीधे तौर पर करते हैं। प्रोग्राम की लिंकिंग के समय ये फंक्शन सोर्स प्रोग्राम के साथ लिंक होते हैं।

GNU C कम्पाइलर लाइब्रेरी में विभिन्न प्रकार की Header files होती हैं, इन Header files की सूचना C प्रोग्राम के कम्पाइलेशन के समय प्रयोग की जाती है; जैसे "stdio.h" हैडर फाइल इनपुट, आउटपुट की प्रक्रिया को पूरा करने के लिये Input/Output लाइब्रेरी फंक्शन्स को उपलब्ध कराती है। इसी तरह "string.h" हैडर फाइल प्रोसेसिंग को पूरा करने के लिये लाइब्रेरी फंक्शन्स उपलब्ध कराती है। GNU C कम्पाइलर के प्रमुख फीचर्स निम्नलिखित हैं–

1. GNU C कम्पाइलर ANSI स्टैण्डर्ड कमेटी से सत्यापित है।

2. GNU C कम्पाइलर में स्टैण्डर्ड लाइब्रेरी फंक्शन्स परिभाषित होते हैं, जो Input/Output, मैमोरी मैनेजमेन्ट, स्ट्रिंग मैनीपुलेशन इत्यादि ऑपरेशन्स की प्रक्रिया को पूरा करते हैं।

3. GNU C कम्पाइलर में परिभाषित स्टैण्डर्ड लाइब्रेरी फंक्शन्स ANSI स्टैण्डर्ड कमेटी के द्वारा सत्यापित हैं।

4. GNU C कम्पाइलर लाइब्रेरी में विभिन्न प्रकार की हैडर फाइल्स होती हैं जिनका प्रयोग C भाषा में Source प्रोग्राम में किया जाता है। इन Header files के फंक्शन्स को सीधे तौर पर सोर्स-प्रोग्राम में प्रयोग किया जाता है।

5. GNU C कम्पाइलर ANSI स्टैण्डर्ड कमेटी से सत्यापित हैं, इसलिये इसे ANSI C भी कहते हैं। ANSI C को मुख्य रूप से UNIX या LINUX ऑपरेटिंग सिस्टम पर प्रयोग करते हैं।

(2) Turbo C कम्पाइलर– Turbo C कम्पाइलर Borland कम्पनी के द्वारा बनाया गया है। Turbo C कम्पाइलर में Header files तथा Library function समाहित होते हैं। Header files को C भाषा में सोर्स प्रोग्राम सम्मिलित करते हैं तथा लाइब्रेरी फंक्शन्स को सीधे तौर पर प्रोग्राम में प्रयोग करते हैं। GCC कम्पाइलर ANSI स्टैण्डर्ड से सत्यापित है तथा POSIX को भी सपोर्ट करता है इसलिये इसे ANSI C कहते हैं। ANSI C का प्रयोग UNIX/LINUX ऑपरेटिंग सिस्टम पर करते हैं; अर्थात् जब UNIX या LINUX ऑपरेटिंग सिस्टर पर C भाषा में प्रोग्राम बनाते हैं तो इसे ANSI C कहते हैं। इसी तरह जब DOS या Windows ऑपरेटिंग सिस्टम पर C भाषा का प्रोग्राम बनाते हैं तब इसे Turb C कहते हैं। Turbo C कम्पाइलर का निर्माण Borland कम्पनी ने किया है इसलिये इसे Borland C भी कहते हैं। Turbo C कम्पाइलर में IDE को Integreated Development Environment कहते हैं। यह प्रोग्रामर के लिये C भाषा में सोर्स प्रोग्राम लिखने के लिये GUI एनवायरनमेन्ट उपलब्ध कराता है; जैसे–Menu bar में उपलब्ध सभी Menus तथा Submenus सोर्स प्रोग्राम में शीघ्रता से इंटरफेस प्रदान करते हैं, जिन्हें Mouse के द्वारा Interact कर सकते हैं।

यह फीचर IDE फीचर होता है। Turbo C कम्पाइलर IDE विण्डो को प्रदर्शित करता है; अर्थात् इसमें प्रोग्राम बनाने के लिये मेन्यू बार उपलब्ध होता है, इस मेन्यू बार पर Menus प्रदर्शित होते हैं। इन Menus के अन्दर Submenus प्रदर्शित होते हैं जो कमाण्ड के रूप में कार्य करते हैं। इन Menus के साथ Interaction करके प्रोग्रामर आसानी से प्रोग्राम बना सकते हैं। जब हम DOS या Windows ऑपरेटिंग सिस्टम में C भाषा में प्रोग्राम बनाते हैं तो इसे Turbo C कहते हैं।

Turbo C कम्पाइलर की तुलना में GNU C कम्पाइलर के प्रमुख लाभ निम्नलिखित होते हैं–

1. GNU C कम्पाइलर ANSI स्टैण्डर्ड कमेटी से सत्यापित है।

2. GNU C कम्पाइलर विभिन्न प्रकार के Predefined लाइब्रेरी फंक्शन्स को उपलब्ध कराता है।

3. GNU C कम्पाइलर में विभिन्न प्रकार की Predefined हैडर फाइल्स उपलब्ध होती हैं।

4. GNU C कम्पाइलर UNIX/LINUX ऑपरेटिंग सिस्टम के POSIX फीचर को सपोर्ट करता है।

5. GNU C कम्पाइलर प्रोग्राम को कमांड प्रॉम्प्ट पर कम्पाइल करता है।

Turbo C कम्पाइलर की तुलना में GNU C कम्पाइलर की मुख्य हानियां निम्नलिखित हैं–

1. GNU C कम्पाइलर IDE फीचर को सपोर्ट नहीं करता है।

2. GNU C कम्पाइलर DOS या Windows ऑपरेटिंग सिस्टम पर कार्य नहीं करता है, इसे विशेष रूप सें UNIX/LINUX के लिये बनाया गया है। ❑

प्रश्न 5 – *Static और Dynamic memory allocation को समझाइए।*

Explain Static and Dynamic memory allocation. ***(Jan. 2012, 13, 16; June 2011, 12, 15)***

अथवा (Or)

स्टेटिक एंव डायनेमिक मेमोरी एलोकेशन में अन्तर स्पष्ट कीजिए।

What is the difference between static and dynamic memory allocation explain.

(June 2014; Jan. 2013, 15)

उत्तर– Static Memory Allocation तथा Dynamic Memory Allocation का प्रयोग वेरियेबल को मैमोरी एलोकेट करने के लिये

Scanned by CamScanner

किया जाता है। जब किसी वेरियेबल को डिक्लेयर किया जाता है तब इसकी डाटा टाइप के अनुसार इस वेरियेबल को मैमोरी एलोकेट की जाती है; जैसे—int a वेरियेबल का नाम है, इसकी डाटा टाइप int है, यह वेरियेबल 2 bytes की मैमोरी एलोकेट करता है। इसी प्रकार int a[5], को डिक्लेयर करते हैं, यहां पर a एक इंटीजर टाइप की ऐरे है। इस ऐरे को 12 byte की मैमोरी एलोकेट की जाती है। इंटीजर टाइप की ऐरे की प्रथम subscript 0 होती है। अतः इस ऐरे में 6 subscripts होती हैं, प्रत्येक subscript 2 bytes की मैमोरी को घेरती है। अतः इस ऐरे को 12 bytes की मैमोरी एलोकेट की जाती है। यह ऐरे 12 bytes की मैमोरी को घेरता है, यदि इसमें 2 वैल्यूज स्टोर करते हैं लेकिन यह 12 bytes की मैमोरी को घेरती है, इसे स्टेटिक मैमोरी एलोकेशन कहते हैं।

Static Memory Allocation तथा Dynamic Memory Allocation के मध्य प्रमुख अन्तर निम्नलिखित है—

Static Memory Allocation	Dynamic Memory Allocation
1. Static Memory Allocation में कम्पाइलेशन के समय ही वेरियेबल को मैमोरी एलोकेट की जाती है; जैसे— किसी ऐरे का Size 5 है तो इसे 12 bytes की मैमोरी एलोकेट की जाती है, क्योंकि ऐरे के साइज का First subscript 0 से शुरू होता है। इसमें यदि हम कम एलीमेन्ट भी डालते हैं तब भी इसमें 12 bytes की मैमोरी ही एलोकेट होती है।	1. Dynamic Memory Allocation में प्रोग्राम के रन टाइम पर मैमोरी एलोकेट की जाती है; जैसे— एक प्रोग्राम में किसी कॉरपोरेशन के Employees की लिस्ट को स्टोर करते हैं। जब लिस्ट में Employees के नाम को Add करते हैं, तो List का Size बढ़ जाता है तथा जब लिस्ट में से Employees को हटाते हैं, तो लिस्ट का Size कम हो जाता है। जब लिस्ट का Size बढ़ता है तो हमें List को मैमोरी एलोकेट करने की जरूरत होती है। इस तरह की स्थिति को हैण्डल करने के लिये Dynamic Data Structure का प्रयोग करते हैं।
2. कम्पाइल टाइम पर एक ऐरे में Element को Specify करने की आवश्यकता होती है।	2. Malloc, Calloc, realloc तथा free चार प्रमुख फंक्शन्स का प्रयोग Dynamic Memory Allocation में किया जाता है।

प्रश्न 6 – ANSI 'C' एवं Turbo 'C' compiler के बीच में comparison दीजिए।

Give the comparison between ANSI 'C' and Turbo 'C' compiler.

उत्तर—

Difference between ANSI 'C' and Turbo 'C'

ANSI 'C'	Turbo 'C'
1. ANSI 'C' American National Standard Institute द्वारा सत्यापित है।	1. Turbo 'C' को Borland कम्पनी ने विकसित किया है इसलिये इसे Borland C Compiler भी कहते हैं।
2. ANSI 'C' को 'GNU 'C' कम्पाइलर के नाम से भी जाना जाता है। इसमें विभिन्न प्रकार की Header files तथा लाइब्रेरी फंक्शन्स समाहित हैं जो ANSI स्टैण्डर्ड के द्वारा सत्यापित है।	2. Turbo 'C' कम्पाइलर में भी विभिन्न प्रकार की Header Files तथा लाइब्रेरी फंक्शन्स समाहित हैं, जिसका प्रयोग Source Program में सीधे तौर पर किया जाता है।
3. ANSI 'C' कम्पाइलर में विभिन्न प्रकार के लाइब्रेरी फंक्शन्स हैं जो विभिन्न प्रकार के Operations; जैसे—Input/Output, String Manipulation इत्यादि की प्रक्रिया को पूरा करते हैं; जैसे—stdio.h हैडर फाइल Input/Output Operation को पूरा करते हैं।	3. Turbo 'C' कम्पाइलर में भी विभिन्न प्रकार के लाइब्रेरी फंक्शन्स हैं, जो विभिन्न प्रकार के Operations; जैसे—Input/Output String Manipulation इत्यादि की प्रक्रिया को पूरा करते हैं; जैसे—string.h स्ट्रिंग मैनीपुलेशन के Operation को पूरा करते हैं।
4. ANSI 'C' Compiler Unix/LINUX ऑपरेटिंग सिस्टम के POSIX फीचर को सपोर्ट करता है।	4. Turbo 'C' कम्पाइलर का प्रयोग DOS या Windows ऑपरेटिंग सिस्टम में C भाषा के प्रोग्राम को कम्पाइल करने के लिये किया जाता है।
5. ANSI 'C' Compiler Text Editor में Source Program की Editing करने के लिये Commands को उपलब्ध कराता है।	5. Turbo 'C' Compiler IDC फीचर को उपलब्ध कराता है इससे C भाषा में Source Program में Editing को आसानी से सम्पन्न किया जाता है।

Scanned by CamScanner

प्रश्न 7 – *GCC में C के program को आप कैसे compile करते हैं?*
How do you compile C program using GCC?

अथवा (Or)

C Program का Compilation gcc से समझाइए।
Explain compilation of C program using gcc. ***(June 2011)***

उत्तर–GCC का प्रयोग करके C प्रोग्राम को कम्पाइल करना (Compilation of C Program Using GCC)–GCC को विस्तृत रूप से GNU C Compiler कहते हैं। GNU C Compiler ANSI (American National Standard Institute) स्टैण्डर्ड के द्वारा सत्यापित किया गया है। GNU C Compiler का प्रयोग UNIX या LINUX में C भाषा में क्रिएट किये गये प्रोग्राम को कम्पाइल करने के लिये किया जाता है। LINUX में प्रयोग की जाने वाली C भाषा को ANSI C कहते हैं। क्योंकि Linux में GNU C Compiler का प्रयोग C भाषा के प्रोग्राम को Compile करने के लिये किया जाता है। GNU C Compiler को ANSI स्टैण्डर्ड ने सत्यापित किया है इसलिये इसे ANSI C कहते हैं। GCC का प्रयोग C प्रोग्राम को कम्पाइल करने के लिये निम्नलिखित Steps को Follow करते हैं–

Step 1 : Linux में Command Prompt पर Vi editor के माध्यम से फाइल को क्रिएट करते हैं, इसके लिये अग्रांकित रूप में कमाण्ड लिखते हैं–

```
vi abc.c
```

इस कमाण्ड को लिखने के पश्चात् Enter Key Press करते हैं, तो Text editor खुल जाता है। abc.c फाइल का नाम है यहां पर c फाइल का Extension Name है। इस Text Editor में C भाषा की कमाण्ड्स का प्रयोग करके अपनी आवश्यकतानुसार प्रोग्राम लिखते हैं तथा फाइल को Save करते हैं।

Step 2 : File को Save करने के लिये Esc Key Press करते हैं तथा shift Key के साथ Colon (:) Key Press करते हैं और wq Type करते हैं। File Save हो जाती है। इसके पश्चात् Enter Key को Press करते हैं और आप Linux Prompt पर आ जाते हैं।

Step 3 : अब Program फाइल को Compile कराते हैं। File को कम्पाइल करने के लिये Linux Prompt पर Compiler के नाम के साथ C भाषा की File का नाम निम्नलिखित रूप में लिखते हैं–

```
#cc abc.c
```

इसके पश्चात् Enter Key Press करते हैं, प्रोग्राम की कम्पाइलेशन प्रक्रिया होती है, यदि प्रोग्राम में Errors नहीं है तो Program Successfully कम्पाइल हो जाता है, यदि प्रोग्राम में Errors हैं तो ये Errors डिबगिंग विण्डो में प्रदर्शित होती हैं। इन Errors को सही करके पुनः प्रोग्राम की कम्पाइलेशन करते हैं। प्रोग्राम कम्पाइलेशन का Message स्क्रीन पर प्रदर्शित होता है।

Step 4 : प्रोग्राम की कम्पाइलेशन करने के पश्चात् प्रोग्राम को क्रियान्वित करते हैं। प्रोग्राम को क्रियान्वित करने के लिये Linux Prompt पर निम्नलिखित कमाण्ड को Type करके Enter Key Press करते हैं–

```
./a.out
```

इसके पश्चात् स्क्रीन पर प्रोग्राम की आउटपुट प्रदर्शित होती है।

GCC कमाण्ड लाइन विकल्प (GCC Command Line Options)—GCC को विस्तृत रूप से GNU C Compiler कहते हैं। GNU C Compiler को ANSI (American National Standard Interface) स्टैण्डर्ड ने सत्यापित किया है। GNU C Compiler में विभिन्न प्रकार की हैडर Files तथा लाइब्रेरी फंक्शन्स हैं जिनका प्रयोग C भाषा में Source प्रोग्राम में सीधे तौर पर करते हैं। GNU C Compiler की Header Files तथा लाइब्रेरी फंक्शन्स को ANSI स्टैण्डर्ड ने सत्यापित किया है।

GNU C Compiler UNIX/LINUX ऑपरेटिंग सिस्टम के POSIX Feature को सपोर्ट करता है इसलिये GNU C Compiler को मुख्य रूप से UNIX/LINUX ऑपरेटिंग सिस्टम में प्रयोग किया जाता है क्योंकि इसे ANSI स्टैण्डर्ड ने सत्यापित किया है इसलिये इसे ANSI C कहते हैं। GNU C Compiler के द्वारा C भाषा के प्रोग्राम को कमाण्ड लाइन के द्वारा कमाण्ड्स का प्रयोग करके कम्पाइल तथा क्रियान्वित करते हैं इसलिये GNU C Compiler के विभिन्न प्रकार के Command Line Options हैं। Program को Compilation तथा Execution के लिये GCC की तरह ही TIGCC का भी प्रयोग कर सकते हैं। TIGCC सामान्यतः Preprocessing, Compilation, Assembly तथा Linking को एक Step में ही पूरा करता है। TIGC में GCC की तुलना में मुख्य अन्तर यह है कि यह हमेशा Assembly या Object Files को Overwrite करता है; जैसे–यदि एक File test.c को Compile किया गया है तो फाइल test.s तथा test.o Overwrite करेंगी। इसके

Scanned by CamScanner

अतिरिक्त TIGCC में GCC की तुलना में एक मुख्य अन्तर यह है कि TIGCC की लाइब्रेरी 'Lib' Subfolder में Install होती है। इसके Command Line Options निम्नलिखित हैं–

(i) **- pack varname–** यह option कम्पाइलेशन के समय Generate की गयी File को Compress करता है।

(ii) **-q–** यह Option Assembling तथा Linking के समय File को ऐसे ही रखता है। इस Option की आवश्यकता File के कम्पाइलेशन के समय नहीं होती है।

(iii) **-vo–**यह Option TIGCC को यह बताता है कि आउटपुट केवल Commands के द्वारा प्रदर्शित होगी।

(iv) **-g–**यह Option GCC को डिबगिंग के लिये वापस बुलाता है।

(v) **-ar–**यह Option एक प्रोग्राम के स्थान पर archive को क्रिएट करता है।

प्रश्न 8 – प्वॉइंटर क्या है? इसका उपयोग करके स्ट्रिंग कैसे हेंडिल करोगे? उदाहरण द्वारा समझाइए!

What is pointer? How do you handle string using pointer? Explain pointer. ***(Jan. 2015)***

उत्तर–स्ट्रिंग एण्ड पॉइन्टर (String and Pointer)–जिस तरह इंटीजर ऐरे को पॉइन्टर Notation के द्वारा एक्सेस करते हैं इसी तरह कैरेक्टर ऐरे या स्ट्रिंग को भी पॉइन्टर Notation के द्वारा एक्सेस कर सकते हैं; जैसे–

```
char name[5] = "raja"
```

यहां पर Name स्ट्रिंग है इसका साइज 5 है। इसका मैमोरी में Representation निम्नलिखित रूप में होता है–

r	a	j	a	\0
1001	1002	1003	1004	1005

स्ट्रिंग के एड्रेस में हमेशा 1 Number का अंतर होता है क्योंकि स्ट्रिंग कैरेक्टर टाईप ऐरे है। कैरेक्टर टाईप डाटा टाईप 1 byte की मैमोरी प्रयोग करता है। यहां पर यदि स्ट्रिंग के प्रथम सबस्क्रिप्ट का एड्रेस 1001 है तो अगली सबस्क्रिप्ट का एड्रेस 1002 होगा।

यदि किसी स्ट्रिंग को पॉइन्टर Notation के द्वारा डिक्लेयर करते हैं तो इसके Size को डिक्लेयर करने की आवश्यकता नहीं होती है। प्रोग्राम के क्रियान्वयन के समय जितने कैरेक्टर्स Input करते हैं यह स्ट्रिंग उतना Size ही Use करती है। निम्नलिखित प्रोग्राम में स्ट्रिंग को पॉइन्टर के द्वारा प्रदर्शित किया गया है–

उदाहरण :

```
#include<stdio.h>
#include<conio.h>
     void main()
     {
         char *name;
         printf("enter name\n");
         scanf("%s", name);
         printf("name is %s ", *name);
         getch();
     }
```

यहां पर name पॉइन्टर टाइप का स्ट्रिंग वेरियेबल है। इसका Size डिक्लेयर नहीं किया गया है। इस प्रोग्राम को क्रियान्वयन के समय जितने कैरेक्टर्स को इस वेरियेबल में Input किया जाता है यह उनकी मैमोरी को ही Use करेगा।

आप एक स्ट्रिंग को दूसरी स्ट्रिंग में एसाइन (Assign) नहीं कर सकते हैं, जबकि आप एक char Pointer को दूसरे char Pointer एसाइन (Assign) कर सकते हैं; जैसे कि निम्नलिखित उदाहरण में प्रदर्शित किया गया है–

उदाहरण :

```
#include<stdio.h>
#include<conio.h>
```

Scanned by CamScanner

```
void main()
{
    char str1[ ]="raja";
    char str[10];
    char *s="rajaram";
    char *q;
    str2=str1 /*wrong */
    q=s; /* Right */
    getch();
}
```

यहां पर str1 तथा str2 कैरेक्टर ऐरे हैं। str1 की Value यदि str2 में एसाइन करते हैं तो यह statement error प्रदर्शित करता है। s तथा q दोनों Pointer string वेरियेबल हैं। s को स्ट्रिंग एसाइन की गयी है यदि s की स्ट्रिंग वैल्यू q को एसाइन करते हैं तो यह statement क्रियान्वित हो जाता है, अर्थात् एक पॉइन्टर स्ट्रिंग को दूसरे पॉइन्टर स्ट्रिंग वेरियेबल में एसाइन कर सकते हैं, जबकि साधारण स्ट्रिंग वेरियेबल को नहीं कर सकते हैं।

एक स्ट्रिंग को परिभाषित करने के पश्चात् इस स्ट्रिंग को दूसरे कैरेक्टर्स एसाइन नहीं कर सकते हैं, लेकिन char पॉइन्टर वेरियेबल में ऐसा कर सकते हैं। निम्नलिखित उदाहरण से यह कथन स्पष्ट होता है–

उदाहरण :

```
#include<stdio.h>
#include<conio.h>
    void main()
    {
        char str1[ ]="raja";
        char *a="rama";
        str1="ram" /*wrong*/
    a="ram" /*right*/
    getch();
}
```

❑

प्रश्न 9 – *Array एवं Pointer के बीच उदाहरण सहित अंतर बताइये।*

What is the difference between Array and Pointers? Explain with example.

उत्तर–ऐरे (Array)–किसी भी प्रोग्रामिंग भाषा में वेरियेबल का महत्वपूर्ण योगदान होता है, क्योंकि वेरियेबल के माध्यम से ही वैल्यूज को मैमोरी में स्टोर करते हैं। किसी भी वेरियेबल में स्टोर की गयी वैल्यू इस वेरियेबल के डाटा टाईप पर निर्भर करती है; जैसे–किसी वेरियेबल की डाटा टाईप इंटीजर है तो इस वेरियेबल में इंटीजर वैल्यू स्टोर की जाती है या किसी वेरियेबल की डाटा टाईप कैरेक्टर है तो इस वेरियेबल में कैरेक्टर वैल्यू स्टोर की जाती है। यदि हम एक इंटीजर टाईप की डाटा टाईप का वेरियेबल डिक्लेयर करते हैं तो इसमें एक Single इंटीजर टाईप का Element स्टोर होता है, यदि हम इसी वेरियेबल में दूसरा Element स्टोर करते है तो इस वेरियेबल में दूसरा Element स्टोर होता है, परन्तु पहला Element हट जाता है। इस तरह हमें दूसरी वैल्यूज को स्टोर करने के लिये दूसरा वेरियेबल डिक्लेयर करने की आवश्यकता होती है। इस तरह यदि वैल्यूज की संख्यायें बढ़ती जायेंगी तो वेरियेबल्स की संख्यायें भी बढ़ जायेंगी और प्रोग्रामर के रूप में इतने वेरियेबल्स को याद रखना आसान नहीं होगा। इसका दूसरा तरीका यह है कि इन वेरियेबल्स के स्थान पर subscript वेरियेबल का प्रयोग करना चाहिये, जिसमें एक Common वेरियेबल का नाम डिक्लेयर करते हैं तथा इसमें जितनी वैल्यूज को स्टोर करना है, उतना Size subscript में डिजाइन करते हैं, जिस टाईप की वैल्यूज को स्टोर करना है उसकी डाटा टाईप को इस subscript वेरियेबल के साथ डिक्लेयर करते हैं; जैसे–int a[10];

इस टाईप के subscript वेरियेबल को ऐरे कहते हैं। यहां पर int डाटा टाईप है, a ऐरे का नाम है तथा 10 इस ऐरे का साइज है, इसका अर्थ यह है कि a एक Common नेम है जिसका 10 subscript है, प्रत्येक subscript में किसी इंटीजर Element को स्टोर कर सकते हैं जिसका नाम क्रमशः a[0], a[1] ....a[9] है। अन्तिम subscript का नाम a[9] होगा, क्योंकि इंटीजर टाईप की ऐरे की प्रथम subscript 0 से शुरू होती है।

Scanned by CamScanner

निम्नलिखित उदाहरण में ऐरे को स्पष्ट रूप से समझाया गया है–

उदाहरण–

```
#include<stdio.h>
#include<conio.h>
    void main()
    {
    int a [5];
    int i, j;
printf("enter any five elements\n");
for(i = 0; i<5; i ++)
    {
    scanf("%d", & a{i});
    }
printf("five elements are\n");
for(i =0; i < 5; i + 1)
    {
    printf("%d", a[i];
    }
getch();
}
```

पॉइण्टर्स (Pointers) (Jan. 2016)–C भाषा में वैल्यूज को स्टोर करने के लिये प्रयोग करता है। किसी भी वेरियेबल की के उस वेरियेबल की Datatype निर्भर करती है। जिस टाईप की डाटा टाईप डिक्लेयर की है उसी वैल्यू को वेरियेबल में स्टोर करते हैं; जैसे-

```
int a = 10;
```

यहां पर a वेरियेबल का नाम है तथा int इस वेरियेबल की Datatype और इस वेरियेबल को 10 वैल्यू एसाइन की गयी है। इस एड्रेस को स्टोर करने के लिये जिस वेरियेबल को प्रयोग करते हैं उसे Pointer कहते हैं। निम्नलिखित उदाहरण में पॉइन्टर का प्रयोग किया गया है–

```
#include<stdio.h>
#include<conio.h>
    void main()
    {
    int a,*b;
    a = 10;
    b = &a;
    printf("%d", a);
    printf("%4 is address of a", b);
    printf("%d is value of address of a", *b);
    getch();
}
```

इस प्रयोग में a इंटीजर टाईप का वेरियेबल है, इस वेरियेबल में 10 वैल्यू को एसाइन किया है। a वेरियेबल का एड्रेस b पॉइन्टर के वेरियेबल में स्टोर किया है। पॉइण्टर वेरियेबल को * (asteric) चिह्न के साथ डिक्लेयर करते हैं।

Scanned by CamScanner

प्रश्न 10 – Structure pointer क्या है?

What is Structure pointer? **(June 2014)**

अथवा (Or)

C भाषा में एक प्रोग्राम लिखिए जो Pointers within structure को explain करें।

Write a C program to illustrate pointers within structure. **(June 2014)**

उत्तर–स्ट्रक्चर, विभिन्न प्रकार के डाटा टाईप के वेरियेबल्स का संग्रहण होता है। स्ट्रक्चर के Inside डिक्लेयर किये गये वेरियेबल्स को Member Variables कहते हैं। इन Member Variables में स्ट्रक्चर के माध्यम से एक्सेस करते हैं। जब किसी स्ट्रक्चर के Object के Members को एक्सेस करते हैं तो डॉट ऑपरेटर (.) का प्रयोग करते हैं। क्योंकि इस स्ट्रक्चर का ऑब्जेक्ट साधारण ऑब्जेक्ट है। यदि किसी स्ट्रक्चर के ऑब्जेक्ट को पॉइन्टर टाईप का डिक्लेयर करते हैं, तब स्ट्रक्चर के Members को एक्सेस करने के लिये डॉट ऑपरेटर (.) के स्थान पर Indirect Operator (→) का प्रयोग करते हैं। निम्नलिखित उदाहरण में इस बिन्दु को स्पष्ट रूप से दर्शाया गया है–

```
#include<stdio.h>
#include<conio.h>
struct emp
    {
    char name[10];
    int age;
    }*e1;
void main()
    {
    print f("Enter Emp name \n");
    scanf("%s", e1→name);
    printf("Enter Emp age \n");
    scanf("%d", e1→age);
    printf("Emp Name is %s", e1→name);
    printf("Emp age is %d", e1→age);
    getch();
    }
```

इस प्रोग्राम में e1 आब्जेक्ट पॉइन्टर ऑब्जेक्ट है। ❑

प्रश्न 11 – तीसरे वैरियेबल का प्रयोग किये बिना दो वैरियेबलों को स्वैप करने का C में एक प्रोग्राम लिखिए।

Write a program in C to swap two variables without using third variable. **(June 2012)**

उत्तर–

```
#include<stdio.h>
#include<conio.h>
void main()
{
int a,b;
clrscr();
printf("enter first number\n");
scanf("%d",&a);
printf("enter second number\n");
```


Scanned by CamScanner

```
scanf("%d",&b);
printf("before swapping values");
printf("%d%d",a,b);
a=a+b;
b=a-b;
a=a-b;
printf("after swapping values");
printf("%d%d",a,b);
getch();
}
```

प्रश्न 12 – पॉइंटर का प्रयोग करते हुए दो स्ट्रिंग को स्टोर करने के लिए C में एक प्रोग्राम लिखिए और उन्हें तीसरे वैरियेबल में कॉनकेटीनेट कीजिए।

Write a program in C to store two strings using pointer and concatenate them in third variable. ***(June 2012)***

उत्तर–

```
#include<stdio.h>
#include<conio.h>
#include<string.h>
void main( )
{
char *s1[10], *s2[10], *s3[10];
clrscr( );
printf("enter first string");
scanf("%s",*s1);
printf("enter second string");
scanf("%s",*s2);
*s3=strcat (*s1, *s2);
printf("Concatenated string is %s", *s3);
getch( );
}
```

प्रश्न 13 – पॉइंटर अर्थमेटिक क्या है? उचित उदाहरणों द्वारा समझाइए।

What is pointer arithmetic? Explain with suitable examples. ***(Jan. 2013, 16; June 2015)***

उत्तर–पॉइन्टर अर्थमेटिक (Pointer Arithmetic)–जिस तरह साधारण वेरियेबल्स पर अर्थमेटिक ऑपरेशनस (Arithmetic Operations) को ऑपरेट करते हैं, इसी तरह पॉइन्टर वेरियेबल्स पर भी अर्थमेटिक ऑपरेशनस (Arithmetic Operations) को ऑपरेट (Operate) कर सकते हैं। अग्रांकित प्रोग्राम के द्वारा पॉइन्टर अर्थमेटिक (Pointer Arithmetic) को स्पष्ट रूप से समझ सकते हैं–

उदाहरण :

```
#include<stdio.h>
#include<conio.h>
```

Scanned by CamScanner

```
void main( )
  {
    int a, *i;
    float b, *j;
    char c, *k;
    a=10;
    b=10.5;
    c='a';
    printf("value of a=%d\n", a);
    printf("value of b=%f\n", b);
    printf("value of c=%c\n", c);
    i=&a;
    j=&b;
    k=&c;
    printf("value of i=%u\n", i);
    printf("value of j=%u\n", j);
    printf("value of k=%u\n", k);
    i++;
    j++;
    k++;
    printf("New value of i=% u\n", i);
    printf("New value of j=% u\n", j);
    printf("New value of k=% u\n", k);
    getch( );
  }
```

Program : Pointer Arithmetic

इस प्रोग्राम की आउटपुट निम्नलिखित रूप में प्राप्त होती है–

```
Value of a=10
Value of b=10.5
Value of c=a
Value of i=2025
Value of j=2040
Value of k=2050
New value of i=2027
New value of j=2044
New value of k=2051
```

इस प्रोग्राम के अनुसार पहले a, b तथा c की वैल्यू क्रमशः 10, 10.5 तथा a प्रदर्शित होती है। इसके पश्चात् i, j तथा k की वैल्यू प्रदर्शित की गई हैं। i में a का एड्रेस j में b का एड्रेस तथा k में c का एड्रेस वैल्यू के रूप में प्रदर्शित होता है। यहां पर i, j तथा k में इंक्रीमेन्ट ऑपरेटर का प्रयोग किया गया है, तीनों वेरियेबल्स पॉइन्टर वेरियेबल्स हैं। जब इन तीनों वेरियेबल्स पर इंक्रीमेन्ट ऑपरेटर को ऑपरेट किया जाता है तो इन वेरियेबल्स के डाटा टाईप के साइज के अनुसार एड्रेस वैल्यू में इंक्रीमेन्ट होता है; जैसे int का Size 2 bytes है तो i की वैल्यू 2025 के स्थान पर 2027 हो जाती है। Float का Size 4 bytes है तो j की वैल्यू 2040 के स्थान पर 2044 हो जाती है। इसी तरह char का Size 1 byte है तो k की वैल्यू 2050 के स्थान 2051 हो जाती है।

यहां पर i, j और k के output value change भी हो सकती है। क्योंकि i, j और k का address change भी हो सकता है।

Scanned by CamScanner

प्रश्न 14 – पॉइंटर्स का उपयोग करते हुए दो वैरिएबल की values को swap करने के लिए C प्रोग्राम लिखिए।

Write a program in C to swap the values of two variables using pointers.

(Jan. 2013; June 2015)

उत्तर–

```
#include<stdio.h>
#include<conio.h>
int swap (int*, int*);
void main()
{
int a=2,b=3;
 clrscr();
printf("\nvalues before swapping \na=%d\nb=%d",a,b);
swap(&a,&b);
printf("\nvalues after swapping\na=%d\nb=%d",a,b);
getch();
}
swap(int *x,int *y)
{
int t;
t=*x;
*x=*y;
*y=t;
}
```

प्रश्न 15 – डायनेमिक मैमोरी अलोकेशन के क्या लाभ हैं ?

What are the benefits of dynamic memory allocation ?

(June 2013)

उत्तर–Dynamic memory allocation के फायदें निम्नलिखित हैं–

1. C में Dynamic memory allocation का मुख्य फायदा flexibility है, अर्थात् किसी variable को memory allocate करने के लिए advance में size की आवश्यकता नहीं पड़ती है।
2. जब प्रोग्राम को compile करते समय यह निश्चित नहीं हो पाता है कि उसे कितनी मैमोरी की आवश्यकता है? ऐसी स्थिति में Dynamic memory allocation के द्वारा उसे जितनी memory की आवश्यकता होती है उतना ले सकते हैं जबकि static memory allocation में ऐसा नहीं होता है।
3. Dynamic memory allocation की आवश्यकता विशेष रूप से Array में होती है। जब Array का आकार निश्चित नहीं होता है।
4. Dynamic data structure के द्वारा रन टाईम पर Data items को adding, deleting या rearranging किया जा सकता है।
5. C में मुख्य रूप से चार Dynamic memory allocation Function (malloc, calloc, free and realloc) का प्रयोग प्रोग्राम के क्रियान्वयन के समय मैमोरी को allocate तथा मैमोरी को free करने के लिये किया जाता है।

प्रश्न 16 – एक C program लिखिए जो mouse का pointer और उसकी लोकेशन को बताये।

Write a C program which displays mouse pointer and position of pointer. **(Jan. 2014)**

उत्तर–Mouse का Pointer और उसकी location को बताने के लिये C Program निम्नलिखित है–

```
#include<dos.h>
```

Scanned by CamScanner

```
#include<stdio.h>
void main()
{
   union REGS i,o;
   int x,y,k;
   //show mouse pointer
   i.x.ax=1;
   int86(0x33,&i,&o);
   while(!kbhit())          //its value will false when we hit key in the key board
     {
       i.x.ax=3;          //get mouse position
       x=o.x.cx;
       y=o.x.dx;
       clrscr();
       printf("(%d , %d)",x,y);
       delay(250);
       int86(0x33,&i,&o);
     }
   getch();
}
```

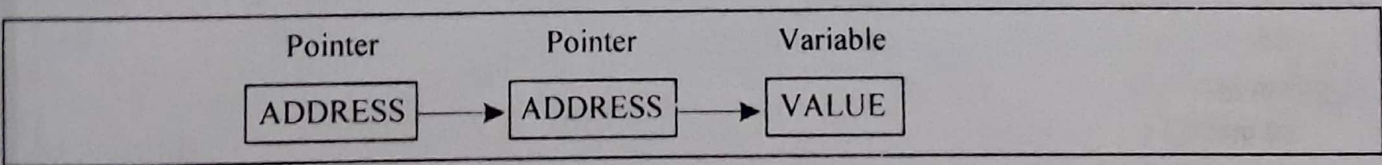
❑

प्रश्न 17 – ***Multilevel of pointers से आप क्या समझते हैं? Program बनाकर सोदाहरण समझाइये।***

What do you mean by multilevel of pointers? Explain with example by making program.

(Jan. 2014)

उत्तर–Multilevel of Pointers–C में हम Pointers को int, char, structure आदि के लिए प्रयोग कर सकते है। इसके अलावा C हमें एक pointer के द्वारा दूसरे pointer को भी point करने की अनुमति प्रदान करता है। यह एक pointers की chain भी हो सकती है अर्थात हम एक pointer से दूसरे, दूसरे से तीसरे, तीसरे से चौथे, ...ऐसे कितने भी pointers को जोड़ सकते हैं। आमतौर पर एक pointer को जोड़ सकते हैं। आमतौर पर एक pointer उस variable के address को रखता है जिसे वह point करता है। पर जब हम pointer to pointer को परिभाषित करते हैं, तो पहला pointer दूसरे pointer का address तथा दूसरा pointer तीसरे pointer या actual variable के address को रखता है। नीचे दिये चित्र में यह दर्शाया गया है–



एक pointer to pointer variable को इसके सामने एक extra * लगाकर denote करते हैं जैसे–

```
int **var;
```

उदाहरण–

```
#include <stdio.h>
int main ()
{
  int  var;
  int  *ptr;
```

```
    int **pptr;
    var = 3000;
    /* take the address of var *
    ptr = &var;
    /* take the address of ptr using address of operator & */
    pptr = &ptr;
    /* take the value using pptr */
    printf("Value of var = %d\n", var );
    printf("Value available at *ptr = %d\n", *ptr );
    printf("Value available at **pptr = %d\n", **pptr);
    getch();
    return 0;
}
```

जब हम ऊपर दिये गये code को compile करके Run करते हैं तो निम्न Output प्राप्त होता है–

आउटपुट–

```
Value of var = 3000
Value available at *ptr = 3000
Value available at **pptr = 3000
```

प्रश्न 18 – निम्न pointers को समझाइये।

Explain the following pointers :

Null pointers, Dangling pointers, Near pointers, Far pointers. ***(Jan. 2014)***

उत्तर–Null Pointer–Null pointer वह pointer होते हैं, जो किसी object को point नहीं करते हैं। Null pointer segment के base address को point करते हैं; जैसे–

(a) int *ptr = (char*) 0;

(b) Float *ptr = (float*) 0;

(c) int *ptr = NULL

उदाहरण–

```
# include<string.h>
# include<stdio.h>
    int main( ) {
    char *str = Null;
Strcpy(str, " C-pointer");
Printf(% s", Str);
return 0;
}
```

Output :

(Null)

Dangling Pointer–यह वह pointer होते हैं, जो उन memory location को point करते हैं जो अब allocate नहीं है। Dangling pointers की वजह से कभी-कभी program crash भी हो जाते हैं। जिन प्रोग्रामों से Dangling pointer होता है। वह ज्यादातर

Scanned by CamScanner

छोटे input पर तो कार्य करते हैं पर बड़े और complex input से वह अक्सर fail हो जाते हैं।

उदाहरण–

```
typedef char *Foo;
Foo new foo(char *X)
{
    Foo tmp = new char[Strlen(x) + 1];
    Strcpy(tmp, x);
    delete[ ] x;
    return tmp;
}
```

यहां पर new foo द्वारा एक dangling pointer create हो गया क्योंकि जिस string को वह point कर रहा था उसे delete कर दिया गया है।

Near Pointer–एक ऐसा pointer जो केवल 64KB डाटा segment को ही point कर सकता है, Near pointer कहलाता है। इसका मतलब Near pointer डाटा segment जैसे graphic video memory, text video memory आदि को एक्सेस नहीं कर सकता। Near pointer का size 2 byte होता है। हम pointer से पहले near keyword का प्रयोग करके near pointer का निर्माण करते हैं।

उदाहरण–

```
#include<stdio.h>
int main( )
{
    int x = 25
    int near *ptr;
    ptr = &x;
    printf("%d", size of ptr);
    return 0;
}
```

Output :

2

Far Pointer–वह pointer जो पुरी RAM मेमोरी को एक्सेस कर सकते हैं। अर्थात जो RAM मेमोरी के सभी 16 segments को एक्सेस कर सकते हैं Far Pointers कहलाते हैं। Far Pointer का size 4 byte या 32 bit होता है।

उदाहरण–

```
#include<stdio.h>
int main( )
{
    int x = 10;
    int far *ptr;
    ptr = &x;
    printf("%d", sizeof ptr);
    return 0;
}
```

Output :

4

Scanned by CamScanner

प्रश्न 19 – यूजर डिफाइन्ड फंक्शन के साथ पॉइण्टर कैसे उपयोग करेंगे? 'C' भाषा के प्रोग्राम द्वारा समझाइये।

How do you use pointer with user defined functions? Explain with suitable 'C' program.

(Jan. 2015)

उत्तर–जब किसी यूजर डिफाइन्ड फंक्शन में main( ) फंक्शन से आर्ग्यूमेण्ट को वेरियेबल के नाम के स्थान पर वेरियेबल के एड्रेस के रूप में Pass करते हैं तो वेरियेबल के नाम के साथ **एड्रेस ऑफ ऑपरेटर (&)** को **Actual Argument** के रूप में Pass करते हैं तथा जब Formal Argument के रूप में Argument को यूजर डिफाइन्ड फंक्शन में रिसीव करते हैं तो वेरियेबल को **Pointer** वेरियेबल के रूप में लिखते हैं। निम्नलिखित उदाहरण में फंक्शन में पॉइन्टर का प्रयोग स्पष्ट किया गया है–

उदाहरण :

```
#include<stdio.h>
#include<conio.h>
void disp(int *i, int *j);
void main( )
{
  int a, b;
  printf("enter two numbers\n");
  scanf("%d%d", &a, &b);
  disp(&a, &b);
  getch( );
}
void disp(int *i, int *j)
{
int k;
k = *i;
*i = *j;
*j = k;
printf("%d%d", * i, *j);
}
```

Program : Using Pointer with user defined Function

इस उदाहरण में a तथा b के एड्रेस को main( ) फंक्शन से disp( ) फंक्शन में Arguments के रूप में Pass किया गया है। इन वेरियेबल्स के एड्रेस को disp( ) फंक्शन में क्रमशः Pointer Variable i तथा j में Refer किया है। इन दोनों की वैल्यूज को आपस में तीसरे वेरियेबल k की सहायता से Interchange करके प्रदर्शित किया है।

Scanned by CamScanner

UNIT-II

प्रश्न 1 – स्ट्रीम (stream) क्या है? 'C' में स्टैण्डर्ड स्ट्रीम क्या है?

What is stream ? What is standard stream in 'C' ? *(Jan. 2016)*

उत्तर–'C' लैंग्वेज की इनपुट/आउटपुट प्रणाली यूजर को हार्डवेयर डिवाइसेस एक्सेस करने के लिए इन्टरफेस प्रदान करती है, यह इन्टरफेस हार्डवेयर डिवाइसेस की एब्स्ट्रेक्ट प्रस्तुति होती है जो हार्डवेयर डिवाइस को एक लॉजिकल फाइल (लॉजिकल इकाई) के रूप में मानकर चलता है। डिवाइसेस की यह एब्स्ट्रेक्ट प्रस्तुति स्ट्रीम कहलाती है।

'C' लैंग्वेज की इनुपट/आउटपुट प्रणाली प्रिन्टर्स, मॉनीटर, की-बोर्ड, हार्डडिस्क आदि अनेक प्रकार की डिवाइसेस के साथ कार्य करती है, ये सभी डिवाइसेस एक-दूसरे से भिन्न होती हैं। लेकिन बफर्ड फाइल प्रणाली इन सभी अलग-अलग तरह की डिवाइसेस को एक लॉजिकल इकाई में बदल देती है, जिन्हें स्ट्रीम कहते हैं। सभी स्ट्रीम एक ही तरह के कार्य करते हैं अतः इन्हें सम्भालना आसान होता है; अर्थात् डिवाइसेस को स्ट्रीम के रूप में उपयोग करने पर 'C' लैंग्वेज एक ही फंक्शन से सभी डिवाइसेस का उपयोग कर सकती है।

स्ट्रीम दो प्रकार के होते हैं–

1. टैक्स्ट स्ट्रीम (Text Stream)
2. बाइनरी स्ट्रीम (Binary Stream)

1. टैक्स्ट स्ट्रीम (Text Stream) :

टैक्स्ट स्ट्रीम अक्षरों का क्रमबद्ध समूह होती है। आंतरिक रूप से 'C' टैक्स्ट को व्यवस्थित रखने के लिए एक 'न्यू लाइन कैरेक्टर' (New Line Character) का उपयोग करती है। जो कम्प्यूटर को यह बतलाता है कि प्रत्येक नई लाइन कहां से आरम्भ हो रही है। यह New Line Character अंतिम लाइन के लिए ऑप्शनल होता है। टैक्स्ट स्ट्रीम में वातावरण की आवश्यकता के अनुसार अक्षरों का अनुवाद भी हो सकता है। उदाहरण के लिए, New Line Caeeiage return को Linefeed Pair में बदला जा सकता है। जिसके कारण लिखे गये अक्षरों की संख्या डिवाइस पर उपलब्ध अक्षरों की संख्या के समान नहीं होती।

2. बाइनरी स्ट्रीम (Binary Stream) :

बाइनरी स्ट्रीम बाइट्स का क्रमबद्ध समूह होता है, जिसमें वन-टू-वन समानता होती है; अर्थात् यहाँ अक्षरों का अनुवाद नहीं होता। यहाँ वास्तविक रूप से लिखे गए बाइट्स की संख्या बाह्य डिवाइस पर बाइट्स की संख्या के समान ही होती है। यह बाइट्स का एक क्रम होता है जिसमें फाइल की समाप्ति या रिकॉर्ड की समाप्ति को दर्शाने के लिए कोई फ्लैग नहीं होता। उसमें फाइल के आकार के द्वारा ही उसकी समाप्ति का निर्धारण होता है। ❑

प्रश्न 2 – Opening फाइलों की विभिन्न अवस्थाओं (File Opening Modes) की व्याख्या करो।

Explain different file opening modes. *(June 2005, Jan. 2011)*

उत्तर–जब हम किसी फाइल को खोलते हैं तो हमें अनिवार्य रूप से यह बताना आवश्यक होता है कि हमें एक नई फाइल का निर्माण करना है या एक पहले से बनी हुई फाइल को पढ़ना है या एक पहले से बनी हुई फाइल में और डाटा जोड़ना है। ये सभी अलग-अलग कार्य प्रकार फाइल मोड (File Mode) कहलाते हैं; अर्थात् हम फाइल को जिस मोड में खोलते हैं केवल वही कार्य कर सकते हैं।

फाइल को खोलने के लिए fopen() फंक्शन का उपयोग किया जाता है। जिसमें आरग्यूमेंट के रूप में फाइल का नाम तथा फाइल मोड दिया जाता है। फाइल मोड यह बतलाता है कि फाइल को खोलकर हमें उसमें क्या कार्य करना है?

ये फाइल आपेनिंग मोड निम्ननुसार होते हैं–

मोड (Mode)	उपयोग (Use)
"w"	यह मोड नई फाइल बनाने के लिए उपयोग किया जाता है, जिसमें हम नयी फाइल बनाकर उसमें डाटा लिख सकते हैं। अगर दिये जा रहे नाम से पहले से फाइल मौजूद हो तो वह फाइल Overwrite हो जाती है तथा फिर से नयी फाइल बन सकती है।
"r"	यह एक read मोड है; अर्थात् हम इस मोड में पहले से बनी हुई फाइल को खोलकर केवल पढ़ सकते हैं, उसमें कुछ लिख नहीं सकते। अगर दिये जा रहे नाम से फाइल उपलब्ध नहीं है तो यह मोड NULL वेल्यू return करता है।

Scanned by CamScanner

"a"	"a" मोड का अर्थ अपेन्ड (append) मोड होता है। इस मोड में हम पहले से बनी फाइल में अंत में नया डाटा जोड़ सकते हैं। अगर दिये गये नाम से फाइल उपलब्ध नहीं है तो यह मोड नई फाइल बना देता है और अगर उपलब्ध फाइल किसी कारण से खुल नहीं रही है तो यह NULL वैल्यू return करता है।
"w+"	इस मोड में हम पहले से बनी हुई फाइल को खोलकर डाटा को पढ़ सकते हैं, उसमें संशोधन कर सकते हैं तथा नया डाटा भी जोड़ सकते हैं।
"r+"	इस मोड में हम पहले से उपलब्ध फाइल को खोलकर पढ़ सकते हैं तथा उसमें संशोधन कर सकते हैं। अगर उपलब्ध फाइल नहीं खुलती है तो यह मोड NULL वैल्यू return करता है।
"a+"	इस मोड में हम पहले से उपलब्ध फाइल को पढ़ सकते हैं तथा उसमें नया डाटा जोड़ सकते हैं। इस मोड में पहले से लिखे हुए डाटा में संशोधन नहीं किया जा सकता।

प्रश्न 3 – बिट वाइज़ ऑपरेटर (Bitwise Operator) को समझाइये।

Explain Bitwise Operator. ***(June 2003, 2005)***

उत्तर–बिटवाइज ऑपरेटर बिट स्तर पर डाटा मैनीपुलेशन के लिए प्रयुक्त किये जाते हैं। ये ऑपरेटर वेरिएबल में संग्रहित मान के बिट्स को दायीं या बायीं ओर Shift करने के लिए तथा डाटा पर लॉजिकल ऑपरेशन (AND, OR Compliment) करने के लिए उपयोग किये जाते हैं। इस आधार पर बिटवाइज ऑपरेटर्स के कार्य निम्नानुसार होते हैं–

(i) Left Shift तथा Right Shift

(ii) लॉजिकल ऑपरेशन AND, OR तथा XOR

(iii) One's Complement

उपरोक्त कार्यों के लिए उपयोग किये जाने वाले विभिन्न बिटवाइज ऑपरेटर निम्नलिखित हैं–

<table>
<tr><th>ऑपरेटर</th><th>कार्य</th><th>विवरण</th></tr>
<tr><td><<</td><td>Left Shift</td><td>बाइनरी अंकों को बायीं ओर Shift करना</td></tr>
<tr><td>>></td><td>Right Shift</td><td>बाइनरी अंकों को दायीं ओर Shift करना</td></tr>
<tr><td>&</td><td>Bitwise AND</td><td>दो बाइनरी मानों पर AND ऑपरेशन</td></tr>
<tr><td>|</td><td>Bitwise OR</td><td>दो बाइनरी मानों पर OR ऑपरेशन</td></tr>
<tr><td>^</td><td>Bitwise XOR</td><td>दो बाइनरी मानों पर XOR ऑपरेशन</td></tr>
<tr><td>~</td><td>One's Complement</td><td>एक बाइनरी मान का One's Complement</td></tr>
</table>

प्रश्न 4 – fopen() फंक्शन का प्रयोग समझाइये।

Explain the use of fopen() function. ***(June 2014; Jan. 2012, 13, 15, 16)***

उत्तर–किसी डाटा फाइल पर कार्य करने के लिए पहले उसे खोलना आवश्यक होता है, और उसके लिए fopen() फंक्शन का उपयोग किया जाता है। इसके लिए पहले FILE कीवर्ड के साथ एक फाइल प्वॉइंटर बनाया जाता है, इसके बाद fopen() फंक्शन द्वारा return किया जाने वाला फाइल का मेमोरी ऐड्रेस फाइल प्वॉइंटर में संग्रहित किया जाता है। fopen() फंक्शन में दो आरग्यूमेंट दिये जाते हैं जिनमें पहला आरग्यूमेंट डाटा फाइल का नाम होता है तथा दूसरा आरग्यूमेंट फाइल का मोड होता है। फाइल का मोड यह दर्शाता है कि फाइल किस प्रकार खुलती है; अर्थात् खुलने पर फाइल को केवल read किया जा सकेगा या उसमें डाटा write भी किया जा सकेगा–यह मोड के द्वारा ही तय होता है।

fopen() फंक्शन की संरचना निम्नानुसार होती है–

```
FILE*filepointername;
filepointername=fopen("filename", "mode");
```

उदाहरण–

```
FILE *fp
fp=fopen("stud.data", "w");
```

Scanned by CamScanner

यहाँ लिया गया w मोड नई फाइल बनाने के लिए उपयोग किया जाता है। "w" के स्थान पर अन्य मोड भी उपयोग किया जा सकता है। एक फाइल निम्नलिखित तरीके से खोली जानी चाहिए, ताकि फाइल के खुलने में कुछ तकनीकी कठिनाई हो तो हमें पता चल सके–

```
FILE *fp
if(fp=fopen("Stud.dat", "w"))==NULL)
    {
    printf("Error in file opening");
    exit(1);
    }
```

प्रश्न 5 – *fclose() फंक्शन का प्रयोग प्रोग्राम के माध्यम से समझाइये।*

Explain the use of fclose() function with the help of a program.

(June 2011, 14; Jan. 2012, 13, 15, 16)

उत्तर–जब हम अपने प्रोग्राम में किसी डाटा फाइल का उपयोग करते हैं तो प्रोग्राम समाप्त होने पर उसे बंद कर देना चाहिए, क्योंकि फाइल बंद कर देने पर उसके द्वारा उपयोग किये जा रहे मेमोरी ब्लॉक स्वतंत्र हो जाते हैं जो किसी दूसरे प्रोग्राम, द्वारा उपयोग किये जा सकते हैं। यहां फाइल बंद करने का एक और महत्वपूर्ण कारण यह भी है कि एक बार में अधिकतम कितनी फाइलें खुली रखी जा सकती हैं, यह संख्या कम्पाइलर और ऑपरेटिंग सिस्टम द्वारा निश्चित है, अब अगर हम अपनी डाटा फाइलों को कार्य पूरा करने के बाद भी बंद ना करें तो कुल खुली हुई फाइलों की संख्या अनावश्यक रूप से बढ़ती जाएगी। इसके अतिरिक्त फाइल के बंद करने पर उसका डाटा हार्डडिस्क में सुरक्षित भी हो जाता है।

डाटा फाइल में बंद करने के लिए fclose() फंक्शन का उपयोग किया जाता है। यह फंक्शन स्वयं ही फाइलों को बंद करते समय डाटा को हार्डडिस्क में सुरक्षित कर देता है तथा मेन मैमोरी से हटा देता है। इस फंक्शन का सिन्टेक्स निम्नानुसार होता है–

```
fclose(filepointername);
```

मान लीजिए हमारा फाइल प्वॉइंटर का नाम fp है, तो यह निम्नानुसार लिखा जाएगा–

```
fclose(fp);
```

यह फंक्शन भी एक इन्टीजर मान वापस करता है। अगर फाइल सफलतापूर्वक बंद हो जाती है तो यह 0 (शून्य) return करता है। इसे हम निम्न प्रोग्राम के द्वारा समझ सकते हैं–

```
#include<stdio.h>
void main()
    {
        FILE *fp;
        int status;
    if((fp=fopen("stud.dat", "r")= = NULL)
    {
        printf("Error in file opening");
        exit(1);
    }
    else
    {
        printf("File opened successfully");
    }
/* closing the file and getting return value */
status=fclose(fp);
if(status= =0)
```

Scanned by CamScanner

```
printf("\n file closed successfully");
else
      printf("Error in file closing");
}
```

प्रश्न 6 – *fputc() तथा fgetc() फंक्शनों को विस्तार से समझाइये।*

Explain in detail of fputc() and fgetc() functions. ***(June 2003, 05, 14; Jan. 2013, 16)***

उत्तर–putc() का अर्थ put character तथा getc() का अर्थ get character होता है तथा इनके आगे लगा f अक्षर फाइल शब्द से लिया गया है, अर्थात् fuptc() फंक्शन फाइल में एक-एक अक्षर लिखने तथा fgetc() फंक्शन फाइल से एक-एक अक्षर पढ़ने के लिए उपयोग किया जाता है।

इन फंक्शनों के सिन्टेक्स निम्नानुसार होते हैं–

```
fputc(variable, filepointer);
```

उदाहरण–

```
fputc(ch, fp);
```

यहाँ ch एक करेक्टर टाइप वेरिएबल है जिसका मान fp फाइल प्वॉइंटर के माध्यम से फाइल में लिखा जाएगा, इस प्रकार fputc() फंक्शन एक बार में एक अक्षर फाइल में लिखता है।

```
variable=fgetc(filepointer);
```

उदाहरण–

```
ch=fgetc(fp);
```

यहां fgetc() फंक्शन फाइल प्वॉइंटर के माध्यम से फाइल से एक अक्षर पढ़कर return करेगा जिसे एक कैरेक्टर टाइप वेरिएबल ch में लेकर हम उसे डिस्प्ले करवा सकते हैं।

यहाँ हम लूप के माध्यम से इन फंक्शनों के द्वारा एक से अधिक अक्षर write या read करवा सकते हैं। fputc() फंक्शन का लूप में उपयोग करने पर हम लूप समाप्त करने के लिए कोई एक विशेष अक्षर (Special Character) निर्धारित कर देते हैं, जैसे ही हम निर्धारित अक्षर टाइप करते हैं लूप समाप्त हो जाता है। इसी तरह fgetc() फंक्शन लूप के द्वारा फाइल समाप्त होने तक फाइल में अक्षर पढ़ सकता है, जिसे हम लूप में EOF के द्वारा देखते हैं।

प्रोग्राम में getch() फंक्शन के द्वारा एक कैरेक्टर टाइप वेरिएबल में की-बोर्ड से अक्षर इनपुट कराया जाता है तथा इस वेरिएबल को fputc() फंक्शन में आरग्यूमेंट के रूप में दिया जाता है। इसी तरह fgetc() फंक्शन द्वारा return किये जा रहे अक्षर को एक कैरेक्टर टाइप वेरिएबल में लेकर उसे putchar() फंक्शन के द्वारा मॉनिटर पर डिस्प्ले किया जाता है।

ऊपर वर्णित सभी बिन्दुओं को हम निम्न प्रोग्राम के द्वारा समझ सकते हैं–

```
#include<stdio.h>
#include<conio.h)
void main()
    {
        FILE *fp;
        char ch;
     fp=fopen("First.txt", "w");
clrscr();
     printf("Enter character $ for termination");
     ch=getch();
     while(ch!='$')
     {
```

Scanned by CamScanner

```
        fputc(ch, fp);
        ch=getch();
    }
    fclose(fp);
    fp=fopen("first.txt", "r"); /*open file in read mode*/
    printf("\n contents of files are : \n");
    do
    {
        ch=fgetc(fp);
        putchar(ch);
} while(ch!=feof());
fclose(fp);
}
```

प्रश्न 7 – *fputs() तथा fgets() फंक्शनों की उपयोगिता समझाइये।*

Explain the usage of fputs() & fgets() functions. ***(Jan. 2004; June 2014)***

उत्तर– fputs() फंक्शन फाइल में स्ट्रिंग को संग्रहित करने तथा fgets() फंक्शन फाइल में स्ट्रिंग को read करने के लिए उपयोग किया जाता है। इन फंक्शनों का उपयोग हम फाइल के अतिरिक्त अन्य स्ट्रीम पर अक्षरों को write या read करने के लिए भी कर सकते हैं।

fgets() फंक्शन में आरग्यूमेंट के रूप में स्ट्रिंग की Length दी जाती है, यह फंक्शन उतने अक्षरों के समूह को एक स्ट्रिंग मानकर चलता है जितनी Length निर्दिष्ट की गई है। इन फंक्शनों के फॉर्मेट निम्नानुसार होते हैं–

fputs() :

```
fputs(string, filepointer/stream);
```

उदाहरण–

```
fputs(sta, fp);
fputs(sta, stdout);
```

यहाँ dta एक स्ट्रिंग ऐरे हैं जिसे पहले उदाहरण में फाइल प्वॉइंटर के द्वारा फाइल में लिखा गया है तथा दूसरे उदाहरण में उसे स्टैन्डर्ड आउटपुट स्ट्रीम अर्थात् मॉनिटर पर भेजा गया है।

fgets() :

```
fgets(string, length, filepointer);
```

उदाहरण–

```
fgets(sta, 513, fp);
```

fgets() फंक्शन द्वारा पढ़े जाने वाले अक्षर की अधिकतम लम्बाई 512 बाइट हो सकती है, अतः यहाँ हम अधिकतम Length 513 ले सकते हैं। यह Length आवश्यकतानुसार कम भी हो सकती है; जैसे–

```
fgets(sta, 50, fp);
```

इस उदहारण में फंक्शन, फाइल से एक बार में 50 अक्षर पढ़ेगा।

प्रोग्राम–निम्नलिखित प्रोग्राम में हमने दोनों फंक्शनों का उपयोग करते हुए एक फाइल से स्ट्रिंग read करके उसे दूसरी फाइल में write करवाया है–

```
#include<stdio.h>
#include<conio.h>
void main()
```

Scanned by CamScanner

```
{
    FILE *fp1;
    FILE *fp2;
    char sta[25];
    fp1=fopen("first.txt", "r");
    fp2=fopen("second.txt", "w");
    while(!feof(fp1))
    {
        fgets(sta, 25, fp1);
        printf("\n %s", sta);
        fputs(sta, fp2);
    }
        fclose(fp1);
        fclose(fp2);
}
```

प्रश्न 8 – *fseek()* फंक्शन को समझाइये।

Explain the fseek() function. ***(Jan. 2005, 12, 13, 14; June 2011)***

उत्तर– fseek() फंक्शन फाइल में निर्धारित बाइट पर फाइल प्वॉइन्ट को सेट करने के लिए प्रयुक्त होता है। इसका सिन्टैक्स निम्नानुसार है–

```
fseek(stream, displacement_ pointer, position);
```

1. stream– यहाँ Stream के रूप में फाइल प्वॉइन्टर ही लिया जाता है, जो फाइल को बतलाता है।

2. displacement– यह बाइट्स की वह संख्या होती है जो हमें वर्तमान स्थिति से आगे या पीछे की ओर स्किप (Skip) करता है। डिस्प्लेसमेंट पॉजिटिव या नेगेटिव हो जाता है।

3. pointer_position– यह प्वॉइन्टर की वर्तमान पोजिशन को सेट करता है। यह पोजिशन हम 0, 1 या 2 दे सकते हैं जहाँ BOF (Begining of File), 1 वर्तमान स्थिति तथा 2 EOF (End of File) को दर्शाता है।

मान लीजिए हमारे पास win.txt नामक एक फाइल है जिसमें एक स्ट्रिंग "Microsoft Windows" संग्रहित है। इस फाइल में हम w पर फाइल प्वॉइन्टर सैट कर Windows शब्द के पढ़ेंगे।

प्रोग्राम–

```
#include<stdio.h>
#include<conio.h>
void main()
    {
    FILE *fp;
    char ch;
    fp=fopen("win.txt", "r");
    fseek(fp, 10l, 0);
    while(!feof(fp))
    {
        ch=fgetc(fp);
        printf("%c", ch);
    }
    fclose(fp);
}
```

Output :

windows

उपरोक्त प्रोग्राम में fseek() फंक्शन फाइल प्वॉइन्टर को BOF से 10 अक्षर आगे सैट कर रहा है। इसके बाद हम लूप के द्वारा एक-एक अक्षर read करके डिस्प्ले करवा रहे हैं।

प्रश्न 9 – ftell() फंक्शन का प्रयोग समझाइये।

Explain the use of ftell() function. ***(Jan. 2013)***

उत्तर– ftell() फंक्शन फाइल में वर्तमान प्वॉइंटर पोजिशन का मान return करता है।

प्रारूप (Syntax) :

```
ftell(filepointer);
```

उदाहरण–

```
ftell(fp);
```

यहाँ fp एक फाइल प्वॉइंटर है।

प्रोग्राम– ftell() फंक्शन का उपयोग हम निम्नलिखित प्रोग्राम के द्वारा समझायेंगे–

```
#include<stdio.h>
#include<conio.h>
void main()
    {
        FILE *fp;
        char ch;
        fp=fopen("win.txt", "r");
        fseek(fp, 10l, 0)
        while(!feof(fp))
        {
        ch=fgetch(fp);
        printf("\n%c", ch);
        printf("\t%d", ftell(fp));
    }
        fclose(fp);
    }
```

Output :

```
w   11
i   12
n   13
d   14
o   15
w   16
s   17
```

प्रश्न 10 – 'C' में कमाण्ड लाइन आरग्यूमेंट से क्या समझते है? उचित उदाहरण देकर समझाइए।

What is meant by Command line argument in C? Explain with suitable example.

(Dec., 2006)

अथवा (Or)

कमाण्ड लाइन आरग्यूमेन्ट्स को समझाइये।

Explain Command line Arguments.

(June 2010)

उत्तर–सामान्यतः हम किसी भी प्रोग्राम को 'C' तथा C++ के एडिटर के मेन्यू में Run ऑप्शन के द्वारा ही क्रियान्वित करते हैं, लेकिन हम प्रोग्राम की executable फाइल बनाकर उसे डॉस प्रॉम्प्ट पर फाइल का नाम टाइप करके भी RUN कर सकते हैं। प्रोग्राम की executable फाइल का विस्तार नाम (Extention).exe होता है।

इस प्रकार .exe फाइल को डॉस कमाण्ड प्रॉम्प्ट पर फाइल का नाम टाइप करके RUN करते समय अगर हम फाइल के नाम के साथ आरग्यूमेंट भी दें तो ये आरग्यूमेंट, कमांड लाइन आरग्यूमेंट कहलाते हैं। ये आरग्यूमेंट मेन फंक्शन में लिए गए पैरामीटर में जाते हैं।

मान लीजिए हमने एक carg.c नामक प्रोग्राम बनाया जिसकी हमने carg.exe नाम से execulable फाइल बनाई। इस .exe फाइल को कमांड प्रॉम्प्ट पर RUN करवाते समय फाइल के नाम के साथ आरग्यूमेंट अग्रानुसार देंगे–

```
c:\>carg   file1.txt   file2.txt
```

यहाँ file1.txt तथा file2.txt दो आरग्यूमेंट हैं जो carg फाइल को दिये गये हैं।

एक 'C' प्रोग्राम में main() फंक्शन इस उद्देश्य के लिए दो विशेष पैरामीटर argc तथा argv के साथ लिखा जाता है। यह निम्नानुसार होता है–

```
void main(int argc, char *argv[ ])
    {
        ......................
        .....................
        statements;
    }
```

यहाँ argc एवं argve[ ] फॉर्मल आरग्यूमेंट हैं जो कमांड लाइन पर दिये गये आरग्यूमेंट को एकत्रित करने के लिए एक कार्यप्रणाली प्रदान करते हैं। जब प्रोग्राम RUN किया जाता है argc दिये गये आरग्यूमेंट्स की संख्या को एकत्रित करता है एवं argv कैरेक्टर टाइप के प्वॉइंटर के ऐरे को एकत्रित करता है। यहाँ argv[0] हमेशा .exe फाइल के नाम को प्रदर्शित करता है तथा बाकी आरग्यूमेंट argv[1], argv[2] आदि में जाते हैं। ऊपर हमने carg के साथ file1.txt तथा file2.txt दो आरग्यूमेंट लिये हैं अतः argc का मान 3 होगा, क्योंकि इसमें फाइल के नाम को मिलाकर कुल तीन स्ट्रिंग संग्रहित होंगे।

निम्न प्रोग्राम में main() फंक्शन में कमांड लाइन आरग्यूमेंट के प्रयोग को दर्शाया गया है–

```
#include<stdio.h>
#include<conio.h>
main(int argc, char *argv[ ])
    {
        int i;
        clrscr();
      printf("Number of command line arguments passed=%d", argc);
      printf("\n these arguments are:\n");
      for(i=0; i<argc; i++)
      printf("argv[%d]= %s/n" i, argv[ ]);
}
```

Output :

```
argv[0]=carg
argv[1]=file1.txt
argv[2]=file2.txt
```

प्रश्न 11 – विभिन्न File Type को समझाइये। File Operation के विभिन्न चरण बताइये।
Explain different File Type and steps of File Operation.

अथवा (Or)

File क्या होती है? इसके विभिन्न operations को समझाइए।
What is File? Explain the various operations of File. **(Jan. 2012, 16)**

अथवा (Or)

फाइल क्या है? विभिन्न प्रकार की फाइल्स की व्याख्या कीजिए।
What is file? Discuss various types of files. **(Jan. 2013)**

अथवा (Or)

फाइल क्या है? 'C' भाषा में उपयोग होने वाले विभिन्न फाईल्स का वर्णन कीजिए।
What is file? Discuss various types of files used in 'C' languge. **(Jan. 2015, 16)**

उत्तर–एक या एक से अधिक इन्सट्रक्शन (Instruction) के समूह को प्रोग्राम कहते हैं। ये Instructions किसी विशेष कार्य को कम्प्यूटर के द्वारा पूरा करने के लिये लिखी जाती है। जिस स्थान पर यह प्रोग्राम Instructions या Information सुरक्षित रखी जाती है, उसे फाइल कहते हैं।

विभिन्न फाइल टाइप (Various File Types)

फाइलें अनेक प्रकार की होती हैं; जैसे–प्रोग्राम फाइलें, डाटा फाइलें, टैक्स्ट फाइलें, ऑब्जेक्ट फाइलें, क्रियान्वयन फाइलें (executable files) आदि।

(i) **प्रोग्राम फाइलें**–जिन फाइलों में हम अपना प्रोग्राम कोड टाइप करके संग्रहित करते हैं, प्रोग्राम फाइलें कहलाती हैं। 'C' भाषा में बनाई जाने वाली प्रोग्राम फाइलों का विस्तार नाम ·C होता है।

(ii) **डाटा फाइलें**–जिन फाइलों में विभिन्न तरह के रेकॉर्ड; जैसे–कर्मचारियों के रेकार्ड, विद्यार्थियों के रेकॉर्ड आदि डाटा रखा जाता है, डाटा फाइलें कहलाती हैं। इन फाइलों का विस्तार नाम ·dat होता है।

(iii) **टेक्स्ट फाइलें**–जिन फाइलों में सामान्य टेक्स्ट रखा जाता है, वे टेक्स्ट फाइलें कहलाती हैं। इनका विस्तार नाम ·Txt होता है।

(iv) **ऑब्जेक्ट फाइलें**–'C' भाषा में बने प्रोग्राम को कम्पाइल करने पर कम्पालइर द्वारा बनाई जाने वाली फाइल ऑब्जेक्ट फाइल कहलाती हैं। इसका विस्तार नाम ·obj होता है।

(v) **क्रियान्वयन फाइलें**–विभिन्न प्रोग्रामों तथा सॉफ्टवेयरों के मशीन कोड जिनके द्वारा हम सॉफ्टवेयर को केवल Run कर सकते हैं लेकिन उसमें परिवर्तन नहीं कर सकते, क्रियान्वयन फाइलें कहलाती हैं। इनका विस्तार नाम ·exe होता है।

फाइल ऑपरेशन्स के चरण (Steps of File Operations)

फाइल से सम्बन्धित कार्यों को दो भागों में विभाजित किया जाता है–

(i) उच्चस्तरीय फाइल ऑपरेशन्स (High-level File Operations)

(ii) निम्नस्तरीय फाइल ऑपरेशन्स (Low-level File Operation)

उच्चस्तरीय फाइल ऑपरेशन्स अपेक्षाकृत सरल तथा आसानी से परिवर्तनीय होने के कारण अधिक उपयोग में लाए जाते हैं, जबकि निम्नस्तरीय फाइल ऑपरेशन विशेष रूप से मशीन-स्तरीय कार्यों के लिए उपयोग में लाए जाते हैं। फाइल ऑपरेशन्स एक निश्चित क्रम में होते हैं; जैसे– सबसे पहले फाइल बनाई जाती है तथा उसे एक नाम दिया जाता है, इसके बाद फाइल खोली जाती है तथा उसमें डाटा पढ़ने या लिखने का कार्य किया जाता है एवं अंतिम चरण में फाइल को बंद किया जाता है। सामान्य फाइल ऑपरेशन्स तथा उनका क्रम निम्नानुसार होता है–

- फाइल का नामकरण (Naming of File)
- फाइल खोलना (Opening a File)
- फाइल से डाटा पढ़ना या लिखना (Reading/Writing)
- फाइल बन्द करना (Closing a File)

❑

प्रश्न 12 – *File handling* क्या है? *File structure* के बारे में बताइए।
What is File handling? Explain file structure. ***(June 2011)***

अथवा (Or)

मुख्य File handling functions की व्याख्या कीजिए।
Describe some of the main file handling functions. ***(June 2011)***

Scanned by CamScanner

अथवा (Or)

File handling technique का उपयोग करके एक फाइल को create करने और save करने का C प्रोग्राम बनाइए।

Write a C Program that is applicable for Creating and Saving a file with file handling technique. ***(Jan. 2014)***

उत्तर– File handling function–एक या एक से अधिक Instructions के समूह को प्रोग्राम कहते हैं। ये Instructions किसी विशेष कार्य को कम्प्यूटर के द्वारा पूरा करने के लिये एक फाइल में लिखी जाती हैं, इस फाइल को प्रोग्राम फाइल कहते हैं। C भाषा में प्रोग्राम फाइल को Save करने के लिये फाइल की Extension ·c लिखते हैं, इससे पता चल जाता है कि यह फाइल C प्रोग्राम की फाइल है। C भाषा की इस प्रोग्राम फाइल में C भाषा के कमाण्ड्स का प्रयोग करते हैं। प्रोग्राम फाइल में Instructions को Save करने के पश्चात् प्रोग्राम को कम्पाइल करते हैं तथा प्रोग्राम की आउटपुट को स्क्रीन पर प्रदर्शित करने के लिये प्रोग्राम को क्रियान्वित करते हैं, तो प्रोग्राम की आउटपुट Save नहीं होती है। जब आप प्रोग्राम की आउटपुट को Save करना चाहते हैं तब इसे किसी फाइल में Save करते हैं, जब इस आउटपुट की पुनः आवश्यकता होती है तब इस फाइल में से आउटपुट को देख लेते हैं या फाइल में से Pick करके स्क्रीन पर प्रदर्शित कर सकते हैं, इस प्रक्रिया को फाइल हैण्डलिंग (File handling) कहते हैं; अर्थात् फाइल में से डाटा को पढ़ने या फाइल में डाटा को लिखने या संशोधित करने की प्रक्रिया को फाइल हैण्डलिंग कहते हैं। फाइल हैण्डलिंग प्रक्रिया में यूजर इनपुट को वेरियेबल में स्टोर करते हैं, इसके पश्चात् इस वैल्यू की आउटपुट को File में लिखते हैं। यह आउटपुट File में Save हो जाती है। इस File में Contents को सीधे देख सकते हैं, यह Contents प्रोग्राम की आउटपुट होती है। जब इस आउटपुट को स्क्रीन पर प्रदर्शित करना चाहते हैं तब File के डाटा को Read वेरियेबल के माध्यम से Read करके स्क्रीन पर प्रदर्शित करते हैं। फाइल हैण्डलिंग में मुख्य रूप से अग्रलिखित ऑपरेशन्स का प्रयोग करते हैं–

(i) Creation of a file of a specific type : इस Operation में हम नयी file को create करते हैं तथा उस file में किस प्रकार के Data को Store करना है यह भी निर्धारित करते हैं।

(ii) Reading/Processing a file : इस Operation में हम पहले से Store की गयी File को Read करते हैं तथा इसमें Store data को अपनी आवश्यकता के अनुसार उपयोग करते हैं।

(iii) Append/Add information to a file : यहां पर हम अपने लिए उपयोगी एवं महत्वपूर्ण data तथा Information को file में store करते हैं।

(iv) Modify/edit data in a file : यहां हम पहले से Save file में से data को लेते हैं तथा उस पर अपनी आवश्यकता के अनुसार बदलाव करके उसे पुनः file में save करते हैं।

(v) Delete items in a file : यदि हमने गलती से कोई data file में store हो गया है जिसकी हमें कोई आवश्यकता नहीं है तो हम operation में हम उस data को मिटाने का काम करते हैं।

(vi) Update the file : यदि हमें file के format से कोई समस्या उत्पन्न हो रही हो तो इस operation में हम उसमें परिवर्तन करके उसे पुनः उपयोगी बनाते हैं।

File handling में प्रयोग किये जाने वाले प्रमुख फंक्शन्स निम्नलिखित हैं–

- **fopen()**–फाइल में डाटा को पढ़ने या लिखने से पहले फाइल को खोलना आवश्यक है। यह फाइल डिस्क में क्रिएट की जाती है। fopen() फंक्शन का प्रारूप निम्नलिखित रूप में होता है–

```
FILE *fp;
fp=fopen("ab.c", "r");
```

यहां पर fp फाइल पॉइन्टर वेरियेबल है, इसमें FILE स्ट्रक्चर का एड्रेस है, जो stdio.h हैडर फाइल में परिभाषित है। इस स्टेटमेण्ट के अनुसार fopen() फंक्शन abc.c फाइल को read mode में open करेगा। fopen() फंक्शन में प्रथम Argument abc.c फाइल का नाम है तथा द्वितीय Argument फाइल का मोड है। यहां पर r का अर्थ read मोड है तथा यह एक स्ट्रिंग है। File mode निम्नलिखित प्रकार के होते हैं, जिन्हें अपनी आवश्यकतानुसार प्रयोग करते हैं–

"r" –यह फाइल को read मोड में खोलता है।

"w"–यह फाइल को write मोड में खोलता है।

"a" –यह पहले से exist फाइल को खोजता है, यदि फाइल exist करती है तो यह फाइल मैमोरी में लोड होती है तथा पॉइन्टर को set किया जाता है जो First Character को Point करता है। यदि फाइल exist नहीं करती है तो एक नई फाइल क्रिएट होती है।

- **fclose()**–फाइल हैण्डलिंग प्रक्रिया में फाइल में Operation को पूरा करने के पश्चात् फाइल को Close करते हैं। इसके लिये fclose() फंक्शन का प्रयोग करते हैं।

Scanned by CamScanner

• **Input/Output फंक्शन**—फाइल में से डाटा को Read करने तथा फाइल में डाटा को Write करने के लिये Input/Output फंक्शन्स का प्रयोग किया जाता है। प्रमुख Input/Output फंक्शन्स निम्नलिखित हैं—

(i) getw() तथा putw()—getw() तथा putw() फंक्शन का प्रयोग फाइल में Integers को हैण्डल करने के लिये किया जाता है। getw() फंक्शन का प्रयोग फाइल में से Integer वैल्यू को Read करने के लिये तथा putw( ) फंक्शन का प्रयोग फाइल में Integer वैल्यू को Write करने के लिये किया जाता है।

(ii) getc() तथा putc()—getc() तथा putc() फंक्शन File में से डाटा को Read करने तथा File में डाटा को Write करने के लिये किया जाता है।

(iii) fprintf तथा fscanf()— **(Jan. 2013, 13, 14; June 2012, 14, 15)**

हम getc() तथा putc() फंक्शन के द्वारा एक समय में एक कैरेक्टर को हैण्डल (Handle) कर सकते हैं इसी तरह getw() तथा putw() फंक्शन के द्वारा इंटीजर वैल्यू को हैण्डल (Handle) कर सकते हैं। fprintf() तथा fscanf() फंक्शन Mixed डाटा को हैण्डल (Handle) कर सकते हैं। fprintf() फंक्शन Mixed डाटा को File में Write करता है। fprintf() फंक्शन का प्रारूप निम्नलिखित रूप में लिखते हैं—

```
fprintf(fp, "control string" , list);
```

यहां पर fp फाइल पॉइन्टर है जो फाइल से एसोशिएट है। control string output specificiations को प्रदर्शित करता है जो Items लिस्ट में होते हैं। List वेरियेबल्स, Constants तथा Strings होती है; जैसे—

```
fprintf(fp, "%s %d", name, age, 2.5);
```

यहां पर name कैरेक्टर ऐरे वेरियेबल है, age इंटीजर तथा 7.5 फ्लोट Constant वैल्यू है।

fscanf() फंक्शन के प्रारूप को निम्नलिखित रूप में लिखते हैं—

```
fscanf(fp, "control string", list);
```

यहां फंक्शन फाइल में से Mixed data को Read करता है। इस फंक्शन में fp फाइल पॉइन्टर है जो फाइल से एसोशिएट है। control string Items लिस्ट के Specifications को प्रदर्शित करता है। list वेरियेबल, constants तथा strings होती है; जैसे—

```
fscanf(fp, "%s%d", & item, & quantity);
```

यहां पर fp फाइल पॉइन्टर है। %s%d control string है। item कैरेक्टर ऐरे तथा quantity इंटीजर वेरियेबल हैं।

(iv) feof() (June 2012)—जब फाइल में से डाटा को Read किया जाता है तब फाइल के end तक डाटा को Read करने के लिये feof() फंक्शन का प्रयोग करते हैं।

निम्नलिखित उदाहरण में File handling के कुछ प्रमुख फंक्शन्स का प्रयोग किया है—

```
#include<stdio.h>
#include<conio.h>
  void main()
  {
  int roll_no, total, i, n;
  char name[10];
  FILE *fp;
fp=fopen("student.dat", "w");
  clrscr( );
printf("no.of students\n");
scanf("%d", &n);
for(i=0; i <n.; i++)
{
printf("student rollno, name, total");
```

Scanned by CamScanner

```
scanf("%d %s%d", & roll_no, name & total);
fprintf(fp, %d%s%d\n", roll_no, name, total);
}
fclose(fp);
fp=fopen("student.dat", "s");
printf("Rollno, Name, total");
while(!feof(fp))
{
fscanf(fp, "%d%s%d", & roll_no, name, & total);
if(total>=70)
printf("%d%s%d", roll_no, name, total);
}
fclose(fp);
getch();
}
```

इस प्रोग्राम में student की सूचना को रन टाईम पर scanf() फंक्शन के द्वारा इनपुट किया गया है। यह सूचना Mixed सूचना (roll no, name, total) के रूप में fprintf() फंक्शन के द्वारा student.dat File में Write की गयी है। इसके पश्चात् File को Close कर दिया है। student.dat फाइल को पुनः Read मोड में Open किया गया है। fscanf() फंक्शन के द्वारा फाइल में से Mixed data को Read करके printf() फंक्शन के द्वारा स्क्रीन पर प्रदर्शित किया गया है। fprintf() तथा fscanf() फंक्शन का प्रयोग सामान्यतः data file में एक्सेस करने के लिये किया जाता है।

❑

प्रश्न 13 – *एक File को writing के लिये कैसे खोलते हैं? उपर्युक्त उदाहरण द्वारा बताइये। एक File pointer को File के अन्त में कैसे निर्धारित करते हैं?*

How will you open a file for writing? Explain with a suitable example. How can you set file pointer at the end of file. **(June 2015; Jan. 2016)**

उत्तर–फाइल में डाटा को लिखने या पढ़ने से पहले फाइल को खोलना आवश्यक है। यह फाइल डिस्क में क्रिएट की जाती है। फाइल को खोलने के लिये fopen() फंक्शन का प्रयोग करते हैं। fopen() फंक्शन का प्रारूप निम्नलिखित रूप में होता है–

```
फाइल पॉइन्टर वेरियेबल = fopen("file name", "file mode")
```

फाइल को writing के लिये w file mode का प्रयोग करते हैं। इसके लिये fopen( ) फंक्शन को निम्नलिखित रूप में डिक्लेयर करते हैं–

```
FILE *fp
fp = fopen ("abc.c", "w");
```

यहां पर fp पॉइन्टर वेरियेबल है, इसमें file स्ट्रक्चर का एड्रेस है, जो stdio.h हैडर फाइल में परिभाषित किया गया है। इस कथन के अनुसार fopen() फंक्शन abc.c फाइल को write mode में open करेगा। निम्नलिखित उदाहरण में इसे स्पष्ट रूप से दर्शाया गया है–

उदाहरण :

```
#include <stdio.h>
#include <conio.h>
void main()
{
FILE *fp;
char c;
```

Scanned by CamScanner

```
printf("Input data as you want");
fp = fopen("abc.txt" "w");
while(cc = getchar( )! = EOF)
put(c, fp);
fclose(fp);
printf("output data as you entered");
fp = fopen("abc.txt", "r");
while((c = getc(fp)! = EOF)
printf("%c", c);
   fclose(f1);
   getch();
   }
```

प्रश्न 14 – *Pointer* की सहायता से दो फाइलों के अवयव आपस में परिवर्तित करने के लिये प्रोग्राम लिखिये।

Write a C program to interchange the contents of two files with the help of pointer.

उत्तर–

```
#include <stdio.h>
#include <conio.h>
#include <stdlib.h>
void main()
{
FILE *fs, *fd;
clrscr();
fs = fopen("a.c", "r");
if(fs = = NULL)
{
printf("can not open file");
exit();
}
fd = fopen("b.c", "w");
if(fd = = NULL)
{
printf("Can not open file");
fclose(fd);
exit();
}
while(1)
{
  ch = getc(fs);
  if(ch = = EOF)
  break;
  else
  putc(ch, fd);
```

Scanned by CamScanner

```
    }
    fclose(fs);
    fclose(fd);
    }
```

प्रश्न 15 – निम्न में अंतर बताइये :

Differentiate between the following :

(i) w and w+

(ii) rb and rb+

(iii) a and a+

(iv) r and r+

उत्तर–(i) w : यह फाइल को write मोड में खोलता है। यदि file exist करती है तो इस फाइल के Contents overwrite होते हैं, यदि फाइल exist नहीं करती है तो एक नई फाइल क्रिएट होती है। यदि फाइल खुलने में unable होती है तो NULL return होता है।

w+ : यह Mode फाइल को search करता है, यदि फाइल exist करती है, इसके Contents हट जाते हैं। यदि फाइल exist नहीं करती है, तो एक नई फाइल क्रिएट होती है, यदि फाइल खुलने में unable होती है तो NULL रिटर्न होती है।

(ii) rb : यह फाइल को बाईनरी कोड में read मोड में खोलता है। यदि फाइल exist करती है तो फाइल मैमोरी में load होती है तथा पॉइन्टर को set किया जाता है, तो List character को Point करता है। यदि फाइल exist नहीं करती है तो NULL रिटर्न होता है।

rb+ : यह फाइल को बाइनरी कोड में search करता है, यदि फाइल exist करती है तो मैमोरी में लोड होती है तथा एक पॉइन्टर को set करता है जो first character को Point करता है। यदि फाइल exist नहीं करती है तो NULL रिटर्न करता है।

(iii) a : यह पहले से exist फाइल को search करता है, यदि file exist करती है तो यह फाइल मैमोरी में Load होती है तथा पॉइन्टर को set किया जाता है जो first character को Point करता है। यदि फाइल exist नहीं करती है तो एक नई फाइल क्रिएट होती है। यदि खुलने में unable होती है तो NULL रिटर्न होती है।

a+ : यह मोड फाइल को search करता है, यदि फाइल exist करती है, तो इसे मैमोरी लोड करता है तथा एक पॉइन्टर को set करता है, जो इसमें first character को Point करता है। यदि File exist नहीं करती है तो एक नई फाइल क्रिएट होती है। यदि फाइल खुलने में unable होती है तो NULL रिटर्न करता है।

(iv) r : यह फाइल को read मोड में खोलता है। यदि फाइल exist करती है तो फाइल मैमोरी में load होती है तथा पॉइन्टर को set किया जाता है जो list character को Point करता है।

r+ : यह मोड फाइल को search करता है, यदि फाइल exist करती है तो मैमोरी में लोड होती है तथा एक पॉइन्टर को set करता है जो first character को Point करता है। □

प्रश्न 16 – *Text file तथा Binary file के अन्तर को स्पष्ट कीजिए।*

Explain the difference between Text file and Binary file. ***(Jan. 2012; June 2015)***

उत्तर–Text File–Text File वह File होती है जिसमें Information कैरेक्टर्स या स्ट्रिंग के रूप में Store होती हैं। Keyboard के माध्यम से Input किये गये कैरेक्टर्स फाइल में Continously Store होते हैं। निम्नलिखित चित्रण में Information को Text File में कैरेक्टर्स के रूप में व्यवस्थित किया गया है–

A text file →	Computer is an electronic device which can be used for data processing, numerical problem solving and so on.
	. . . ← EOF (End of File)

इस चित्रण में Text File को दर्शाया गया है। इसमें Information को String के रूप में Store किया है, इसे Text File कहते हैं।

Scanned by CamScanner

Binary file–Binary files रिकॉर्ड्स की ऐरे की तरह ही होती हैं। रिकॉर्ड्स डिस्क फाईल में स्टोर होते हैं जबकि ऐरे मैमोरी में स्टोर होती हैं। बाईनरी फाईल में रिकॉर्ड्स डिस्क पर उपस्थित होते हैं। ये स्थायी रूप में होते हैं तथा हमेशा उपलब्ध होते हैं। Text files की तुलना में बाईनरी फाईल्स के दो प्रमुख फीचर्स होते हैं। प्रथम फीचर के अनुरूप हम किसी भी समय फाईल में कहीं भी किसी भी रिकॉर्ड को देख सकते हैं जो एक ऐरे में रेन्डम एक्सेस को उपलब्ध कराते हैं तथा किसी भी समय कहीं भी किसी रिकॉर्ड को बदल सकते हैं। द्वितीय फीचर के अनुरूप बाईनरी फाईल्स, Text file की तुलना में Read तथा Writes times को शीघ्रता से एक्सेस करते हैं।

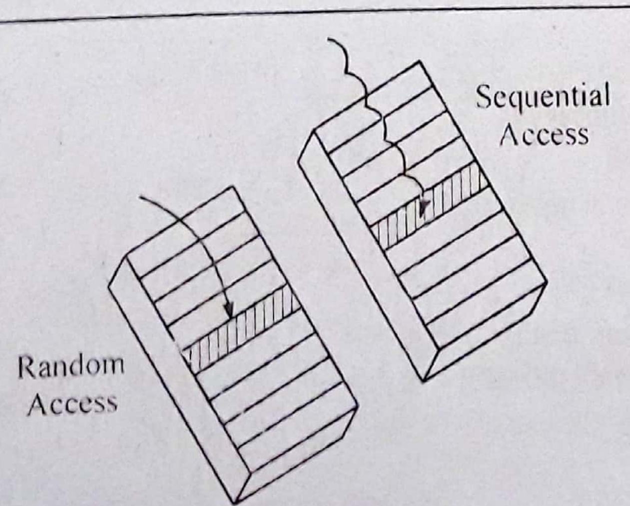
निम्नलिखित उदाहरण में इस विधि को स्पष्ट रूप से दर्शाया गया है–

उदाहरण–

```
#include<stdio.h>
#include<conio.h>
    struct abc
    {
    int a,b,c;
    };
void main()
    {
        int i,j;
    FILE *f;
    struct abc a1;
    f=fopen("ab","wb");            //write in binary mode
    for(i=1;i<=10; i++)
    {
        a1.i=a;
    fwrite(& a1, sizeof(struct a1), 1,f);
    }
    fclose(f);
    f=fopen("abc", "rb");                    //read in binary mode
    for(i=1; i<=10; i++)
    {
        fread(& a1, sizeof(struct a1), 1, f);
        printf("%d", a1.i);
    }
    fclose(f);
```

❑

प्रश्न 17 – ***Direct access* व *Random access* क्या है?**
What is Direct and Random access? ***(June 2011, 15)***



उत्तर–रेंडम एक्सेस को सीधा एक्सेस (direct access) भी कहा जाता है। इसमें डाटा को सीधे पढ़ा जा सकता है। इसी कारण से रेंडम फाइले कृमिक फाइलों की तुलना में तेज गति से एक्सेस होती हैं। रेंडम एक्सेस फाइल में रिकॉर्ड को रेंडम क्रम में रखा जाता है। रेंडम एक्सेस फाइल में प्रत्येक रिकॉर्ड के साथ relative index number जुड़ा होता है। जब भी एक रेंडम एक्सेस फाइल से एक रिकॉर्ड को पढ़ा जाना है, तो एक कम्प्यूटर प्रोग्राम इस रिकॉर्ड के लिए एक relative index number का उत्पादन करता है। इस number की सहायता से रिकॉर्ड का पता लगाया जाता है। रेंडम एक्सेस फाइल के निम्नलिखित लाभ हैं–

(1) इसमें इच्छित जानकारी को तेज गति से एक्सेस किया जा सकता है। एक decision making पर्यावरण में, जहाँ जानकारी की आवश्यकता तुरंत होती है, वहां डाटा की पुर्नप्राप्ति के लिए रेंडम एक्सेस एक आवश्यकता है।

(2) यह एक समय में अपेक्षाकृत कुछ रिकॉर्ड को पुनः प्राप्त करने के लिए कुशल तकनीक है।

(3) यह एक विधि है जिससे फाइलों को up-to-date रखा जा सकता है। जैसे–-जैसे– transactions और घटनाएँ होती हैं।

(4) एक रेंडम एक्सेस फाइल में हम फाइल में कही से भी डाटा को रीड कर सकते है और कहीं पर भी जानकारी लिख सकते है। एक क्रमिक-एक्सेस फाइल में, हम डाटा केवल क्रमिक रूप में पढ़ और लिख सकते हैं।

डिस्क रेंडम एक्सेस मीडिया है जबकि magnetic टेप क्रमिक एक्सेस मीडिया है।

रेंडम फाइलों का निर्माण करना और उपयोग करना थोड़ा कठिन अवश्य है परन्तु रेंडम फाइल तेज गति से एक्सेस होने के कारण बहुत समय बचाती है। इन फाइलों को सेकेंड्री स्टोरेज डिवाइस जैसे–डिस्क पर ही स्टोर किया जाता है। ❑

प्रश्न 18 – फंक्शन्स को उदाहरण सहित समझाइए :

Explain putw() functions with example. ***(June 2012)***

उत्तर–putw( )–putw( ) फंक्शन का प्रयोग File में इंटीजर्स को हैण्डल करने के लिये किया जाता है। putw( ) फंक्शन का प्रयोग इंटीजर वैल्यू को File में Write करने के लिये किया जाता है। putw( ) फंक्शनूस का सामान्य प्रारूप निम्नलिखित रूप में किया जाता है–

```
putw(integer, fp);
```

putw( ) फंक्शन में दो आर्ग्यूमेन्ट Pass किये जाते हैं। प्रथम आर्ग्यूमेण्ट इंटीजर वैल्यू होती है तथा दूसरा आर्ग्यूमेन्ट फाइल पॉइन्टर वेरियेबल होता है। निम्नलिखित उदाहरण में putw( ) फंक्शन का प्रयोग किया गया है–

उदाहरण :

```
#include<stdio.h>
#include<conio.h>
  void main( )
  {
  FILE *f1, *f2, *f3;
  int a, i;
  printf("contents of first file\n");
  f1=fopen("first.txt", "w");
  for(i=1; i<=20; i++)
  {
  scanf("%d", &a);
  if(number== -1)b
  break;
  putw(number, f1);
  }
fclose (f1);
f1=fopen("first.txt", "r");
f2=fopen("odd.txt", "w");
f3=fopen("even.txt", "w");
```

Scanned by CamScanner

```
while((a=getw(f1)!=EOF)
    {
    if a %2== 0)
    putw(f3);
    else
    putw(a, f2);
    }
fclose(f1;
fclose(f2);
fclose(f3);
f2=fopen("odd.txt", "r");
f3=fopen("even.txt", "r");
printf("\n contents of odd file\n");
while(a=getw (f2))!=EOF)
printf("%d", a);
printf("\n contents of even file\n");
while( a=getw (f2))!=EOF)
printf("%d", a);
fclose(f2);
fclose(f3);
getch( );
}
```

इस उदाहरण में abc.txt file में Numbers को इनपुट किया गया है। इसके पश्चात् abc.txt file को Read mode में Open किया है, इस फाइल में से even Numbers को even.txt फाइल में putw( ) फंक्शन के द्वारा write किया गया है तथा odd Numbers को odd.txt फाइल में Write किया गया है। इसके पश्चात् सभी Files को Close कर दिया है। पुनः even.txt तथा odd.txt फाइल को Read mode में Open किया है तथा abc.txt फाइल में से even Numbers तथा odd Numbers को स्क्रीन पर प्रदर्शित किया गया है। even.txt, odd.txt तथा abc.txt फाइल में डाटा हमेशा के लिये सुरक्षित रहता है। हम डाटा को कभी भी देख सकते हैं। यही फाइल हैण्डलिंग का लाभ है। ❑

प्रश्न 20 – *rewind( ) फंक्शन को उदाहरण देकर समझाइए।*

Explain rewind( ) function with example. ***(Jan. 2013)***

उत्तर– rewind( ) : rewind( ) फंक्शन में फाइल पॉइन्टर Argument के रूप में Pass किया जाता है, यह File की Start Position को reset करता है। इसका प्रारूप निम्नलिखित रूप में होता है–

```
rewind(fp);
```

उदाहरण–

```
#include <stdio.h>
int main()
{
  FILE *fp;
  int ch;
  fp = fopen("file.txt", "r");
```


Scanned by CamScanner

```
    if( fp != NULL )
    {
      while( !feof(fp) )
      {
        ch = fgetc(fp);
        printf("%c", ch);
      }
      rewind(fp);

      while( !feof(fp) )
      {
        ch = fgetc(fp);
        printf("%c", ch);
      }
      fclose(fp);
    }
    return(0);
}
```

प्रश्न 21 – निम्नलिखित फंक्शन्स का वर्णन कीजिए :

Explain the following functions :

(i) fread( ) ***(June 2012, 2013, 15; Jan. 2014)***

(ii) fwrite( ) ***(June 2013, 15; Jan. 2014)***

उत्तर–(i) fread( )–fread( ) के द्वारा किसी भी file के डाटा को structure में पढ़ा जा सकता है। अतः किसी भी program में fread( ) को प्रयुक्त करने के लिए structure को define करना आवश्यक होता है। fread( ) चार arguments को अपने प्रयोग में लाता है। पहला argument structure variable का address होता है, दूसरा argument उस variable का size होता है, तीसरा arguments no. of records को show करता है तथा चौथा **argument** file का pointer होता है; जैसे–

```
fread(&e, sizeof(e), 1, fp);
```

यहां e एक structure variable है जिसमें डाटा फाइल से पढ़कर store किया जाता है एवं fp वो file pointer है, जिससे related file में डाटा पढ़ा जाता है तथा तीसरे argument की संख्या दर्शाती है कि एक बार में केवल एक ही record पढ़ा जाना है।

(ii) fwrite()–fwrite() के द्वारा किसी भी file में डाटा को structure के द्वारा लिखा जाता है। अतः किसी भी program में fwrite() को प्रयुक्त करने के लिए structure को define करना आवश्यक होता है। fwrite() चार arguments को अपने प्रयोग में लाता है। पहला argument, structure variable का address होता है, दूसरा argument उस variable का size होता है, तीसरा argument no. of records को show करता है तथा चौथा argument file का pointer होता है। जैसे–

```
fwrite(&e, sizeof(e), 1, fp);
```

यहां e एक structure variable है जो hold data को file में लिखने में प्रयुक्त होता है। fp एक file pointer है जिससे related file में डाटा लिखा जाता है तथा तीसरे argument की 1 संख्या दर्शाती है कि एक बार में केवल एक ही record लिखा जाना है।

Scanned by CamScanner

प्रश्न 22 – फाइल ऑर्गेनाइजेशन से आप क्या समझते हैं? Employee की file create करने के लिए C प्रोग्राम लिखिए। फाइल निर्माण के लिए अपनी आवश्यकतानुसार fields का चयन कीजिए।

What is meant by file organization? Write a C program to create employee file. Take required fields in the file as per your choice. ***(June 2014)***

उत्तर–File Organisation—डाटा को एक सही रूप में ऑर्गेनाइज करने की प्रक्रिया को File Organization कहते हैं। File Organization तीन प्रकार की होती हैं–

1. Sequential File Organization
2. Indexed Sequential File Organization
3. Direct File Organization

1. Sequential File Organization—Sequential File Organization में Records उसी क्रम में स्टोर होते हैं, जिस क्रम में उन्हें File में लिखा जाता है।

2. Indexed Sequential File Organization—Indexed Sequential File Organization में, Records प्राइमरी Key के अनुरूप क्रम में स्टोर होते हैं तथा फाइल में Random Access की अनुमति को देने के लिये Index क्रिएट किया जाता है। इस टाईप का ऑर्गेनाइजेशन, फाइल को Sequentially एक्सेस करने की भी अनुमति देता है। Indexed Sequential अत्यधिक प्रयोग किया जाने वाला File Organization है।

3. Direct File Organization(June 2015)—Direct File Organization में, Records को एक रिलेटिव नम्बर के माध्यम से स्टोर तथा वापिस किया जाता है, जो फाईल में रिकॉर्ड की स्थिति को देता है। इस टाईप का ऑर्गेनाइजेशन Sequentially एक्सेस के लिये भी Allow करता है।

एक फाइल employee.txt में employee की id name तथा salary को user द्वारा input कराके store किया गया है। जिस employee की salary 25,000 से अधिक है, उन employee की list को निम्नलिखित प्रोग्राम द्वारा प्रदर्शित किया गया है–

```
#include<stdio.h>
#include<conio.h>
void main()
{
int id, tot,i,n;
char name[10];
file *f1;
f1=fopen("employee.txt","w");
clrscr();
printf("how many employees");
scanf("%d",&n);
for( i=0;i<=n;i++)
{
printf("employee id, name, total");
scanf("%d%s%d",&id,name,&tot);
fprintf(f1,"%d%s%d\n",id,name,tot);
}
fclose(f1);
f1=fopen("employee.txt","r");
clrscr();
printf("employee id, name, total");
while(! feof(f1))
{
fscanf(f1 ,"%d%s%d",&id,name,&tot);
if(tot>25,000)
```

Scanned by CamScanner

```
printf("%d \t %s %d ",id,name,tot);
}
fclose(f1);
getch();
}
```

प्रश्न 23 – *# else को उदाहरण सहित समझाइये।*

Explain # else with example.

(June 2015)

उत्तर–#else directives–#else का प्रयोग #if preprocessor directive के साथ किया जाता है। इसका प्रारूप निम्नलिखित रूप में होता है–

```
#if constant – expression
    statements
#else
    statements
#endif
```

यदि Constant Expression Nonzero को Evaluate करता है तो #if तथा #else के statements Compiled होते हैं अन्यथा #else तथा #endif के statements Compiled होते हैं।

निम्नलिखित उदाहरण में #else directive का प्रयोग किया गया है–

```
#include<stdio.h>
#include<conio.h>
#define FLAG 3
    void main( )
    {
        int a=10, b=3
#if FLAG>=3
    printf("value of FLAG is greaterthan equal to 2\n");
    a=a+b;
    b=a*b;
#else
    printf("value of FLAG is less than 2\n");
    a=a – b;
    b=a/b;
#endif
printf("a=%d, b=%d", a, b);
getch( );
}
```

Scanned by CamScanner

UNIT-III

प्रश्न 1 – Function और Macro के बीच अंतर स्पष्ट करें। Marco को Argument के साथ कैसे परिभाषित किया जाता है, उदाहरण के साथ उल्लेख करें।

Differentiate between Macro & function ? How to define macro with an argument ? Explain with example. *(June 2003, 05, 06, 11; Jan. 2011)*

उत्तर–प्रोग्राम में उपयोग होने वाले जटिल समीकरणों को दिया जाने वाला सांकेतिक नाम मैक्रो कहलाता है, जिसके द्वारा हम उस समीकरण को प्रोग्राम में कहीं भी उपयोग कर सकते हैं। अगर हमारे प्रोग्राम में कोई जटिल समीकरण है जिसका हमें प्रोग्राम में कई बार उपयोग करना है, तो उसे प्रोग्राम में बार-बार टाइप करने में समय अधिक लगेगा तथा अनावश्यक रूप से प्रोग्राम का कोड भी अधिक बड़ा होगा। इसे सरल बनाने के लिए उपयोग की जाने वाली तकनीक मैक्रो कहलाती है। इस तकनीक में जटिल समीकरण को एक सांकेतिक नाम देकर प्रोग्राम में सबसे ऊपर #define प्री-प्रोसेसर डायरेक्टिव के साथ लिख दिया जाता है तथा प्रोग्राम में उस समीकरण का जहाँ-जहाँ भी उपयोग करना हो वहाँ उसे दिये गए सांकेतिक नाम के द्वारा कॉल कर लिया जाता है। समीकरण को दिया जाने वाला यह सांकेतिक नाम मैक्रो कहलाता है।

इसका सिंटेक्स निम्नानुसार होता है–

```
#define MACRO_NAME equation
```

सामान्यतः मैक्रो का नाम अंग्रेजी के बड़े अक्षरों में ही दिया जाता है। यद्यपि यह कोई नियम नहीं है लेकिन प्रोग्राम में मैक्रो का नाम अलग से दिखाई दे जिससे उसे पहचानना आसान हो इसलिए बड़े अक्षरों का ही उपयोग किया जाता है।

(June 2013)

मैक्रो के लाभ (Advantages of Macro)

मैक्रो के प्रमुख लाभ निम्न हैं–

1. समय की बचत होती है।
2. जटिल समीकरण जिसका प्रोग्राम में अनेक स्थानों पर उपयोग हुआ हो बार-बार टाइप नहीं करना पड़ता।
3. प्रोग्राम कोड छोटा तथा समझने में आसान हो जाता है।
4. प्रोग्राम की गति बढ़ जाती है।
5. प्रोग्राम को सरलता से परिवर्तनीय बनाता है; अर्थात् मैक्रो की परिभाषा में परिवर्तन करने पर वह प्रोग्राम में जहां-जहां भी प्रयुक्त हो रहा है सभी जगह अपडेट हो जाता है।

इसके लिए विपरीत फंक्शन एक ऐसी सुविधा है जिसके द्वारा हम मुख्य प्रोग्राम को छोटे-छोटे ब्लॉक में विभाजित कर सकते हैं। मुख्य प्रोग्राम से अलग बनाया गया एक छोटा ब्लॉक जिसे मुख्य प्रोग्राम में कॉल किया जा सकता है फंक्शन कहलाता है। इसमें हम मुख्य प्रोग्राम से आरग्यूमेंट पास कर सकते हैं तथा फंक्शन भी मुख्य प्रोग्राम को मान लौटा सकता है। एक ही फंक्शन को मुख्य प्रोग्राम में अनेक स्थानों पर कॉल किया जा सकता है।

मैक्रो तथा फंक्शन में अन्तर (Difference in Macro and Function)

मैक्रो तथा फंक्शन के बीच मुख्य अंतर निम्नलिखित हैं–

मैक्रो	फंक्शन
1. मैक्रो कॉल में मैक्रो टेम्प्लेट मैक्रो के विस्तार से प्रतिस्थापित होता है।	1. फंक्शन कॉल में प्रोग्राम का नियन्त्रण आरग्यूमेंट के साथ प्रोग्राम के कोड पर स्थानान्तरित होता है।
2. मैक्रो प्रोग्राम की गति बढ़ाते हैं लेकिन कम्पाइल किये गए कोड में मैक्रो के स्थान पर उसकी वास्तविक परिभाषा कॉपी होने के कारण कम्पाइल्ड कोड का आकार भी बढ़ाते हैं।	2. यह प्रोग्राम कोड को छोटा एवं संक्षिप्त बनाता है।
3. सामान्यतः मैक्रो छोटे समीकरणों के लिए बनाया जाता है।	3. प्रोग्राम के बड़े मॉड्यूल्स को अलग-अलग फंक्शनों में बांटा जाता है।

Scanned by CamScanner

4. कम्पाइल्ड कोड में मैक्रो टैम्प्लेट को मैक्रो विस्तार से प्रतिस्थापित किये जाने से प्रोग्राम की गति बढ़ जाती है।	4. फंक्शन में प्रोग्राम नियंत्रण स्थानान्तरित होने पर आरग्यूमेंट देने और रिटर्न मान प्राप्त करने में समय अधिक लगता है जिससे प्रोग्राम की गति कम हो जाती है।
5. कम्पाइल करते समय आरग्यूमेंट के डाटा टाइप की जांच नहीं की जाती।	5. कम्पाइल करते समय आरग्यूमेंट के डाटा टाइप की जांच की जाती है तथा आरग्यूमेंट और पैरामीटर में अंतर होने पर एरर बतलाता है।
6. सी०पी०यू० पर अतिरिक्त बोझ नहीं पड़ता	6. फंक्शन कॉल के स्थान से प्रोग्राम नियन्त्रण स्थानान्तरित होने के कारण सी०पी०यू० पर अतिरिक्त बोझ पड़ता है।

आरेग्यूमेंट के साथ मैक्रो (Marco with Argument) **(June 2013)**

मैक्रो में भी वेरिएबल तथा कॉन्सटेंट उपयोग किये जाते हैं, अतः फंक्शन के समान ही मैक्रो में भी मुख्य प्रोग्राम के आरग्यूमेंट पास किये जा सकते हैं तथा मैक्रो द्वारा return किये गए मान को मुख्य प्रोग्राम में एक वेरिएबल में लिया जा सकता है।

उदाहरण–

```
# define CUBE (x) (x*x*x)
```

यहाँ पहले छोटे कोष्ठक में दिया गया x एक आरग्यूमेंट है जिसमें मुख्य प्रोग्राम से मान पास होगा। इसे हम निम्न प्रोग्राम के द्वारा समझ सकते हैं–

```
#include<stdio.h>
#define CUBE(a) (a* a*a)
viod main()
{
    int n;
    int c;
    printf("Enter a number");
    scanf("%d", & n);
    c=CUBE(a);
    printf("Cube of given number=%d", c);
}
```

प्रोग्राम में a एक आरग्यूमेंट है तथा मैक्रो द्वारा return किया गया मान c वेरिएबल में आ रहा है। □

प्रश्न 2 – प्री-प्रोसेसर डायरेक्टिव्स से आप क्या समझते हैं? इसकी कार्यप्रणाली की व्याख्या करें। दो प्री-प्रोसेसर डायरेक्टिव की उदाहरण के साथ व्याख्या करें।

What do you understand by pre-processor directives. Explain their working concepts? Give two examples of pre-processor directive. ***(June 2003, 04, 06; Jan. 2013)***

अथवा (Or)

Preprocessor directives को example द्वारा समझाइए।

Explain preprocessor directive with example. ***(Jan. 2014, 15)***

उत्तर–प्री-प्रोसेसर ऐसे कमाण्ड होते हैं जो प्रत्येक प्रोग्राम के हैश (#) के साथ सबसे ऊपर लिखे जाते हैं, जैसे #include यह एक फाइल को Include करने का डायरेक्टिव है जो हेडर फाइलों को तथा यूजर की किसी अन्य प्रोग्राम फाइल को खुले हुए प्रोग्राम में Include करता है। प्री-प्रोसेसर 'C' के ऐसे प्रोग्राम होते हैं जो भेजे गए डायरेक्टिव के अनुसार C के सोर्स प्रोग्राम को संशोधित करता है। प्री-प्रोसेसर एक ऐसा प्रोग्राम होता है जो यूजर द्वारा लिखे गए सोर्स प्रोग्राम के कंपाइलेशन के पहले क्रियान्वित होता है।

प्री-प्रोसेसर डायरेक्टिव निम्न प्रकार के होते हैं–

(i) मैक्रो एक्सपेंशन (Macro Expansion) (ii) फाइल इन्क्लूजन (File inclusion)

(iii) कंडीशनल कम्पाइलेशन (Conditional Compilation)

Scanned by CamScanner

(i) मैक्रो एक्सपैंशन (Macro Expansion)

प्रोग्राम में void main() के ऊपर लिखे गए कोड तथा समीकरण जिन्हें एक सांकेतिक नाम के द्वारा पहचाना जाता है, मैक्रो एक्सपैंशन कहलाते हैं तथा इनकी पहचान के लिए उपयोग किया जाने वाला सांकेतिक नाम मैक्रो कहलाता है। ये मैक्रो #define प्री-प्रोसेसर डायरेक्टिव के साथ लिखा जाता है। मैक्रो C कोड को अधिक सरल तथा संक्षिप्त बनाते हैं। इसे हम निम्न प्रोग्राम के द्वारा समझ सकते हैं–

```
#include<stdio.h>
#define AND &&
void main()
{
    int per;
    printf("Enter your percentage");
    scanf("%d,", & per);
if(per >=45 AND per <60)
    printf("you are second division");
}
```

सामान्य रूप से हम AND कंडीशन के लिए && ऑपरेटर का उपयोग करते हैं लेकिन यहाँ मैक्रो के माध्यम से AND शब्द का उपयोग किया गया है जिससे कंडीशन समझने की दृष्टि से अधिक सरल हो गई है।

(ii) फाइल इन्क्लूजन (File Inclusion)

C प्रोग्राम में फाइल इन्क्लूजन डायरेक्टिव हैडर फाइल को प्रोग्राम में जोड़ने तथा यूजर के ही किसी अन्य प्रोग्राम को नये प्रोग्राम में जोड़ने के लिए प्रयुक्त होता है। इसके लिए #include की-वर्ड (Key-word) का उपयोग किया जाता है। हैडर फाइल को प्रोग्राम में जोड़ने के लिए #include के साथ < > चिह्नों का उपयोग किया जाता है जिनके बीच हैडर फाइल का नाम लिखा जाता है तथा यूजर की किसी C फाइल को नये प्रोग्राम में जोड़ने के लिए फाइल का नाम " " के बीच लिखा जाता है।

उदाहरण–

```
#include<stdio.h>
#include "Area·c"
```

एंगल ब्रकेट्स < > प्री-प्रोसेसर को एक या अधिक स्टैंडर्ड डायरेक्ट्रीज में फाइल सर्च करने को कहता है। इन डायरेक्ट्रीज में हैडर फाइल होती है, जो सिस्टम द्वारा प्रदान की जाती है तथा हमारे प्रत्येक C प्रोग्राम में प्रयुक्त होती है। जब हम < > के स्थान पर " " (इन्वर्टेड कॉमाज) का उपयोग करते हैं तब प्री-प्रोसेसर पहले स्वयं प्रोग्रामर की डायरेक्ट्री में तथा उसी डायरेक्ट्री में इन्क्लूड की जा रही फाइल को ढूंढता है जिसमें नया प्रोग्राम है। एक हैडर फाइल के अन्दर भी दूसरे #include डायरेक्टिव सम्मिलित हो सकते हैं।

(iii) कंडीशनल कंपाइलेशन (Conditional Compilation)

किसी कंडीशन के आधार पर कोड के किसी भाग को चुनने के लिए उपयोग किये जाने वाले प्री-प्रोसेसर डायरेक्टिव कंडीशन कंपाइलेशन कहलाते हैं। कंडीशन कंपाइलेशन के लिए # ifdef तथा #endif प्री-प्रोसेसर डायरेक्टिव उपयोग किये जाते हैं, जिनका सामान्य प्रारूप निम्नानुसार होता है–

```
#ifdef macroname
    <statement1>;
#else
    <statement2>;
#endif
```

❑

प्रश्न 3 – *कंडीशनल कम्पाइलर डायरेक्टिज को समझाइए*

Explain Conditional Compiler Directives ? ***(June 2005, 11, 13; Jan. 2012)***

अथवा (Or)

कंडीशनल डायरेक्टिज को उदाहरण सहित समझाइए।

Explain Conditional directive with example. ***(Jan. 2015)***

उत्तर–कंडीशनल कम्पाइलेशन डायरेक्टिव अपने आप में बहुत महत्वपूर्ण तथा उपयोगी हैं क्योंकि इनके द्वारा हम कंडीशन आधारित

Scanned by CamScanner

प्री-प्रोसेसर कोड तैयार कर सकते हैं। यह सुविधा प्री-प्रोसेसर कोड को अत्यंत शक्तिशाली बना देती है, क्योंकि किसी भी प्रोग्राम कोड में कंडीशन के अनुसार चुनाव की सुविधा काफी महत्व रखती है। इसके लिए #ifdef तथा #endif प्री-प्रोसेसर डायरेक्टिव उपयोग किये जाते हैं, जिनकी सहायता से हम किसी कंडीशन के आधार पर सोर्स कोड के किसी भाग को चुन सकते हैं या कंडीशन पूरी ना होने की स्थिति में कम्पाइलर कोड के उस भाग को लोड कर आगे बढ़ सकता है। इसके लिए कंडीशनल कम्पाइलेशन डायरेक्टिव का प्रारूप निम्नानुसार होगा–

```
#ifdef macroname
    statement1;
    statement2;
    statement3;
    #endif
```

उपरोक्त प्रारूप में हमारे पास केवल एक ही कंडीशन आधारित कोड समूह है जो कंडीशन पूरी होने पर चुना जाएगा या कंडीशन पूरी ना होने पर छोड़ दिया जायेगा, अर्थात् इस प्रारूप के द्वारा हम कंडीशन के आधार पर किसी कोड समूह को चुनने या छोड़ने का काम कर सकते हैं।

उपरोक्त कंडीशन डायरेक्टिव्ज के साथ else का उपयोग करके हम किन्हीं दो कोड समूहों में से किसी एक का चुनाव कर सकते हैं। इस सुविधा के लिए उपयोग किया जाने वाला प्रारूप निम्नानुसार होता है–

```
#if condition
    statement1;
    statement2;
    statement3;
#else
    statement4;
    statement5;
    statement6;
#end if
```

इस प्रारूप में कंडीशन लागू होने पर statement1, 2 और 3 चुने जाएंगे तथा कंडीशन लागू ना होने की स्थिति में statement4, 5 तथा 6 चुने जाएंगे। इस प्रकार हम कंडीशन के आधार पर किन्हीं दो statement समूहों में से किसी एक को चुन सकते हैं।

कंडीशन कंपाइलेशन डायरेक्टिव्ज का उपयोग निम्नलिखित विशेष स्थितियों में भी किया जाता है–

1. जब एक ही फंक्शन दो अलग-अलग हैडर फाइलों में परिभाषित किया गया हो तथा दोनों हैडर फाइलें एक ही प्रोग्राम में शामिल की गई हों, तब उनके बीच उत्पन्न विरोध को हटाने के लिए।
2. प्रोग्राम को पोर्टेबल बनाने के लिए; अर्थात् इनके द्वारा प्रोग्राम को दो बिल्कुल ही अलग-अलग तरह से कम्प्यूटरों पर चलाने लायक बनाया जा सकता है।
3. कोड की प्रचलित लाइनों को Comment out करने के लिए। ऐसा प्रायः किसी कोड को प्रोग्राम से अस्थायी रूप से हटाकर उसका प्रभाव देखने के लिए किया जाता है या फिर यूजर की संतुष्टि के लिए अंतिम समय में प्रोग्राम के कोड़ में परिवर्तन करने के लिए भी इसका उपयोग किया जाता है। कंडीशन के आधार पर Comment out से तात्पर्य किसी कोड समूह को क्रियान्वित होने से रोकने के लिए उसके साथ कोई ऐसी कंडीशन डालने से है जो कभी पूरी ही ना हो। □

प्रश्न 4 – नीचे दिये गये डायरेक्टिव्ज के उपयोगों का उल्लेख करें–

Give the usage of directives given below : ***(June 2005)***

(i) #if ***(Jan. 2016; June 2014, 15)*** (ii) #undef ***(Jan. 2016)***

(iii) #pragma ***(Jan. 2013, 14; June 2014, 15)***

उत्तर– (i) #if

if एक कंडीशनल डायरेक्टिव है जिसके द्वारा हम कंडीशन का मिलान कर सकते हैं तथा कंडीशन के आधार पर किसी कोड समूह का चुनाव कर सकते हैं। इसे कंडीशन कम्पाइलेशन डायरेक्टिव कहा जाता है जिसका अर्थ यह है कि कंडीशन के आधार पर चुने जाने

Scanned by CamScanner

वाले कोड समूह इस प्री-प्रोसेसर से घिरे रहते हैं। इसका अर्थ है कि अगर #if डायरेक्टिव की सहायता से निर्धारित की गई कंडीशन सत्य होती है तो कोड का वह भाग जो #if डायरेटिव के अंतर्गत लिखा गया है, कम्पाइल करने के लिए कम्पाइलर को भेजा जाएगा। अगर कंडीशन असत्य है तो कोड का यह भाग कम्पाइल ही नहीं होगा। इसका सबसे बड़ा लाभ यह है कि प्रोग्राम को कम्पाइल करने के बाद बनने वाले ऑब्जेक्ट कोड में केवल वही कोड रहते हैं जिनका क्रियान्वयन होना ही है, कंडीशन पूरी ना होने के कारण जिन कोड समूह का क्रियान्वयन होना ही नहीं हैं वे ऑबजेक्ट कोड में रखे ही नहीं जाते जिससे ऑब्जेक्ट कोड छोटा हो जाता है। इसका प्रारूप निम्नानुसार होता है–

```
#if conditon
    statement1;
    statement2;
    statement3;
    .................
    .................
#endif
```

#if के साथ हम #else का उपयोग कर किन्हीं दो कोड समूहों में से किसी एक का चुनाव भी कर सकते हैं। इस स्थिति में इसका प्रारूप निम्नानुसार होगा–

```
#if condition
    statement1;
    statement2;
    ...............
    ...............
#else
    statement n;
    ...............
    ...............
#endif
```

(ii) #undef *(Jan. 2016)*

#undef डायरेक्टिव का उपयोग #define द्वारा परिभाषित किए गए मैक्रो को समाप्त करने के लिए किया जाता है। मान लीजिए हमने कोई मैक्रो डिफाइन किया है जिसे हम समाप्त करना चाहते हैं, इसके लिए हम #undef डायरेक्टिव का उपयोग करेंगे। इसका प्रारूप निम्नानुसार होता है–

```
#undef MACRO_NAME
```

उदाहरण–

```
#define AGE "Year old"
printf("I am 28%s", AGE);
#udef AGE
```

उपरोक्त कोड में एक AGE नाम का मैक्रो परिभाषित किया गया है जिसे printf() फंक्शन में उपयोग में लिया गया है तथा इसके बाद मैक्रो को समाप्त कर दिया गया है।

Scanned by CamScanner

(iii) #pragma

pragma एक विशेष डायरेक्टिव है क्योंकि इसकी कोई निश्चित परिभाषा नहीं होती। यह एक कम्पाइलर आधारित डायरेक्टिव है अर्थात् अलग-अलग कम्पाइलर में इसके कार्य अलग-अलग होते हैं। मुख्यतः इसका उपयोग सिस्टम के कुछ फीचर्स को ऑन या ऑफ करने के लिए किया जाता है; जैसे–

```
#pragma startup
#pragma exit
```

आदि।

यहां startup किसी फंक्शन को मुख्य प्रोग्राम के पहले RUN करवाने के लिए तथा exit किसी फंक्शन को मुख्य फंक्शन के बाद RUN करवाने के लिए उपयोग किया जाता है। इसे हम निम्न प्रोग्राम के द्वारा समझ सकते हैं–

```
void sub1();
void sub2();
#pragma startup sub1();
#pragma exit sub2();
main()
    {
    printf("\n this is main program");
}
void sub1()
    {
    printf("\n this is sub1");
    }
void sub2()
    {
    printf("\n This is sub2");
    }
```

Output :

```
This is sub1
This is main program
This is sub2
```

प्रश्न 5 – प्री-प्रोसेसर की आवश्यकता क्या है? अपने उत्तर को उचित उदाहरण से समझाइये।

What is the need of pre-processor ? Explain your answer with proper example.

(June 2006; Jan. 2015)

उत्तर–C में प्री-प्रोसेसर बहुत आवश्यक है। हम अपने C प्रोग्राम में अनेक पहले से बने हुए फंक्शन तथा कोड उपयोग में लाते हैं जो किसी अन्य प्रोग्राम फाइल या हैडर फाइल में लिखे गए होते हैं। प्री-प्रोसेसर डायरेक्टिव कम्पाइलेशन के पहले उन्हें प्रोग्राम में जोड़ता है ताकि कम्पाइलेशन के समय कम्पाइलर को वे फंक्शन मिल सकें।

Scanned by CamScanner

उदाहरण–यहाँ हम एक बहुत ही सरल उदाहरण के माध्यम से प्री-प्रोसेसर के महत्व को समझ सकते हैं–

```
#include<stdio.h>
#include<conio.h>
void main()
{
    clrscr();
    printf("This is a trail program");
}
```

उपरोक्त उदाहरण में हमने clrscr() फंक्शन का उपयोग किया है जो conio.h हैडर फाइल में होता है। अतः #include<conio.h> मुख्य प्रोग्राम को कम्पाइल करने से पहले ही इस हैडर फाइल को प्रोग्राम में जोड़ देगा। अब जब प्रोग्राम कम्पाइल करते समय कम्पाइलर clrscr(); लाइन को कम्पाइल करेगा तो प्री-प्रोसेसर द्वारा जोड़ी गई हैडर फाइल से एक फंक्शन को ले लेगा। यहाँ अगर हम conio.h हैडर फाइल को प्रोग्राम में ना जोड़ें तो कम्पाइलर को clrscr() फंक्शन नहीं मिलेगा और एरर आ जाएगी। इस प्रकार हम समझ सकते हैं कि मुख्य प्रोग्राम में बाहर से लिया गया कोई भी फंक्शन या कोड मुख्य प्रोग्राम के कम्पाइलेशन के समय कम्पाइलर को उपलब्ध करवाने के लिए प्री-प्रोसेसर का उपयोग किया जाना आवश्यक है। यह बात प्रोग्राम में ही मुख्य फंक्शन से बाहर डिफाइन किए गए मैक्रो तथा अचर (Constants) पर भी लागू होती है। इसे हम निम्न उदाहरण से समझ सकते हैं–

```
#include<stdio.h>
#define PI 3.14
viod main()
{
    float r=8.25;
    float area;
    area=PI*r*r;
    printf("Area of circle=%f", area);
}
```

उपरोक्त प्रोग्राम में main() फंक्शन को कम्पाइल करने से पहले ही #define प्री-प्रोसेसर PI का मान सेट कर देगा। अब main() फंक्शन को कम्पाइल करते समय जब कम्पाइल area=PI *r*r लाइन पर पहुँचेगा तो उसे PI का मान मिल जाएगा। ❑

प्रश्न 6 – *निम्न डायरेक्टिव्ज की उपयोगिता बतलाइये–*

Explain the usage of following directivies :

(Jan. 2013; June 2014)

(i) #line

(ii) # # ऑपरेटर्स

उत्तर–(i) #line

#line डायरेक्टिव का उपयोग पहले से बने हुए (Predefined) _LINE_ तथा _FILE_ मैक्रो के साथ किया जाता है। इसमें से _LINE_ मैक्रो वर्तमान में कम्पाइल हो रही लाइन का नम्बर return करता है तथा _FILE_ मैक्रो कम्पाइल हो रही फाइल का नाम वापस करता है।

इसका प्रारूप निम्नानुसार है–

```
#line linenumber
```

प्रोग्राम (Program) :

```
#include<stdio.h>
#line 50;
main()
```

Scanned by CamScanner

```
{
    printf("Line number=%d", _LINE_ );
    printf("File name=%s", _LILE_ );
}
```

(ii) # # ऑपरेटर्स

टोकन पोस्टिंग ऑपरेटर्स कहलाते हैं। इनका उपयोग मैक्रो में पास किए गए आरग्यूमेंट्स को आपस में जोड़कर एक मान बनाने के लिए किया जाता है। यह प्री-प्रोसेसर # # के दोनों तरफ के आरग्यूमेंट्स को आपस में जोड़कर सिंगल टोकन में बदल देता है।

उदाहरण–

```
#include<stdio.h>
#include<conio.h>
#define combine(a, b, c) a # # b # # c
main()
{
    int i;
    i=combine(7, 8, 6);
    printf("%d", i);
    getch();
}
```

Output :

786

प्रश्न 7 – *Header File क्या है? Header File को हमारे प्रोग्राम में जोड़ने के लिए Pre-Processor कैसे मदद करते हैं?*

What is a Header File ? How do pre-processor helps to join Header File on our program ?

(June 2011; Jan. 2005, 12, 13, 15)

उत्तर– C तथा C++ फंक्शन आधारित भाषाएं हैं जिनमें सभी कार्यों के लिए बने-बनाए (Inbuilt) फंक्शनों का उपयोग किया जाता है, जैसे–C भाषा में मॉनिटर पर आउटपुट देने तथा यूजर से इनपुट लेने के लिए printf() तथा scanf(), स्क्रीन क्लियर कराने के लिए clrscr(), की-बोर्ड से अक्षर इनपुट कराने के लिए getch(), आदि फंक्शनों का उपयोग किया जाता है। इस प्रकार 'C' में हमें अनेक रेडिमेड फंक्शन मिलते हैं जिनका उपयोग कर हम अपने प्रोग्राम बनाते हैं। ये फंक्शन उनके कार्यों के अनुसार अलग-अलग समूहों में बांटकर अलग-अलग फाइलों में रखे गए हैं, जो हैडर फाइलें कहलाती हैं। इन हेडर फाइलों का विस्तार नाम (Extention) .h होता है। प्रत्येक हेडर फाइल अलग-अलग तरह के फंक्शनों के समूह को संग्रहित करती है; जैसे–इनपुट, आउटपुट से संबंधित फंक्शन stdio.h हैडर फाइल में रखे गए हैं, जबकि गणितीय कार्यों से संबंधित फंक्शन math.h हैडर फाइल में संग्रहित किये गये हैं।

परिभाषा– *"'C' के अलग-अलग इनबिल्ट फंक्शनों के समूह को संग्रहित करने वाली फाइलें हैडर फाइल कहलाती हैं।"*

प्री-प्रोसेसर की भूमिका *(Jan. 2014)*– हमें अपने प्रोग्राम में जिन फंक्शनों का उपयोग करना हो उनसे सम्बन्धित हैडर फाइलों को अपने प्रोग्राम में जोड़ना आवश्यक होता है, जिसके लिए #include प्री-प्रोसेसर का उपयोग किया जाता है।

उदाहरण–

```
#include<stdio.h>
#include<conio.h>
#include <math.h>
```

Scanned by CamScanner

दिये गये उदाहरण के अनुसार ये तीन हैडर फाइलें जोड़ने पर हमें अपने प्रोग्राम में सभी स्टैन्डर्ड इनपुट-आउटपुट, कन्सोल इनपुट-आउटपुट तथा गणित से संबंधित Inbuilt फंक्शनों का उपयोग कर सकते हैं। हम देखते हैं कि इनमें से stdio.h हैडर फाइल प्रत्येक प्रोग्राम में जोड़ी जाती है, क्योंकि प्रत्येक प्रोग्राम में printf() तथा scanf() फंक्शनों का उपयोग होता ही है। इसके अलावा हम अपनी आवश्यकतानुसार अलग-अलग हैडर फाइलें अपने प्रोग्राम में include कर सकते हैं। ❑

प्रश्न 8 – *Macro क्या है? 'C' प्रोग्राम में Macro की उपयोगिता को समझाने के लिए प्रोग्राम बनाइये।*

What is Macro ? Write a program to explain the use of Macro in 'C' program.

(Jan. 2012, 16)

अथवा (Or)

Macro को उदाहरण द्वारा समझाइए।?

Explain Macros with example. *(June 2011, 12, 13, 14, 15)*

उत्तर–ऐसे जटिल समीकरण या कोड जो प्रोग्राम में एक से अधिक बार उपयोग किये जाने हों एक सांकेतिक नाम के साथ मुख्य प्रोग्राम के पहले परिभाषित किये जा सकते हैं जिन्हें हम दिये गये सांकेतिक नाम के द्वारा प्रोग्राम में कहीं भी कॉल कर सकते हैं। इस प्रकार परिभाषित किये गए कोड मैक्रो कहलाते हैं। इन्हें मुख्य प्रोग्राम के ऊपर परिभाषित करने के लिए #define प्री-प्रोसेसर डायरेक्टिव का उपयोग किया जाता है। मैक्रो का प्रारूप निम्नानुसार होता है–

```
#define MACRONAME segment of code
```

यहाँ मैक्रो का नाम सामान्यतः अंग्रेजी के बड़े अक्षरों (Capital Letters) में ही दिया जाता है ताकि वे अलग दिखाई दें। मैक्रो को मुख्य प्रोग्राम में कॉल करते समय हम उसमें आरग्यूमेंट भी पास कर सकते हैं तथा मैक्रो द्वारा return किये जाने वाले मान को मुख्य प्रोग्राम में एक वेरिएबल में संग्रहित करा सकते हैं।

मैक्रो के रूप में परिभाषित किये गये कोड को हम अलग फंक्शन बनाकर उसे मुख्य प्रोग्राम में कॉल करके उपयोग करने का तरीका भी अपना सकते हैं लेकिन इस तरीके में हर बार प्रोग्राम कॉल के स्थान से फंक्शन में प्रोग्राम कन्ट्रोल स्थानान्तरित होता है जिसमें समय अधिक लगता है, जबकि मैक्रो में प्रोग्राम को कम्पाइल करने पर बनने वाले ऑब्जेक्ट कोड में कम्पाइलर, प्रोग्राम में जहाँ-जहाँ मैक्रो को कॉल किया गया है वहाँ उसके प्री-प्रोसेसर मैक्रो टैम्प्लेट (ऊपर परिभाषित किये गये कोड) को कॉपी कर लेता है जिससे प्रोग्राम की गति बढ़ जाती है। इसलिए छोटे कोड तथा समीकरणों के लिए फंक्शन की अपेक्षा मैक्रो का उपयोग करना अधिक अच्छा होता है, जबकि बड़े कोड के लिए फंक्शन का उपयोग किया जाना चाहिए। इस विवरण को हम निम्न प्रोग्राम के माध्यम से और अच्छी तरह समझ सकते हैं–

```
#include<stdio.h>
#define AREA (z) (3.14 * z * z)
main()
{
float r1=7.5, r2=4.25, a;
a=AREA(r1);
printf("Area of first circle=%f ", a);
a=AREA(r2);
printf("Area of second circle=%f ", a);
getch();
}
```

Scanned by CamScanner

इस प्रोग्राम में हमने AREA नाम से एक मैक्रो बनाया है जो दो अलग-अलग circles का Area ज्ञात करने के लिए फंक्शन में दो बार प्रयुक्त हो रहा है। ❑

प्रश्न 9 – *एक प्रोग्राम लिखिए जो logical operaters के लिए Macro Define करें। Undefining Macro को समझाइये।*

उत्तर–प्रोग्राम (Program) :

```
#include<stdio.h>
#define AND &&
#define OR ||
main()
{
    int a=5, b=25, c=50;
    if((a<10) AND (b<50 OR c<50))
        printf("\n Given Condition is true");
    else
        printf("\n Given Condition is false");
}
```

Undefining Macro

मैक्रो को Undefine करने के लिए #undefe डायरेक्टिव का उपयोग किया जाता है। यह डायरेक्टिव परिभाषित किये गये मैक्रो को निरस्त कर देता है।

इसका सिन्टेक्स निम्नानुसार है–

```
#undef MACRO_NAME
```

यहाँ MACRO_NAME में उस मैक्रो का नाम दिया जाता है जिसे निरस्त करना है। उदाहरण–

```
#undef AREA;
```

दिये गये उदाहरण में AREA नामक मैक्रो समाप्त हो जायेगा। ❑

प्रश्न 10 – *Function तथा Preprocessor directive के बीच अंतर बताइये। C में एक प्रोग्राम लिखिए जो macro के द्वारा पाँच अंकों का average निकाले।*

उत्तर–Function तथा Preprocessor directive में अन्तर–

Main प्रोग्राम से बाहर बनाया गया प्रोग्राम का स्वतंत्र लॉक फंक्शन कहलाता है, जिसे Main प्रोग्राम में कॉल (Call) किये जाने पर प्रोग्राम का नियंत्रण निश्चित आरग्यूमेंट के साथ फंक्शन पर जाता है।

जबकि Main प्रोग्राम के ऊपर #define प्रीप्रोसेसर डायरेक्टिव के साथ परिभाषित किये गये Constants तथा Macro को मुख्य प्रोग्राम में कॉल किये जाने पर परिभाषित Constant या मैक्रो का कोड प्रोग्राम कम्पाइल करने पर मुख्य प्रोग्राम में उस स्थान पर कॉपी हो जाता है जहाँ उसे कॉल किया गया है। इस तरीके में प्रोग्राम का निमंत्रण स्थानान्तरित ना होने के कारण प्रोग्राम के Run होने की गति बढ़

Scanned by CamScanner

जाती है, अतः छोटे कोड तथा ऐसे Constant जिन्हें प्रोग्राम में एक से अधिक बार उपयोग करना है प्रीप्रोसेसर डायरेटिक्व के माध्यम से define किया जाना अच्छा होता है, जबकि बड़े कोड फंक्शन के माध्यम से कॉल किया जाना अच्छा होता है ताकि प्रोग्राम का आकार अधिक बड़ा न हो।

Macro द्वारा पाँच अंकों का औसत–

```
#include<stdio.h>
#define AVG(a, b, c, d, e) C(a+b+c+d+e)/5)
main()
   {
        int a, b, c, d, e, f;
        printf("Enter five numbers:");
        scanf("%d, "%d, "%d, "%d, "%d" &a, &b, &c, &d, &e);
             f=AVG (a, b, c, d, e);
        printf("Average="%d", f);
   }
```

प्रश्न 11– निम्नलिखित directives को समझाइये– ***(Jan. 2013; June 2014)***

(i) #error

(ii) #ifndef ***(Jan. 2016)***

(iii) #ifdef

उत्तर–(i) #error

इस डायरेक्वि का उपयोग कम्पाइलर द्वारा एरर मैसेज डिस्प्ले करने के लिए किया जाता है। इसका प्रारूप निम्नानुसार है–

```
#error message
```

#error का उपयोग कम्पाइलेशन के दौरान किसी कंडीशन के संतुष्ट ना होने की स्थिति में एरर मैसेज प्रदर्शित करने हेतु किया जाता है। #error डायरेटिक्व के साथ दिया जाने वाला मैसेज डबल कोट्स में नहीं लिखा जाता। जैसे ही प्रोग्राम कन्ट्रोल #error डायरेक्टिव पर पहुँचता है कम्पाइलेशन की प्रक्रिया रुक जाती है। इसका उपयोग मुख्यतः डीबगिंग के लिए किया जाता है।

(ii) #ifndef

#ifndef का अर्थ if not define होता है। अर्थात् इसका उपयोग किसी कोड के define ना होने की स्थिति में की जाने वाली प्रक्रिया को निर्धारित करने के लिए किया जाता है।

उदाहरण–

```
#ifndef AVG
      printf("AVG will not work");
#endif
```

यहाँ अगर हमारे प्रोग्राम में AVG मैक्रो define नहीं है तो उपरोक्त मैसेज प्रदर्शित होगा। यह डायरेक्टिव #ifdef के विपरीत अर्थ में प्रयुक्त होता है।

Scanned by CamScanner

(iii) #ifdef

इस डायरेक्टिव का उपयोग #define द्वारा किसी कोड के डिफाइन होने की स्थिति में की जाने वाली कार्यवाही को परिभाषित करने के लिए किया जाता है।

उदाहरण–

```
#ifdef MACRO_NAME
    statement1
    statement2
#endif
```

यहाँ अगर दिया गया MACRO_NAME, define किया गया है तो #ifdef के अन्तर्गत दिये गये statements क्रियान्वित होंगे अन्यथा नहीं। ☐

प्रश्न 12 – निम्न को समझाइए–

Describe the process of :

Defining functions like macros.

उत्तर–Defining functions like macros– Macros, Functions की तरह ही होते हैं; जैसे–Circle का Area Calculate करने के लिये Macro को क्रिएट करते हैं। फंक्शन के द्वारा भी Circle का एरिया Calculate कर सकते हैं। यद्यपि Macro calls, Function calls की तरह हैं लेकिन Macro तथा Function दोनों Same नहीं हैं। Macro call में Preprocessor, Macro Template को Macro Expansion Replace कर देता है, जबकि Function call में control निश्चित argument के साथ pass होता है। Macros प्रोग्राम में तेजी से क्रियान्वित करता है लेकिन Macro, प्रोग्राम के size को बढ़ा देता है, जबकि फंक्शन्स प्रोग्राम के size को कम करता है। यदि हम एक प्रोग्राम Macro को 100 बार प्रयोग करते हैं, Macro Expansion हमारे source code में विभिन्न places में जाता है, इस तरह प्रोग्राम का size बढ़ जाता है। यदि फंक्शन का प्रयोग करते हैं तो यह प्रोग्राम 100 विभिन्न places में call किया जाता है, यह प्रोग्राम में वही space लेता है।

फंक्शन में Argument passing तथा Returned value में समय लगता है, इसलिये प्रोग्राम की स्पीड slow हो जाती है। Macros इस स्थिति को Avoid करते हैं, क्योंकि ये कम्पाइलेशन से पहले source code में पहले से ही Expanded होते हैं। ☐

प्रश्न 13 – 'C' के विभिन्न प्रकार के errors बताइए। उनको किस तरह से handle किया जाता है? बताइए।

What are the different types of errors in 'C'? How are they handled? Explain.

(June 2011; Jan. 2012)

उत्तर–C भाषा में फाइल हैण्डलिंग के माध्यम से फाइल में Data को Write करते हैं या फाइल में से डाटा को Read करते हैं, तब यह Check अवश्य करना चाहिये कि File में Data सही तरीके से लिखा गया है या नहीं, या File में से Data सही तरीके से Read हुआ है या नहीं, क्योंकि यह संभव हो सकता है कि फाइल में से डाटा को Write या Read करने में किन्हीं कारणवश Errors हों। फाइल में डाटा को Write या Read करने के लिये I/O ऑपरेशन में Errors को find करने के लिये ferror( ) फंक्शन का प्रयोग करते हैं। यदि Read/Write ऑपरेशन में Errors आती हैं तो यह फंक्शन Nonzero वैल्यू को रिटर्न करता है अर्थात् यदि Read/Write ऑपरेशन सफलतापूर्वक क्रियान्वित होता है, तो यह फंक्शन Zero वैल्यू रिटर्न करता है अन्यथा Nonzero वैल्यू को रिटर्न करता है। निम्नलिखित उदाहरण में ferror( ) फंक्शन का प्रयोग किया गया है–

Scanned by CamScanner

```
# include<stdio.h>
#include<conio.h>
  void main()
  {
  FILE *fp;
  char ch;
  fp=fopen("abc.c", "w");
  while (! feof(fp))
     {
  ch=getc(fp);
  if(ferror( ))
  {
  printf("error in reading file");
  break;
  }
else
printf("%c", ch);
  }
  fclose(fp);
  getch();
}
```

इस प्रोग्राम में getc() फंक्शन स्पष्टतः Fail होगा क्योंकि File write मोड में खुली हुई है, जबकि getc() को File में से डाटा को Read करने के लिये चुना है। यहां पर ferror() फंक्शन Nonzero वैल्यू को रिटर्न करेगा। Error Message को Print करने के लिये printf() फंक्शन के स्थान पर perror() फंक्शन का प्रयोग भी कर सकते हैं।

C भाषा में मुख्य रूप से चार प्रकार की Errors होती हैं–

(i) Syntax errors

(ii) Runtime errors

(iii) Logical errors

(iv) Latent errors

(i) Syntax errors–C भाषा में बनाये गये प्रोग्राम में कमाण्ड्स को गलत तरीके से लिखते हैं तो प्रोग्राम को कम्पाइल करते समय कम्पाइलर इन errors को डिटेक्ट करता है तथा डिबलिंग विण्डो में error को लाइन नं० के साथ प्रदर्शित करता है। इस तरह की errors को Syntax errors कहते हैं, जैसे–यदि printf() Statement को सोर्स प्रोग्राम में इस तरह लिखते हैं–

```
printf("hello")
```

इस स्टेटमेण्ट में C भाषा के नियमों के अनुसार Semicolon (;) से Terminate नहीं किया है। अतः डिबगिंग विण्डो में यह error प्रदर्शित होती है।

(ii) Runtime errors–Runtime errors वह errors होती हैं जो प्रोग्राम में डिक्लेयर किये गये datatype mismatch या array element ऐरे रेन्ज से बाहर होते हैं।

इन mismatch datatypes या ऐरे एलीमेन्ट को कम्पाइलर प्रोग्राम को कम्पाइल करते समय detect नहीं कर पाता है लेकिन प्रोग्राम के क्रियान्वयन के समय सही आउटपुट को प्रदर्शित नहीं करते हैं। सामान्यतः Runtime error को हटाना मुश्किल कार्य होता है।

(iii) Logical errors–इस तरह की errors प्रोग्राम क्रियान्वयन के लॉजिक से सम्बन्धित होती हैं। इस तरह के एक्शन गलत Path को ग्रहण करते हैं। यह particular कंडीशन स्टेटमेण्ट के Evaluation के Incorrect order को समझने में सक्षम नहीं होती हैं। Logical errors कम्पाइलर के समय जनरेट error मैसेज को प्रदर्शित नहीं करती हैं। ये गलत परिणाम को दर्शाती हैं। ये errors प्रॉब्लम को सही रूप से न समझने के कारण होती हैं; जैसे–

Scanned by CamScanner

```
if(x==y)
printf("both are equal \n");
```

यहां पर यदि x तथा y float types की values हैं तो यह values बराबर नहीं होती हैं या test स्टेटमेण्ट while(x!=y) एक Infinitive loop को जनरेट करता है।

(iv) Latent errors—यह एक hidden error होती है जो तब प्रदर्शित होती है जब डाटा का Particular set का प्रयोग होता है; जैसे—

```
a=(x+y)/(p-q);
```

जब p तथा q equal होते हैं तब errors होती हैं।

❑

प्रश्न 14 – एक header file बनाने के पद लिखिये। एक उपयुक्त उदाहरण से समझाइये।

Write the steps to create a header file. Explain with a suitable example.

अथवा (Or)

Header file कैसे create की जाती है? कुछ header file के नाम बताइए।

How is header file created? Name few header files. ***(June 2011, 15; Jan. 2012, 13, 15)***

अथवा (Or)

User define header file से आप क्या समझते हैं?

What do you understand by user defined header file.

उत्तर—यदि हमारे प्रोग्राम का आकार काफी बड़ा है, इसका अच्छा तरीका यह है कि Code को अलग-अलग विभिन्न फाइलों में विभाजित कर दिया जाये, प्रत्येक फाइल में फंक्शन्स का set होता है। यह अच्छी प्रोग्रामिंग तकनीकी है कि इन बड़े प्रोग्राम के विभिन्न sections हों। इन फाइलों को Main प्रोग्राम फाइल के शुरूआत में #include के साथ लिखते हैं। Header file का Extension .h होता है।

User defined header files—यूजर के द्वारा डिफाइन की गयी हैडर फाइल को लिखने का प्रारूप निम्नलिखित है—

```
#include "file name"
```

इस प्रारूप को इस रूप में भी लिख सकते हैं—

```
#include<file name>
```

प्रथम प्रारूप का अर्थ है कि यह कमाण्ड फाइल को Current directory तथा directories की specified list में Search करती है। द्वितीय प्रारूप का अर्थ है कि यह कमाण्ड फाइल को केवल directories की specified list में Search करती है।

Header फाइल बनाने के लिये पहले हम एक फाइल बनाते हैं, इसका नाम a.c रखते हैं।

```
#include <stdio.h>
#include <conio.h>
struct emp
{
char name[10];
int age;
}e;
```

इस फाइल को save करते हैं। अब एक नई फाइल क्रिएट करते हैं। इस फाइल में a.c फाइल को Include करते हैं, इसका फायदा यह है कि a.c का स्टेटमेण्ट सीधे इस File में call कर सकते हैं, जैसे यहां पर emp structure को call करते हैं।

Scanned by CamScanner

```
#include <stdio.h>
#include <conio.h>
#include "a.c"
  void main()
  {
    printf("enter employee name\n");
    scanf("%s", e.name);
    printf("enter employee age\n");
    scanf("%d", & e.age);
    printf("employee name is % s", e.name);
    printf("employee age is %d", e.age);
    getch( );
  }
```

इस तरह a.c को Header file के रूप में इस प्रोग्राम को Include किया है। इस प्रोग्राम में a.c में डिक्लेयर किये गये स्ट्रक्चर को प्रयोग किया है।

कुछ प्रमुख header file है जैसे- #include (Stdio.h), #include (Conio.h), #include (Math.h) आदि।

❑

प्रश्न 15 – *# PRAGMA DIRECTIVE क्या है? इन्हें कैसे ON तथा OFF करते हैं? क्या एक Predefine macro को Undefine किया जा सकता है?*

What are # PRAGMA DIRECTIVE? How do you set them ON and OFF? Can you undefine a predefined macro?

उत्तर–# PRAGRAMA DIRECTIVE, defined directive का इम्प्लीमेन्टेशन है जो Compiler के लिये कई Instructions देता है। इसका प्रारूप निम्नलिखित रूप में होता है–

```
# pragma name
```

यहां पर name pragma name है जो हम देना चाहते हैं। विभिन्न Compilers के लिये Pragmas भी विभिन्न हो सकते हैं। कुछ # pragma statements निम्नलिखित हैं–

• **# pragma startup :** यह प्रोग्रामर के लिये function specify करता है जो Program startup पर call किया जाता है; अर्थात् main( ) फंक्शन को call करने से पहले यह फंक्शन Call किया जाता है।

```
# pragma startup func1
```

func1 फंक्शन main( ) फंक्शन से पहले call किया जायेगा।

• **# pragma exit :** यह प्रोग्रामर के लिये फंक्शन Specify करता है जो प्रोग्राम के exit करने पर call किया जाता है; अर्थात् प्रोग्राम के terminate होने से पहले यह फंक्शन Call किया जाता है; जैसे–

```
# pragma exit func2
```

func2() फंक्शन Program Termination से पहले Call किया जायेगा।

❑

प्रश्न 16 – *Linking एवं loading को explain कीजिए।*

Explain linking and loading.

उत्तर– Linking : Linking एक ऐसी प्रक्रिया है जो Source प्रोग्राम के कम्पाइल होने के पश्चात् होती है। C भाषा में लिखे गये Source प्रोग्राम को कम्पाइलर मशीन कोड में परिवर्तित करता है तथा क्रियान्वयन के पश्चात् रिजल्ट को स्क्रीन पर प्रदर्शित करता है। सोर्स प्रोग्राम

Scanned by CamScanner

की Linking प्रक्रिया को निम्नांकित चित्र में प्रदर्शित किया गया है–

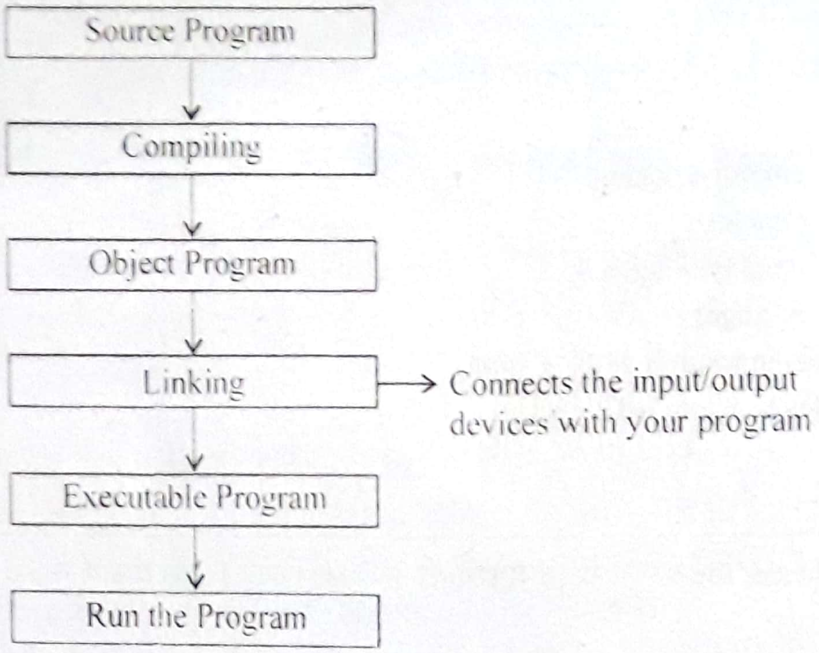


Fig : Linking Process

Overload : Overload term, फंक्शन से सम्बन्धित है। फंक्शन को तीन प्रकार से प्रयोग किया जा सकता है–

1. Function No Arguments No Return Value
2. Function with Arguments No Return Value
3. Function with Arguments with Return Value

जब Function में हम कोई Arguments Pass करते हैं तो ये Arguments किसी भी Type के हो सकते हैं तथा एक से अधिक Functions का एक ही प्रोग्राम में प्रयोग करने पर इनके नाम अलग-अलग होते हैं। हम Function एक से अधिक लेने पर इनके नाम भी एकजैसे प्रयोग कर सकते हैं लेकिन इनमें Pass किये जाने वाले Arguments भिन्न होने चाहिये, इसे Overload कहते हैं तथा फंक्शन पर इसे प्रक्रिया के होने पर इसे Function Overloading कहते हैं। निम्नलिखित उदाहरण में इसे स्पष्ट रूप से समझाया गया है–

उदाहरण :

```
#include<stdio.h>
#include<conio.h>
void show(int a, int b);          /* both function
void show(float c, float.b);      num are same but
void main()                       arguments are different */
{
int i, j;
float k, l;
i = 10;
l = 20;
k = 10.5,
l = 20.5;
show(i, l);
show(k, l);
getch();
}
void show(int a, int b)
{
```

```
    printf("%d%d", a, b);
    }
void show(float a, float b);
    {
    printf("%f % f", a, b);
    }
```

❑

प्रश्न 17 – ***Error handling* को समझाइये तथा *debugging* क्या है? विस्तार से समझाइये।**
Explain the error handling and what is debugging? Explain in detail. ***(Jan. 2012)***

उत्तर–Error handling [**नोट**–इसके उत्तर के लिए **पृष्ठ 56** पर **प्रश्न-13** के उत्तर का अध्ययन करें।]

Debugging प्रोग्राम की errors को ढूंढना तथा सही करने की प्रक्रिया को **डिबगिंग** कहते हैं। प्रोग्राम को errors को ढूंढने तथा सही करके पुनः क्रियान्वित करने के लिये सबसे अच्छा तरीका यह है कि प्रोग्राम को टुकड़ों में अर्थात् प्रत्येक statement को step by step कम्पाइल तथा क्रियान्वित करना चाहिये। इसके लिये F8 Key को Press करते हैं। इसमें प्रत्येक Flow step by step शुरू से अंत तक क्रियान्वित होकर step में प्रोग्राम की आउटपुट को दर्शाता है; जैसे–C भषा में प्रोग्राम को लिखने के पश्चात् F8 Key को Press करते हैं। एक Flow को क्रियन्वित करने के पश्चात् यह आउटपुट की स्क्रीन पर प्रदर्शित करता है। Enter Key Press करते हैं तो पुनः source को प्रोग्राम में वापिस आते हैं, पुनः F8 Key Press करते हैं। यह प्रक्रिया प्रोग्राम के अंतिम स्टेटमेन्ट तक दोहराते हैं। ❑

प्रश्न 18 – ***Macros without argument* को प्रोग्राम द्वारा समझाइये।**
What do you mean by macros without argument. Explain with Program. ***(June 2012, 13)***

उत्तर–

```
#include<stdio.h>
#include<conio.h>
#define UPPER 10// Macro
void main( )
{
int i;
for(i=i;i<=UPPER, i++)
{
printf("%d", i);
}
getch( );
}
```

इस प्रोग्राम में for लूप condition में UPPER का प्रयोग किया गया है, जिसकी value 10 को define statement के साथ preprocessor directive के रूप में डिफाइन किया है। यह लूप 10 बार क्रियान्वित होगा।

#define UPPER 10

यह स्टेटमेण्ट macro expansion या macro कहलाता है। Preprocessing के समय preprocessor प्रोग्राम में UPPER को 10 से प्रतिस्थापित कर देगा। इस तरह हम define स्टेटमेण्ट के साथ macro का प्रयोग preprocessor directive के रूप में कर सकते हैं। ❑

प्रश्न 19 – ***निम्नलिखित code का output क्या होगा?***
What will be the output of the following code? ***(Jan. 2014)***

```
# include <stdio.h)
# define START main() {
# define PRINT printf ("* * * * * * ***").
```

Scanned by CamScanner

```
# define END
        START
        PRINT
```

Justify your answer.

उत्तर–जब हम इस प्रोग्राम को run करते हैं तो हमें error message मिलते हैं, जो कि program में syntax तथा logical errors की वजह से उत्पन्न हुए हैं जैसे–

1. # include<stdio.h> का error no such directory के रूप में display होगा। इसे सही करने के लिए हमें इसे इस प्रकार लिखना होगा–
   # include<stdio.h>
2. # define printf ("******"). भी रन नहीं होगा क्योंकि printf के अन्त में . (dot) के स्थान पर ; (Semicolon) का प्रयोग किया जाना चाहिए। इसलिए हम इसे निम्न प्रकार लिखेंगे–
   # define printf ("******");
3. # define END के स्थान पर # define END} का प्रयोग होगा क्योंकि हमने प्रोग्राम में # define START main( ) { को terminate नहीं किया।
4. # define START main( ) { के स्थान पर # define Start void main( ) { का प्रयोग होगा। क्योंकि प्रोग्राम में return statement mentioned नहीं है। अतः प्रोग्राम कुछ भी return नहीं करेगा इसलिए हमें main के आगे void का प्रयोग करना पड़ेगा।
5. हमें प्रोग्राम के अन्त में END को भी call करना पड़ेगा अन्यथा हमारा प्रोग्राम propely terminate नहीं होगा।
   अतः हमारा पूरा प्रोग्राम इस प्रकार बनेगा–

```
# include <stdio.h>
# define Start void main( ){
# define Print printf("******");
# define END}
START
PRINT
END
```

Output :

```
******
```

प्रश्न 20 – *# elif directives के उपयोग को समझाइये।*

Explain the use of # elif directives. *(Jan. 2016)*

उत्तर–**# elif directives :** # elif directives का प्रयोग conditional complation directives के लिय किया जाता है। यदि हम Compiler को Source code के part से skip करना चाहते हैं, तो इसके लिये Processing commands # ifdef तथा #endif को Source कोड में Insert करते हैं। जितने time हम # if का प्रयोग करते हैं ता इसे # endif से Close करते हैं, लेकिन जब # elif का प्रयोग करते हैं तब केवल एक #endif का प्रयोग करते हैं, जैसा कि अग्रांकित प्रोग्राम में दर्शाया है–

```
#include<stdio.h>
#include<conio.h>
#define FLAG 1
  void main( )
  {
```

Scanned by CamScanner

```
int a = 10, b = 2
# if FLAG = = 0
printf("value of FLAG is zero");
a++;
b++;
# elif FLAG = = 1
printf("value of FLAG is one");
a- -;
b- -;
# elif FLAG = = 2
printf("value of FLAG is one");
a = a - 3;
b = b - 3;
# else
printf("value of FLAG is greater than two or less than zero");
a = a + b;
b = a - b;
# endif
printf("a = %d, b = %d", a, b);
printf("program completed");
getch( );
}
```

Scanned by CamScanner

UNIT-IV

प्रश्न 1 – 'C' प्रोग्रामिंग में उपयोग होने वाले विभिन्न ग्राफिक्स फंक्शनों को समझाइये।

Explain different graphics functions in 'C' programming. *(June 2006,10; Jan. 2012, 15)*
(Jan. 2013; June 2013)

उत्तर– 1. initgraph() फंक्शन

initgraph() फंक्शन का उपयोग ग्राफिक्स मॉड सक्रिय करने के लिए किया जाता है। इसका प्रारूप निम्नानुसार है–

```
initgraph(graphicsdriver, graphic_mode, driver_path);
```

यहाँ graphic_driver तथा graphic_mode को दो इन्टीजर वेरिएबलों gd तथा gm के द्वारा दर्शाया जाता है। ग्राफिक्स ड्राइवर का नाम तथा मॉनिटर का ग्राफिक्स मोड ज्ञात करने के लिए एक ड्राइवर आरग्यूमेंट DETECT का उपयोग किया जाता है। DETECT अपने आप ग्राफिक्स ड्राइवर का नाम तथा मॉनिटर के ग्राफिक्स मोड का पता लगाकर उन्हें gd तथा gm वेरिएबलों में संग्रहित कर देता है। यह कार्य निम्न प्रकार से होता है–

```
int gd, gm,
gd=DETECT;
gm=DETECT;
```

initgraph() फंक्शन में आरग्यूमेंट के रूप में उपरोक्त दोनों वेरिएबलों का ऐड्रस पास किया जाता है। driver_path उप-डायरेक्ट्री को दर्शाता है जिसमें ड्राइवर फाइल स्थित है। यह पाथ फंक्शन आरग्यूमेंट में " " में दिया जाता है। फंक्शन कॉल–

```
initgraph(&gd, &gm, "C:||TC||BGI");
```

2. closegraph() फंक्शन

closegraph() फंक्शन का उपयोग initgraph() फंक्शन द्वारा सक्रिय किये गये ग्राफिक्स मोड को समाप्त करने के लिए किया जाता है। यह फंक्शन ग्राफिक्स मोड को समाप्त करके पुनः डिफाल्ट वीडियो मोड को सेट कर देता है। इसका प्रारूप निम्नानुसार होता है–

```
closegraph();
```

3. restorecrtmode() फंक्शन

restorecrtmode() फंक्शन ग्राफिक्स मोड के समाप्त होने पर डिफॉल्ड वीडियो मोड को रीस्टोर करने का कार्य करता है। दरअसल closegrph() तथा restorecrtmode() फंक्शन एक साथ उपयोग किये जाते हैं। जिसमें closegraph() फंक्शन पहले ग्राफिक्स मोड को समाप्त करता है, इसके बाद restorecrtmode() फंक्शन वीडियो मोड को सेट करता है।

restorecrtmode() फंक्शन के साथ getgraphmode() एवं setgraphmode() फंक्शन का उपयोग ग्राफिक मोड और वास्तविक वीडियो मोड के बीच एक-दूसरे में जाने के लिए किया जा सकता है। इस फंक्शन का प्रारूप निम्न प्रकार से है–

```
restorecrtmode();
```

4. floodfill() फंक्शन

(June 2013)

इस फंक्शन का उपयोग किसी आकृति में रंग भरने के लिए किया जाता है। इसका प्रारूप निम्नानुसार है–

```
floodfill (x, y, color);
```

यहाँ x तथा y आरग्यूमेंट आकृति के उन आंतरिक बिन्दुओं को बतलाते हैं जिनके बीच रंग भरना है तथा color आकृति की बाह्य रेखा outline() के रंग को बतलाता है, यह बाह्य रेखा का रंग यह निश्चित करता है, कि जैसे ही यह रंग अर्थात् आकृति की बाह्य रेखा मिल जाएगी, रंग भरने की प्रक्रिया रुक जायेगी।

Scanned by CamScanner

5. outtext() फंक्शन

(Jan. 2012)

outtext() फंक्शन का उपयोग ग्राफिक्स मोड में रहते हुए टेक्स्ट मैसेज प्रदर्शित करने के लिए किया जाता है। इसका प्रारूप निम्नानुसार है–

```
outtext("message");
```

उदाहरण–

```
outtext("welcome");
```

6. outtextxy() फंक्शन

(Jan. 2012)

outtextxy() फंक्शन का उपयोग स्क्रीन पर किसी निश्चित स्थान पर टेक्स्ट मैसेज प्रदर्शित करने के लिए किया जाता है। इस फंक्शन मैसेज के साथ दो और आरग्यूमेंट दिये जाते हैं जो x और y अंश पर उस बिन्दु का निर्धारण करते हैं जहाँ से टेक्स्ट मैसेज प्रदर्शित करना है; जैसे–

```
Outtextxy (a, b, "message");
```

उपरोक्त प्रारूप में a, x अक्ष का स्थान एवं b, y अक्ष का स्थान निर्धारित करता है।

उदाहरण–

```
outtextxy (50, 200, "welcome");
```

इस उदाहरण में x अक्ष पर 50वें और y अक्ष पर 200वें स्थान पर "welcome" संदेश प्रदर्शित होगा।

7. line() फंक्शन

यह फंक्शन एक सीधी रेखा खींचने का कार्य करता है। इसमें चार आरग्यूमेंट दिये जाते हैं जिनमें पहले दो आरग्यूमेंट x तथा y अक्ष पर रेखा का आरम्भिक बिन्दु निर्धारित करते हैं, इसी तरह दूसरे दो आरग्यूमेंट x तथा y अक्ष पर रेखा का अंतिम बिन्दु निर्धारित करते हैं। **प्रारूप–**

```
line(x1, y1, x2, y2);
```

उदाहरण–

```
line(20, 30, 50, 30);
```

दिये गये उदाहरण में रेखा 20वें कॉलम व 30वीं रो से आरम्भ होकर 50वें कॉलम 30वीं रो पर समाप्त होगी।

(June 2013)

8. setlinestyle() फंक्शन

setlinestyle() फंक्शन का उपयोग खींची जा रही लाइन की style निर्धारित करने के लिए किया जाता है। इसका प्रारूप निम्नानुसार है–

```
setlinestyle (m, n, t);
```

यहाँ m एक इन्टीजर मान है जो रेखा की style निर्धारित करता है। n एक अनसाइन्ड इन्टीजर मान है जो रेखा का user defined पैटर्न निर्धारित करता है। n का मान केवल उस स्थिति में दिया जाता है जब पहले आरग्यूमेंट m द्वारा लाइन Style USERBIT_LINE तय की गई हो बाकी सभी स्थितियों में n का मान शून्य रखा जाता है। लाइन की Style USERBIT_LINE निर्धारित करने के लिए m का मान 4 दिया जाता है।

फंक्शन में t तीसरा इन्टीजर मान है जो रेखा की मोटाई निर्धारित करता है। पहले आरग्यूमेंट m के माध्यम से दी जाने वाली सम्भावित लाइन Style निम्नलिखित होती है–

Scanned by CamScanner

Style Name	Integer Value	Style
SOLID_LINE	0	————
DOTTED_LINE	1	....
CENTER_LINE	2	-.-.-.-.-.-
DASHED_LINE	3	- - - - - -
USERBIT_LINEI	4	Underlined Style

तीसरे आरग्यूमेंट t के द्वारा लाइन की मोटाई निम्न प्रकार से निर्धारित की जाती है–

Line_Width	Integer_Value	Meaning
Normal_Width	1	Equal to 1 Pixel Width
Thick_Width	3	Equal to 3 Pixel Width

उदाहरण–

```
setlinestyle(1, 0, 3);
```

या

```
setlinestyle(DOTTED_LINE, 0, THICK_WIDTH);
```

इसके द्वारा अधिक मोटाई की डॉटिड रेखा बनेगी।

9. circle() फंक्शन

इस फंक्शन का उपयोग वृत्त बनाने के लिए किया जाता है। इसमें तीन आरग्यूमेंट दिये जाते हैं, जिनमें पहले दो आरग्यूमेंट x तथा y अक्षरों के सापेक्ष वृत्त का केन्द्रबिन्दु निर्धारित करते हैं तथा तीसरा आरग्यूमेंट वृत्त की त्रिज्या को दर्शाता है।

प्रारूप–

```
circle(x, y, r);
```

उदाहरण–

```
circle (50, 100, 40)
```

दिये गये उदाहरणों में 50, 100 केन्द्र निर्देशांक तथा 40 त्रिज्या के साथ एक वृत्त बनेगा।

10. ellipse() फंक्शन

इस फंक्शन का उपयोग एलिप्स (परवलय) बनाने के लिए किया जाता है। इस फंक्शन में आरग्यूमेंट के रूप में परवलय के केन्द्र का x तथा y मान, परलवय के प्रारम्भिक बिन्दु का कोण, परवलय के अंतिम बिन्दु का कोण, परवलय की दीर्घ अक्ष तथा परवलय की लघु अक्ष दी जाती है।

प्रारूप–

```
ellipse(x, y, sa, ea, lr, sr);
```

यहाँ x एवं y इन्टीजर मान हैं जो परवलय के केन्द्र को निर्धारित करते हैं।

sa एक इन्टीजर मान है जो परलवय के प्रारम्भिक बिन्दु का कोण प्रदर्शित करता है।

ea एक इन्टीजर मान है जो परवलय के अंतिम बिन्दु का कोण निर्धारित करता है।

lr यह इन्टीजर मान परवलय के अर्ध-दीर्घ अक्ष को दर्शाता है।

sr यह इन्टीजर मान परलवय के अर्ध-लघु अक्ष को दर्शाता है।

उदाहरण–

```
ellipse(150, 200, 0, 360, 60, 25);
```

यहाँ 150, 200 केन्द्रीय निर्देशांक पर 0 अंश प्रारम्भिक बिन्दु के कोण से 360 अंश अंतिम बिन्दु के कोण तक 120 दीर्घ अक्ष (60+60) तथा 50 लघु अक्ष (25+25) का एक परवलय बनेगा।

Scanned by CamScanner

11. arc() फंक्शन

इस फंक्शन का उपयोग निर्धारित केन्द्र एवं त्रिज्या पर दिये गये प्रारम्भिक बिन्दु से अंतिम बिन्दु तक एक ऐसा चाप बनाने के लिए किया जाता है।

प्रारूप–

```
arc(x, y, sa, ea, r);
```

यहाँ x तथा y इन्टीजर चाप के केन्द्र को निर्धारित करते हैं।

sa इन्टीजर चाप के प्रारम्भिक बिन्दु का कोण निर्धारित करता है।

ea चाप के अंतिम बिन्दु का कोण निर्धारित करता है।

r इन्टीजर चाप की त्रिज्या को प्रदर्शित करता है।

12. setfillstyle() फंक्शन

setfillstyle() फंक्शन किसी आकृति में एक पैटर्न भरने के लिए उपयोग किया जाता है। इस फंक्शन में दो आरग्यूमेंट होते हैं जिनमें पहला आरग्यूमेंट पैटर्न निर्धारित करता है तथा दूसरा आरग्यूमेंट भरे जा रहे पैटर्न को निर्धारित करता है। **प्रारूप–**

```
setfillstyle(P, C);
```

यहाँ p इन्टीजर पैटर्न तथा c इन्टीजर मान रंग को निर्धारित करता है। इस फंक्शन के द्वारा निम्नलिखित पैटर्न दिये जा सकते हैं–

Pattern	Integer Value	Meaning
EMPTY_FILL	0	Background color
SOLID_FILL	1	Solid color
LINE_FILL	2	Horizontal lines
LTSLASH_FILL	3	|||| pattern
SLASH_FILL	4	Thick ||| pattern
BKSLASH_FILL	5	Thick ||| pattern
LTBKSLASH_FILL	6	||| pattern
HATCH_FILL	7	Thin cross hatch
XHATCH_FILL	8	Thick cross hatch
INTERLEAVE_FILL	9	Interleaving lines
WIDE_DOT_FILL	10	Widely spread dots
CLOSE_DOT_FILL	11	Densely packed dots
USER_FILL	12	User define fill pattern

13. putpixel() फंक्शन

यह फंक्शन x तथा y आरग्यूमेंट के माध्यम से निर्दिष्ट pixel को दिये गये रंग में हाइलाइट (Highlight) करता है। यह फंक्शन पूरी स्क्रीन को एक ग्राफ के रूप में देखता है तथा दिये गये रंग में निर्दिष्ट pixel को प्रदर्शित करता है।

प्रारूप–

```
putpixel(x, y, color);
```

उदाहरण–

```
putpixel(20, 30, 1);
```

अथवा

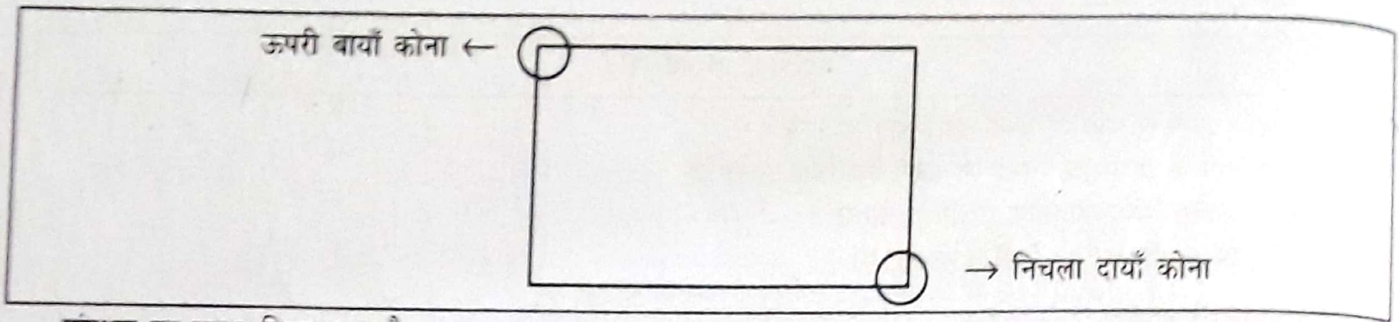
```
putpixel(20, 30, BLUE);
```

उपरोक्त उदाहरण 20वें कॉलम तथा 30वीं रो में नीले रंग में एक पिक्सेल प्रदर्शित करेगा।

Scanned by CamScanner

14. bar() फंक्शन

bar() फंक्शन एक आयत बनाने के लिए उपयोग किया जाता है। इसमें चार आरग्यूमेंट दिये जाते हैं जिनमें से पहले दो आरग्यूमेंट आयत के ऊपरी बायें कोने को प्रदर्शित करते हैं तथा दूसरे दो आरग्यूमेंट निचले दाहिने कोने को प्रदर्शित करते हैं जिससे आयत का आकार निर्धारित होता है।



फंक्शन का प्रारूप निम्नानुसार है–

```
bar(left, top, right, bottom);
```

15. bar3d() फंक्शन

यह फंक्शन एक त्रि-आयामी आयत का निर्माण करता है। इसका प्रारूप निम्नानुसार है–

```
bar3d(x, y1, x2, y2, d, tf);
```

यहां x1, y1 ऊपरी बायें कोने के निर्देशांक को प्रदर्शित करता है।

x2, y2 निचले दायें कोने के निर्देशांक को प्रस्तुत करता है।

d एक इन्टीजर मान है जो आयत की गहराई को दर्शाता है, इसी से आयत का तीसरा आयाम बनता है।

tf यह इंटीजर मान टॉप फ्लेग को दर्शाता है, जिसका मान आयत के ऊपरी बायें कोने के निर्देशांक में जोड़ा जाता है। ❑

प्रश्न 2 – 'C' ग्राफिक्स में प्रयुक्त Color फंक्शनों का वर्णन कीजिए।

Explain color function use in 'C' graphics. *(Jan. 2004, 12)*

उत्तर–'C' के ग्राफिक्स फंक्शनों के अन्तर्गत आने वाले Color से सम्बन्धित फंक्शन निम्नलिखित हैं–

1. Setcolor() फंक्शन–

यह फंक्शन ग्राफिक्स चित्र का रंग निर्धारित करता है। इसका प्रारूप निम्नानुसार होता है–

```
setcolor(m);
```

यहाँ m एक पूर्णांक या रंग का नाम होता है, जैसे–

```
setcolor(4);
```

या

```
setcolor(RED);
```

यह ड्रॉइंग के लिए लाल रंग निर्धारित करेगा।

2. Setbkcolor() फंक्शन

यह फंक्शन स्क्रीन का बैकग्राउण्ड का रंग निर्धारित करता है इसका प्रारूप निम्न है–

```
setbk color(m);
```

यहाँ m एक पूर्णांक या बैक ग्राउण्ड रंग का नाम होता है।

Scanned by CamScanner

उदाहरण–

```
setbkcolor();
```

या

```
setbkcolor(BLUE);
```

यहाँ स्क्रीन का बैक ग्राउण्ड रंग नीला होगा।

3. getcolor() फंक्शन

इस फंक्शन का उपयोग चित्र में फोरग्राउण्ड का रंग देने के लिए किया जाता है–

```
fcolor=getcolor();
```

यहाँ fcolor एक इन्टीजर वेरिएबल है।

4. getbkcolor() फंक्शन

(June 2011)

getbkcolor() फंक्शन का उपयोग वर्तमान बैकग्राउण्ड का रंग देने हेतु किया जाता है–

```
bcolor=getbkcolor();
```

यहाँ bcolor एक इन्टीजर वेरिएबल है।

5. getmaxcolor() फंक्शन

getmaxcolor() फंक्शन चुने गए ग्राफिक्स मोड द्वारा समर्थित अधिकतम रंगों की संख्या बतलाता है। ग्राफिक्स मोड द्वारा समर्थित अधिकतम रंगों की वास्तविक संख्या getmaxcolor() फंक्शन द्वारा return की जा रही संख्या से एक अधिक होगी, क्योंकि रंगों की गिनती शून्य से प्रारम्भ होती है। इसका प्रारूप निम्नानुसार है–

```
maxclrs=getmaxcolor();
```

यहाँ maxclrs एक इन्टीजर वेरिएबल है।

❑

प्रश्न 3 – C भाषा में ग्राफिक्स का उपयोग कैसे करेंगे? उदाहरण सहित बताइए।

How do you handle graphics in C language? Explain with example. *(Jan. 2014)*

अथवा (Or)

C में ग्राफिक्स मोड किस प्रकार सक्रिय किया जाता है?

How to activate graphic mode in C. *(Jan. 2004, 11, 16; June 2011, 13)*

उत्तर–Graphics Mode सैट करना

ग्राफिक्स मोड सेट करने के लिए initgraph() फंक्शन का उपयोग किया जाता है। इस फंक्शन में आरग्यूमेंट के रूप में ग्राफिक्स ड्राइवर का नाम, मॉनिटर का ग्राफिक्स मोड तथा ड्राइवर का पाथ दिया जाता है। इसका प्रारूप निम्नानुसार होता है–

innitgraph(graphic_driver, graphic_mode, driverpath);

यहाँ ग्राफिक्स ड्राइवर का पता लगाने के लिए ड्राइवर आरग्यूमेंट DETECT का उपयोग किया जाता है। ग्राफिक्स मोड भी DETECT आरग्यूमेंट के द्वारा स्वतः ही निर्धारित हो जाता है। ग्राफिक्स मोड तथा ग्राफिक्स ड्राइवर दोनों ही एक पूर्णांक (Integer) के रूप में होते हैं।

उदाहरण–

```
int gd, gm;
gd=DETECT;
initgraph (&dg, &gm, "C:||tc||bgi");
```

Scanned by CamScanner

getmaxx()

इस फंक्शन का उपयोग x कोर्डिनेट का अधिकतम मान प्राप्त करने के लिए किया जाता है।

getmaxy()

इस फंक्शन का उपयोग y कोर्डिनेट का अधिकतम मान प्राप्त करने के लिए किया जाता है।

getgraphmode()

इस फंक्शन का उपयोग वर्तमान ग्राफिक्स मोड का मान प्राप्त करने के लिए किया जाता है।

setgraphmode()

इस फंक्शन का उपयोग ग्राफिक्स मोड का मान सेट करने के लिए किया जाता है। ☐

प्रश्न 4 – Graphics Mode तथा Text Mode के बीच अंतर बताइये।

Differentiate between Graphics Mode and Text Mode. ***(June 2011)***

उत्तर–टेक्स्ट मोड (Text Mode)

- Text मोड का उपयोग अक्षरों को प्रदर्शित करने के लिए किया जाता है। अर्थात् इस मोड में केवल टाइप किया गया Text ही प्रदर्शित होता है ग्राफिक्स ऑब्जेक्ट नहीं।
- इस मोड में सभी संभावित 256 अक्षर तथा चिह्न आदि प्रदर्शित होते हैं।
- स्क्रीन पर 80 कॉलम तथा 25 रो (row) के माध्यम से 25 पंक्तियां तथा प्रत्येक पंक्ति में 80 अक्षर प्रदर्शित हो सकते हैं।
- '\n' के द्वारा हम अगली पंक्ति पर जा सकते हैं।

ग्राफिक्स मोड (Graphics Mode)

- इस मोड का उपयोग ग्राफिक्स ऑब्जेक्ट प्रदर्शित करने के लिए किया जाता है।
- ग्राफिक्स निर्देश सिर्फ ग्राफिक्स मोड में ही कार्य करते हैं।
- ग्राफिक्स मोड में टेक्स्ट आउटपुट प्राप्त करने के लिए विशेष फंक्शनों का उपयोग करना पड़ता है।
- ग्राफिक्स रिजॉल्यूशन कम्प्यूटर के प्रकार तथा ग्राफिक्स कार्ड पर निर्भर होता है।
- ग्राफिक्स आउटपुट स्क्रीन पर किसी भी विन्दु से आरम्भ किया जा सकता है। ☐

प्रश्न 5 – एक वृत्त बनाते हुए ग्राफिक्स की अवधारणा का उपयोग कर एक 'C' प्रोग्राम लिखिए।

Write a program to draw a circle using Graphics in 'C'. ***(June 2006, 15; Jan. 2013, 16)***

उत्तर–

```
#include<stdio.h>
#include<conio.h>
#include<graphics.h>
#include<stdlib.h>
main()
{
    int gd=DETECT;
    int gm;
    initgraph (&gd, &gm, "C||TC||BGI");
    circle(100, 100, 10);
    getch();
    close graph();
    restorecrtmode();
}
```

☐

Scanned by CamScanner

प्रश्न 6 – 'C' language में Graphics का प्रोग्राम बनाइए तथा इससे ellipse draw कीजिए।
Write a graphics program in C to display ellipse. *(June 2012)*

उत्तर–प्रोग्राम (Program) :

```
#include<stdio.h>
#include<conio.h>
#include<stdlib.h>
#include<graphics.h>
int main(void)
   {
      /*auto detections */
      int gdriver=DETECT, g mode;
      int midx, midy;
      int stangle=0, endangle=360;
      int xradius=100, y radius=50;
   {
      /* set graphic maode */
   }
      initgraph(&gdriver, &gmode," ");
   }
         mid x=getmaxx()/2;
         mid y=getmaxy()/2;
         setcolor(getmaxcolor());

         /* draw ellipse */
   }
         ellipse(midx, midy, stangle, endangle, xradius, yradius);
         fillellipse (midx, midy, xradius, yradius);
         getch();
         closegraph();
         return 0;
   }
```

प्रश्न 7 – निम्न से आप क्या समझते हैं–
What do you understand by :
(i) Graphics Mode **(June 2011; Jan. 2012)**
(ii) Video Adapter. **(June 2011)**

उत्तर–(i) Graphics Mode–C भाषा में जब हम कोई प्रोग्राम बनाते हैं तो इसकी आउटपुट साधारण Text फॉर्मेट में प्रदर्शित होती है। सामान्यतः background blue color में तथा foreground white color में प्रदर्शित होती है। मॉनीटर का प्रयोग ग्राफिक्स को प्रदर्शित करने में किया जाता है। मॉनीटर एक video display adapter circuit board के माध्यम से CPU के साथ जुड़ा होता है। एक Color मॉनीटर VGA Card 640 × 480 Pixels Resolution के ग्राफिक मोड को सपोर्ट करता है। यह Card 16 Colors को सपोर्ट करता है। मॉनीटर स्क्रीन पर एक Pixel एक डॉट (·) होता है। इस तरह के कई Dots (....) Horizontally तथा Vertically रूप में जुड़े रहते हैं, इन्हें मॉनीटर का Resolution कहते हैं। यदि हम मॉनीटर की स्क्रीन पर Text का Color अलग-अलग रंगों में अपने प्रोग्राम के माध्यम से बदलना चाहते हैं तो इसे बदल सकते हैं। इसी तरह Screen की background का Color का रंग, Characters का Style बदलना चाहते हैं तो यह भी कर सकते हैं। Circle, Rectangel, Line इत्यादि Graphics को भी प्रोग्राम के माध्यम से बना सकते हैं। C भाषा

Scanned by CamScanner

में ऐसा करने के लिये पहले से ही graphics फंक्शन्स graphics.h हैडल फाइल में उपस्थित है। जब प्रोग्राम में इन फंक्शन्स का प्रयोग करते हैं तब graphics.h हैडर फाइल के सोर्स प्रोग्राम में दूसरी Header files की तरह Include करते हैं।

मॉनीटर पर किसी Picture को प्रदर्शित करने के लिये Resolution तथा Color को set करते हैं। सभी Graphic functions graphics.h हैडर फाइल में उपलब्ध होते हैं।

Graphics Mode को set करने के लिये initgraph( ) फंक्शन का प्रयोग किया जाता है। इसका प्रारूप निम्नलिखित रूप में होता है–

```
initgraph(graphic_driver, graphic_mode, driver_path);
```

यहां पर graphic driver मॉनीटर टाईप का इंटीजर है। DETECT driver Argument का प्रयोग मॉनीटर टाईप को detect करने के लिये किया जाता है।

graphic_mode भी मॉनीटर टाईप का इंटीजर है जो स्वतः ही DETECT driver का प्रयोग करके set हो जाता है।

driver_path उस Directory को दर्शाता है, जिसमें driver File स्टोर होती है। Directory को " " या turbo से Specified करेंगे जैसे–

```
int gd, gm;
gd=DETECT;
initgraph(&gd, &gm, " ");
```

(ii) Video Adapter– Video Adapter, graphic driver होता है। graphic driver मॉनीटर टाईप का इंटीजर है। जिस मॉनीटर पर graphics का प्रयोग किया जाता है उस मॉनीटर के graphic driver या Video Adapter का इंटीजर टाईप का वेरियेबल डिक्लेयर किया जाता है जो स्वतः ही DETECT driver का प्रयोग करके set हो जाता है। इसके लिये initgraph( ) फंक्शन का प्रयोग किया जाता है। इस फंक्शन में तीन आर्ग्यूमेन्ट्स Pass किये जाते हैं। प्रथम आर्ग्यूमेन्ट graphic driver का रेफरेन्स है। द्वितीय आर्ग्यूमेन्ट graphic mode का रेफरेन्स है। तीसरा आर्ग्यूमेण्ट उस सिस्टम में लोड की गयी C का पूरा Path है। निम्नलिखित उदाहरण में यह फंक्शन प्रयोग किया गया है–

```
#include<stdio.h>
#include<conio.h>
#include<graphics.h>
  void main()
  {
    int gd, gm, x1, y1, x2, y2;
    gd=DETECT;
    clrscr( );
printf("enter start point co-ordinate");
scanf("%d%d", & x1, &y1);
printf("enter end point co-ordinate ");
scanf("%d%d", &x2, &y2);
initgraph(&gd, &gm, " ");
line(x1, y1, x2, y2);
  getch( );
  closegraph( );
}
```

Scanned by CamScanner

प्रश्न 8 – C में graphics functions की सहायता से एक cube बनाने के लिये प्रोग्राम लिखिए।
Write a program in 'C' graphics to draw a cube with the help of graphics functions.

उत्तर–

```
#include<stdio.h>
#include<conio.h>
#include<graphics.h>
void main()
{
int gd, gm, s;
gd = DETECT;
clrscr();
printf("enter side of cube");
scanf("%d", &s);
initgraph( &gd, &gm, " ");
lineto(50 + s, 55);
lineto(50+s, 55 + s);
lineto(50, 55 + s);
lineto(50, 55);
getch();
}
```

प्रश्न 9 – रेक्टेगंल (आयत) खीचने के लिए C प्रोग्राम लिखिए तथा इसे लाल रंग से भरिए।
Write a C program to draw a rectangle and fill it with red color. ***(June 2012, 15; Jan. 2013)***

उत्तर–

```
#include<stdio.h>
#include<conio.h>
#include<graphics.h>
void main()
{
int gd, gm, x1, y1, x2, y2;
gd = DETECT;
clrscr();
printf("enter left top coordinates(x1, y1)");
scanf("%d %d", & x1, & y1);
printf("enter right battom coordinates(x2, y2);
scanf("%d %d", &x2, &y2);
initgraph(&gd, &gm, " ");
setbk color(WHITE);
set coler(BLUE);
set lines type(SOLID_LINE, 0, THICK_WIDTH);
set fillstyle(LTSCASH_FILL_RED);
```


Scanned by CamScanner

```
        rectange(x1, y1, x2, y2);
        floodfill(x1 + 4, y1+4, BLUE);
        outtextry(10, 250, "Press enter key to exit");
    getch();
    closegraph();
    }
```

प्रश्न 10 – *Graphics mode में text को आप कैसे display करेंगे? 'C' program के साथ explain कीजिए।*
How do you display Text in graphics mode? Explain with 'C' program.

(June 2011; Jan. 2012)

उत्तर–Graphics mode में Text को प्रदर्शित करने के लिये cputs( ) फंक्शन का प्रयोग करते हैं। इसका प्रारूप निम्नलिखित रूप में होता है–

```
cputs ("message");
```

double quotes(" ") में लिखा गया मैसेज Graphics Mode में प्रदर्शित होता है। Graphic Mode में Text को प्रदर्शित करने के लिये graphics.h हैडर फाइल को प्रोग्राम में Include करते हैं। निम्नलिखित उदाहरण में 'C' भाषा में Text function का प्रयोग करके दो Numbers में से बड़े Number को प्रदर्शित करने का प्रोग्राम लिखा है–

```
#include<stdio.h>
#include<conio.h>
#include<graphics.h>
    void main()
    {
    int i, j;
    clrscr();
    text mode(c80);
    text background(BLUE);
    text color(YELLOW);
    gotoxy(30, 20);
    cputs("Enter two Numbers");
    scanf("%d %d", & i, & j);
    Text color(RED + BLINK);
    if(i > j);
    {
        gotoxy(30, 25);
        pritf("%d", i);
        cputs("is the biggest Number");
    }
else
    {
    gotoxy(30, 15);
    printf("%d", j);
    cputs("is the biggest Number");
    }
```

Scanned by CamScanner

```
getch();
}
```

❑

प्रश्न 11 – *Graphics में resolution को समझाइये।*

What is resolution in graphics? Explain. ***(June 2011; Jan. 2016)***

उत्तर–Resolution एक छवि (image) के तीखेपन (sharpness) और स्पष्टता को संदर्भित करता है। यह शब्द अक्सर मॉनीटर, प्रिंटर और बिट-मैप ग्राफिक्स इमेज का वर्णन करने के लिए प्रयोग किया जाता है। Dot-matrix और लेजर प्रिंटर के केस में, resolution डॉट्स की संख्या प्रति इंच को इंगित करता है। उदाहरण के लिए, 300-dpi (dots per inch) प्रिंटर वह है जो 300 अलग डॉट्स को एक लाइन में 1 इंच लम्बाई में प्रिंट करने में सक्षम हैं। इसका अर्थ हुआ कि वह प्रिंटर 90,000 डॉट्स प्रति वर्ग इंच में प्रिंट कर सकता है।

ग्राफिक्स मॉनीटर के लिए, स्क्रीन resolution पूरी स्क्रीन पर डॉट्स (pixels) की संख्या का प्रतीक है। उदाहरण के लिए, एक 640-by-480 पिक्सेल स्क्रीन 480 लाइनों में से प्रत्येक पर 640 अलग डॉट्स प्रदर्शित करने में सक्षम है या कहिए लगभग 300,000 पिक्सेल। स्क्रीन के आकार के अनुसार इसे हम अलग dpi माप में translate कर सकते हैं। उदाहरण के लिए, एक 15 इंच VGA मॉनीटर (640×480) 50 डॉट्स प्रति इंच प्रदर्शित करता है।

प्रिंटर, मॉनीटर, स्कैनर और अन्य इनपुट/आउटपुट डिवाइस को अकसर high resolution, medium resolution or low resolution, के रूप में वर्गीकृत किया जाता है। लगातार इन वर्गों के लिए वास्तविक resolution श्रेणियाँ shift हो रही है जैसे–जैसे–प्रौद्योगिकी में सुधार हो रहा है।

अतः अब हम यह कह सकते है कि कम्प्यूटर ग्राफिक्स के क्षेत्र में शब्द resolution के विभिन्न अर्थ है। इसका अर्थ निर्भर करता है कि यह किस संदर्भ में प्रयोग किया जा रहा है।

Image resolution इमेज के विवरण का वर्णन करती है। इस term का प्रयोग raster digital images, film images और अन्य प्रकार की इमेज के लिए किया जाता है। Higher resolution का अर्थ है अधिक इमेज विवरण।

डिजिटल इमेज के resolution को विभिन्न तरीको से बताया जा सकता है। यह तरीके हैं–

(1) Pixel resolution
(2) Spatial resolution
(3) Spectral resolution
(4) Temporal resolution
(5) Radiometric resolution
(6) Distinguishable squares

Optical resolution इमेजिंग सिस्टम की क्षमता का वर्णन करता है कि वह कितने अच्छे से उस आब्जेक्ट की बारीकियाँ दिखाता हैं जिसकी इमेज ली जा रही है। एक इमेजिंग सिस्टम के कई व्यक्तिगत घटक हो सकते है। इनमें लेंस और रिकॉर्डिंग और डिस्प्ले घटक भी शामिल हैं। इनमें से प्रत्येक घटक सिस्टम के optical resolution के लिए योगदान देता है। अतः optical resolution वह physical resolution है जिस पर एक डिवाइस इमेज को capture कर सकता है। इस शब्द का प्रयोग अकसर optical scanners और digital cameras के संदर्भ में किया जाता है। उदाहरण के लिए, अधिकतर scanners का optical resolution 300 dpi होता है। इसका अर्थ हुआ कि scanners वास्तव में 90,000 pixels प्रति square inch को capture कर सकता है। ❑

प्रश्न 12 – *निम्नलिखित को समझाइये :* ***(Jan. 2012)***

Explain the following :

(i) line $(x_1, y_1, x_2, y_2, 1)$

(ii) circle (x, y, r)

उत्तर–(i) line(x, y, x_2, y_2) : line() फंक्शन का प्रयोग मॉनीटर स्क्रीन पर Line को draw करने के लिये किया जाता है। line( ) फंक्शन में स्क्रीन के Co-ordinate position को argument के रूप में पास करते हैं। इसका प्रारूप निम्नलिखित रूप में होता है–

```
line(x1, y1, x2, y2)
```

यहां पर x1 तथा y1 line के starting co-ordinate है तथा x2, y2 ending coordinate है। जैसे–

```
line(60, 30, 250, 150);
```

Scanned by CamScanner

यह स्टेटमेन्ट स्क्रीन पर 60, 30 starting point से 250, 150 ending point तक एक लाइन को draw करता है।

(ii) Circle(x,y,r) : Circle() फंक्शन का प्रयोग circle की radius तथा center को specify करके circle को draw करने के लिये किया जाता है। इसका प्रारूप निम्नलिखित रूप में होता है–

```
Circle(x,y,z);
```

यहां पर x तथा y circle के center इंटीजर्स है तथा r circle की radius इंटीजर वैल्यू है। जैसे–

```
Circle(170, 200, 40);
```

यह स्टेटमेन्ट 170,200 center इंटीजर पॉइन्ट से 40 radius के अनुसार circle को draw करेगा।

प्रश्न 13 – ग्राफिक मोड इनीसियेट (प्रारंभ करना) करने के लिए C ग्राफिक में एक प्रोग्राम लिखिए तथा एक लाइन खींचिए।

Write a program in C graphic to initiate graphic mode and draw a line. *(June 2012)*

उत्तर–

```
#include<stdio.h>
#include<conio.h>
#include<graphics.h>
void main( )
{
int gd, gm, x1, y1, x2, y2;
gd=DETECT;
clrscr( );
printf("enter start point co-ordinate:");
scanf("%d%d", &x1, &y1);
printf("enter end point co-ordinate");
scanf("%d%d", &x2, &y2);
initgraph(&gd, &gm, " ");
setbkcolor(BLUE);
setcolor(RED);
line(x1, y1, x2, y2);
outtextxy(25, 350, "press enter key to exit");
getch( );
closegraph( );
}
```

प्रश्न 14 – ग्राफिक्स मोड में आप फाइल का निर्माण कैसे करेंगे? उदाहरण सहित बताइए।

How do you create files in graphics mode? Explain with example. *(Jan. 2013, 16)*

उत्तर–[नोट–इसके उत्तर के लिए **पृष्ठ 71** पर **प्रश्न-7(i)** एवं **पृष्ठ 69** पर **प्रश्न-3** के उत्तर का अध्ययन करें।]

उदाहरण :

```
#include <stdio.h>
#include <cenio.h>
# include <graphics.h>
void main ( )
{
int gd, gm, x1, y1, x2, y2;
```

Scanned by CamScanner

```
gd = DETECT;
clrscr ( );
printf ("enter start point co-ordinate");
scanf ("%d%d", &x1, &y1);
printf ("enter end point co-ordinate");
scanf ("%d%d", &x2, &y2)
initgraph (&gd, &gm, " ");
setbkcolor (BLUE);
Setcolor (RED);
Line (x1, y1, x2, y2);
Outtextxy (25, 350, "Press enter key to exit");
getch ( ):
closegraph ( );
}
```

प्रश्न 15 – '*C*' भाषा में लाइन फंक्शन का उपयोग करके एक *rectangle* का निर्माण कीजिए एवं उसे *fill* भी कीजिये।

Write a 'C' program to draw rectangle using line function and then fill it.

(Jan. 2013; June 2013, 15)

उत्तर–

```
#include <stdio.h>
#include <conio.h>
#include <graphics.h>
  void main()
{
  int gd, gm, s, p;
  get = DETECT;
  clrscr();
printf("enter length of rectangle");
scanf("%d", &s);
printf("enter breadth of rectangle");
scanf("%d", &p);
initgraph(&gd, &gm, " ");
setbkcolor(GREEN);
setcolor(YELLOW);
setlinestyle(SOLID_LINE, o, THICK, WIDTH);
setfillstyle(LTSCASH_FILL, RED);
moveto(50,55);
lineto(50+s, 55);
lineto(50, 55+p);
lineto(50+s, 55+p);
moveto(50, 50+p);
lineto(50+s, 55+p);
floodfill(x1 + 4, y1 + 4, BLUE);
```

Scanned by CamScanner

```
outtextxy(25, 350, "press enter to exit");
getch();
}
```

प्रश्न 16 – ग्राफिक्स मोड को उपयोग करते हुए एक Box बनाने हेतु C program लिखिए और उसको Screen पर Move करते मोड में दिखाइये।

Write a C program with the help of graphics mode to draw a box and it should be in moving state on the screen as output. ***(Jan. 2014)***

उत्तर–ग्राफिक्स मोड का प्रयोग करके एक Box बनाने तथा उसे screen पर move करते mode में दिखाने के लिये C program निम्न प्रकार होगा–

```
#include<graphics.h>
#include<dos.h>
#include<conio.h>
main()
{
 int i, j = 0, gd = DETECT, gm;
 initgraph(&gd,&gm,"C:\\TC\\BGI");
 settextstyle(DEFAULT_FONT,HORIZ_DIR,2);
 outtextxy(25,240,"Press any key to view the moving box");
 getch();
 setviewport(0,0,639,440,1);
 for( i = 0 ; i <= 420 ; i = i + 10, j++ )
 {
  rectangle(50+i,275,150+i,400);
  setcolor(j);
  delay(100);
  if( i == 420 )
   break;
  clearviewport();
 }
 getch();
 closegraph();
 return 0;
}
```

प्रश्न 17 – Polygon clipping के लिये एक C program लिखिए।

Write a C program that depicts the concept of polygon clipping. ***(Jan. 2014)***

उत्तर–Polygon Clipping के लिये C Program निम्नलिखित है–

```
#include<stdio.h>
#include<conio.h>
```

Scanned by CamScanner

```
#include<graphics.h>
#include<math.h>
void clip(float,float,float);
int i,j=0,n;
int rx1,rx2,ry1,ry2;
float x1[8],y1[8];
void main()
{
int gd=DETECT,gm;
int i,n;
float x[8],y[8],m;
clrscr();
initgraph(&gd,&gm,"C:\\TC\\BGI ");
printf("coordinates for rectangle : ");
scanf("%d%d%d%d",&rx1,&ry1,&rx2,&ry2);
printf("no. of sides for polygon : ");
scanf("%d",&n);
printf("coordinates : ");
for(i=0;i<n;i++)
{
scanf("%f%f",&x[i],&y[i]);
}
cleardevice();
outtextxy(10,10,"Before clipping");
outtextxy(10,470,"Press any key....");
rectangle(rx1,ry1,rx2,ry2);
for(i=0;i<n-1;i++)
line(x[i],y[i],x[i+1],y[i+1]);
line(x[i],y[i],x[0],y[0]);
getch();
cleardevice();
for(i=0;i<n-1;i++)
{
m=(y[i+1]-y[i])/(x[i+1]-x[i]);
clip(x[i],y[i],m);
clip(x[i+1],y[i+1],m);
}
m=(y[i]-y[0])/(x[i]-x[0]);
clip(x[i],y[i],m);
clip(x[0],y[0],m);
outtextxy(10,10,"After clipping");
```

Scanned by CamScanner

```
outtextxy(10,470,"Press any key....");
rectangle(rx1,ry1,rx2,ry2);
for(i=0;i<j-1;i++)
line(x1[i],y1[i],x1[i+1],y1[i+1]);
getch();
}
void clip(float e,float f,float m)
{
while(e<rx1,e>rx2,f<ry1,f>ry2)
{
if(e<rx1)
{
f+=m*(rx1-e);
e=rx1;
}
else if(e>rx2)
{
f+=m*(rx2-e);
e=rx2;
}
if(f<ry1)
{
e+=(ry1-f)/m;
f=ry1;
}
else if(f>ry2)
{
e+=(ry2-f)/m;
f=ry2;
}
}
x1[j]=e;
y1[j]=f;
j++;
}
```

❑

प्रश्न 18 – *"graphics.h" हैडर फाइल का वर्णन कीजिए। सी भाषा में एक प्रोग्राम लिखिए जिसमें line function का उपयोग करके rectangle बनाइए एवं उसे blue colour से fill कीजिए।*

Explain "graphics.h" header file. Write a C program to create rectangle using line function and fill it with blue colour. ***(June 2014, 15)***

उत्तर–graphics.h हैडर फाइल C भाषा में पहले से परिभाषित फाइल है। इस हैडर फाइल में C भाषा में ग्राफिक्स के प्रोग्राम बनाने के

Scanned by CamScanner

लिये प्रयोग किये जाने वाले लाइब्रेरी फंक्शनूस पहले से परिभाषित होते हैं जिन्हें Source प्रोग्राम में सीधे तौर पर प्रयोग करते हैं। graphics.h हेडर फाइल को main( ) प्रोग्राम से पहले include करते हैं। इस हेडर फाइल के प्रमुख फंक्शनस निम्नलिखित हैं–

getmaxx( ) फंक्शन–getmaxx( ) फंक्शन का प्रयोग Current Resolution में x co-ordinate की maximum value को get करने के लिये किया जाता है; जैसे–mx = getmaxx( );

यह x co-ordinate की maximum वैल्यू को mx वेरियेबल में return करेगा। x co-ordinate की maximum वैल्यू को निम्नलिखित statement के द्वारा प्रदर्शित करते हैं–

```
printf("%d", mx);
```

getmaxy( ) फंक्शन–getmaxy( ) फंक्शन का प्रयोग Current Resolution में y co-ordinate की maximum value को प्राप्त करने के लिये करते हैं; जैसे–

```
my = getmaxy( );
```

यह स्टेटमेंट my वेरियेबल में y co-ordinate की maximum value को return करके store करता है। y co-ordinate की maximum value को निम्नलिखित statement के द्वारा प्रदर्शित करते हैं–

```
printf("%d", my);
```

[नोट–**graphics.h** हेडर फाइल के विस्तृत function जानने के लिये **पृष्ठ 64** पर **प्रश्न-1** एवं **पृष्ठ 68** पर **प्रश्न-2** के उत्तर का अध्ययन करें।] ❑

[नोट–**program** के लिए **पृष्ठ 77** पर **प्रश्न-15** के उत्तर का अध्ययन करें।]

प्रश्न 19 – ***विभिन्न ग्राफिक्स फंक्शन्स का उपयोग करते हुए circle, arcs, polygon एवं rectangle निर्माण के लिए C प्रोग्राम लिखिए।***

Write a C program which is used to draw circle, arcs, polygon and rectangle using various graphics functions. ***(June 2014, 15)***

उत्तर–

```
#include<stdio.h>
#include<conio.h>
#include<graphics.h>
main()
{
  int gd = DETECT, gm;
  int pentagon[12] ={340,150, 320,110, 360,70, 400,110, 380,150, 340,150};
  initgraph(&gd, &gm,"C:\\TC\\BGI");
  setbkcolor(BLACK);
  setcolor(RED);
  rectangle(50,80,250,280);
  circle(150,175,40);
  arc(150,150,0,135,50);
  drawpoly(6,pentagon);
  getch();
  closegraph();
  return 0;
  getch( );
}
```

❑❑

Scanned by CamScanner

UNIT-V

प्रश्न 1 – *ROM-BIOS की व्याख्या कीजिए।*
Explain the ROM-BIOS.
(June 2005,11; Jan. 2009, 11)

अथवा (Or)

ROM-BIOS क्या होते हैं तथा इनकी उपयोगिता क्या है?
What are ROM-BIOS and what is the use of ROM-BIOS. ***(Jan. 2012; June 2013)***

उत्तर– रोम-बायोस विशेष फंक्शन का समूह होता है जो विभिन्न हार्डवेयर इकाइयों के बीच तालमेल स्थापित करने का कार्य करते हैं। जब भी हम कम्प्यूटर ऑन करते हैं तब ये फंक्शन कम्प्यूटर हार्डवेयर तथा अन्य पेरिफेरल डिवाइसेस के बीच कम्यूनिकेशन लिंक के रूप में कार्य करते हैं। ये फंक्शन सिस्टम के सभी आधारभूत कार्यों को सम्पन्न कराते हैं; जैसे– स्क्रीन पर डाटा डिस्प्ले कराना, हार्डडिस्क से डाटा रीड (Read) करना आदि। रोम-बायोस फंक्शनों का कोई नाम नहीं होता जिसके कारण इन्हें हाई-लेवल लैंग्वेजों में उपयोग किये जाने वाले फंक्शनों की तरह आसानी से कॉल नहीं किया जा सकता। इन्हें कॉल करने के लिए पहले इनके मेमोरी ऐड्रेसों को प्राप्त किया जाता है। ये ऐड्रेस लो मेमोरी एरिया में इंटरप्ट वेक्टर टेबल में संग्रहित होते हैं। शेष बायोस फंक्शन का एड्रेस प्राप्त करने के बाद कंट्रोल को इन ऐड्रेस पर पास करके फंक्शन को कॉल किया जाता है। यह सम्पूर्ण प्रक्रिया काफी जटिल है क्योंकि ये रोम-बायोस फंक्शन एड्रेस याद नहीं रखे जा सकते। जब रोम-बायोस फंक्शन कॉल किया जाता है तब इंटरप्ट के माध्यम से अस्थायी रूप से माइक्रोप्रोसेसर के वर्तमान कार्य को रोक दिया जाता है ताकि कंट्रोल को रोम-बायोस फंक्शन को पास किया जा सके। ❑

प्रश्न 2 – *'C' में int86() फंक्शन को उदाहरण सहित समझाइये।*
Explain int86() function in 'C' with an example. ***(Jan. 2015; June 2010, 11, 13, 14, 15)***

उत्तर– 'C' में int86() फंक्शन माइक्रोप्रोसेसर के लिए इंटरप्ट उत्पन्न करने के लिए उपयोग किया जाता है। int86() फंक्शन एक सॉफ्टवेयर इंटरप्ट उत्पन्न करता है। यहां int का अर्थ interrupt तथा 86 का तात्पर्य 8086 माइक्रोप्रोसेसर है। यह एक स्टैण्डर्ड लाइब्रेरी फंक्शन है जो <dos.h> हेडर फाइल में उपलब्ध होता है। इसके द्वारा स्क्रीन पर कर्सर की लोकेशन बदलना तथा मैमोरी साइज ज्ञात करना आदि कार्य भी किये जा सकते हैं। इसकी सहायता से दिये गये इंटरप्ट विशेष को ट्रेस करके बायोस सॉफ्टवेयर तथा हार्डवेयर के साथ कार्य करता है।

int86() में आरग्यूमेंट के रूप में इनवोक किये जा रहे रोम-बायोस का इंटरप्ट नम्बर तथा दो यूनियन वेरिएबलों के एड्रेस दिये जाते हैं जिनमें यूनियन वेरिएबल, फंक्शन को भेजे जाने वाले मान को दर्शाता है, जबकि दूसरा यूनियन वेरिएबल रोम-बायोस फंक्शन द्वारा return की जाने वाली वेल्यू को प्रदर्शित करता है। इसका प्रारूप निम्नानुसार होता है–

```
int86 (Interrup number, address of union1, address of union 2);
```

उदाहरण–

```
int86(16, & inregs, & outregs);
```

दिये गये उदाहरण में–
(i) 16 इंटरप्ट नम्बर है–
(ii) & inregs रोम-बायोस को भेजी जा रही रजिस्टर वेल्यू को एड्रेस है।
(ii) $ outregs रोम-बायोस द्वारा return की जा रही रजिस्टर वैल्यू का एड्रेस है। ❑

प्रश्न 3 – *'C' में रोम-बायोस को CPU रजिस्टर द्वारा आर्ग्यूमेंट पास करने का वर्णन कीजिए।*
Explain how to pass argument in ROM-BOIS using CPU register.
(June 2011; Jan. 2012, 13, 16)

उत्तर– माइक्रोप्रोसेसर की अपनी स्वयं की स्टोरोज लोकेशन्स सी०पी०यू० रजिस्टर कहलाती है, जिनका उपयोग प्रोसेसिंग के समय सी०पी०यू० को भेजे गए डाटा को अस्थायी रूप से संग्रहित करने के लिए किया जाता है। इन मेमोरी रजिस्टर्स की गति अत्यंत तीव्र होती है। ये मेमोरी रजिस्टर चार प्रकार के होते हैं–

1. जनरल परपस रजिस्टर्स (General Purpose Registers)
2. सैग्मेंट रजिस्टर्स (Segment Registers)

Scanned by CamScanner

3. ऑफसेट रजिस्टर्स (Offset Registers)
4. फ्लैग रजिस्टर (Flag Registers)

1. जनरल परपस रजिस्टर्स (General Purpose Registers)

इनके अंतगत निम्नलिखित चार तत्वों को अस्थायी रूप से डाटा संग्रहित करने तथा प्रोसेसिंग संबंधित अन्य कार्यों के लिए उपयोग किया जाता है–

AX–Accumulator
BX–Base
CX–Count
DX–Data

उपरोक्त चारों 16 बिट रजिस्टर होते हैं जो 8-8 बिट के दो भागों में बंटे होते हैं जिनमें पहला भाग हाई-ऑर्डर 8 बिट रजिस्टर कहलाता है जिसके अन्तर्गत क्रमशः AH, BH, CH तथा DH रजिस्टर होते हैं तथा दूसरा भाग लो-ऑर्डर 8 बिट रजिस्टर कहलाता है जिसके अन्तर्गत AL, BL, CL तथा DL रजिस्टर होते हैं।

2. सैग्मेंट रजिस्टर्स (Segment Registers)

सैगमेंट रजिस्टर 8086 माइक्रोप्रोसेसर की प्रत्येक लोकेशन को 16 बिट सैग्मेंट के रूप में प्रस्तुत करते हैं। सैग्मेंट रजिस्टर चार प्रकार के होते हैं–

(a) CS ये रजिस्टर पोर्ट सैग्मैंट की पहचान करते हैं।
(b) DS ये डाटा सैग्मेंट रजिस्टर होते हैं जो प्रोग्राम के अंतर्गत आने वाले डाटा भाग को संग्रहित करते हैं।
(c) ES ये अतिरिक्त सैग्मेंट (Extra Segment) रजिस्टर होते हैं।
(d) SS ये रजिस्टर स्टोर सैग्मेंट की पहचान करते हैं।

3. ऑफसेट रजिस्टर्स (Offset Registers)

इनका उपयोग सैग्मेंट रजिस्टर एवं मैमोरी में प्वॉइंटर एड्रेस जोड़ने के लिए किया जाता है। इनके अन्तर्गत इन्सट्रक्शन प्वॉइंटर (IP) स्टैक प्वॉइंटर (SP) सोर्स इन्डेक्स (SI) तथा डेस्टिनेशन इन्डेक्स (DI) आते हैं।

4. फ्लैग रजिस्टर्स (Flag Registers)

फ्लैग रजिस्टर्स विभिन्न प्रोसेसर की स्टेटस से सम्बन्धित फ्लैग को दर्शाते हैं, जो यह दर्शाता है कि कौन-सी प्रोसेस सफलतापूर्वक पूरी हुई है और कौन सी नहीं?

ये फ्लैग निम्न प्रकार के होते हैं–

(a) OF (Overflow Flag)
(b) DF (Driection Flag)
(c) IF(Interrupt Flag)
(d) TF (Track Flag)
(e) SF (Sigh Flag)
(f) ZF (Zero Flag)
(g) CF (Carry Flag)

रोम-बायोस सी०पी०यू० रजिस्टर के द्वारा आरग्यूमेंट सॉफ्टवेयर इंटरप्ट के माध्यम से पास किये जाते हैं। ❑

Scanned by CamScanner

प्रश्न 4 – *Int86 function* का उपयोग करके *C* में प्रोग्राम लिखिए जो *Cursor* का आकार बदले।

Write C program using Int86( ) function which can change the shape of cursor.

(Jan. 2011, 15)

उत्तर–

Int86() का उपयोग Cursor का आकार बदलने के लिए प्रोग्राम–

```
#include<stdio.h>
#include<conio.h>
#include<dos.h>
viod cursor_off (void)
{
    Union REGS reg;
}
    /* set sursor size */
}
        reg.h.ah=0x01;
        reg.h.d=0x10;
        reg.h.ch=0x11;
        int 86(0×10, &reg, &reg);
    }
viod set_cursor_type(int cursortype)
    {
        Union REGS reg;

        /* set cursur size */
        reg.h.ah=0×01;
        reg.x.cx=i cursortype;

        int86(0×10, &reg, &reg);
    {
        int get_cursor_type (void)
    {
        Union REGS reg;
        /* get cursor size (cx) */
        reg.h.ah=0×03;
        int86(0×10, &reg, &reg);
        return(reg.x.cx);
    }
    void main()
        {
```

Scanned by CamScanner

```
    int ct;
    cursor_off();
    set_cursor_type(6);
    ct=get_cursor_type();
    printf("%d", ct);
}
```

प्रश्न 5 – Intdos() Function को समझाइये। Intdos() Function के द्वारा किस तरह directory create एवं delete करते हैं?

Explain Intdos( ) function. How can we **create** and **delete** directory through Intdos( ) function.

(Jan. 2010, 11, 15; June 2011, 12, 13, 14, 15)

उत्तर–Intdos() फंक्शन का उपयोग उन सिस्टम कॉल्स (System Calls) को चलाने के लिए किया जाता है जो DX या AL के अतिरिक्त अन्य रजिस्टर्स के द्वारा आरग्यूमेंट लेते हैं या जो Carry flag के माध्यम से एरर दर्शाते हैं। वे सेवाएँ जो केवल DX तथा AL रजिस्टर्स का उपयोग करती हैं तथा Carry flag का उपयोग नहीं करतीं bdos() फंक्शन के द्वारा चलायी जा सकती हैं।

Indtos() फंक्शन पहले रजिस्टर्स का मान inregs पर सैट करता है, इसके बाद डॉस सिस्टम कॉल को चलाता है। ये सिस्टम कॉल interrupt 2/h कहलाते हैं। सिस्टम कॉल के बाद रजिस्टर्स का मान outregs में सैट हो जाता है, इस अवस्था में सिस्टम के Carry flag का status, outregs की cflag फील्ड में संग्रहित हो जाता है। अगर flag का मान शून्य हो तो इसका अर्थ है कि प्रोग्राम में कोई सिस्टम एरर नहीं है, जबकि अगर flag में एक बार non-zero मान आए तो इसका अर्थ है कि प्रोग्राम में DOS error है।

अगर हमें बहुत बड़ा मेमोरी एड्रेस डॉस में पास करना है तो intdos() के स्थान पर intdosx() फंक्शन का उपयोग किया जाता है।

intdos() फंक्शन के द्वारा सिस्टम कॉल्स के माध्यम से directroy create तथा delete की जाती है।

[नोट–**Directory Create** एवं **Delete** के **Program** के लिए **पृष्ठ 91** पर **प्रश्न-13 (ii)** एवं **(iii)** के उत्तर का अध्ययन करें।]

प्रश्न 6 – संक्षिप्त टिप्पणी लिखिए–

Write short notes on :

(i) # and ## Preprocessor Operator **(Jan. 2011)**

(ii) DMA Functions **(June 2012, 14; Jan. 2013,13)**

उत्तर– **(i) # and # # Preprocessor Operator–**

 # Preprocessor Operator–इसे stringizing operator कहते हैं। यदि Macro की definition में Macro Expansion में string के Inside format Argument को रखा जाता है, तो ये actual argument से Replace नहीं किये जाते हैं। निम्नलिखित उदाहरण में इस ऑपरेटर का प्रयोग किया गया है–

```
#include<stdio.h>
#include<conio.h>
#define DISP(var, format) printf(#var"=%" #format "\n", var);
void main( )
{
int x=5;
  float y=105;
  char z='$';
  DISP(x,d);
```

```
    DISP(y, 0.2f);
    DISP(z, c);
    DISP(x*y, 0.2f);
    getch( );
}
```

Preprocessor Operator—इसे Token Passing Operator भी कहते हैं। Token Passing Operator ## का प्रयोग Macro definition में दो Tokens को एक Single Token Concatenate करने के लिये करते हैं। निम्नलिखित उदाहरण में इस ऑपरेटर का प्रयोग किया है—

```
#include<stdio.h>
#include<conio.h>
#define JOINT(a,b) a # # b
#define MARKS(subject) marks_## subject
void main( )
{
  int i=0, j=20;
int marks_phy=80, marks_Eng=85;
printf("%d %d", JOINT(i, 3), JOINT (i, 4);
printf("%d %d", MARKS(p.hy), MARKS(Eng));
getch( );
}
```

(ii) DMA function—DMA function वह function होता है जिसके द्वारा मैमोरी के Size का पता किया जाता है। ROM-BIOS में उपलब्ध फंक्शन्स में से एक फंक्शन Machine में Installed base Memory के size को रिटर्न करता है। यह Utility तब अधिक उपयोगी होती है जब आप यह नहीं जानते कि जिस Machine को आप use कर रहे हैं वह Machine कितनी मैमोरी का प्रयोग करती है। निम्नलिखित उदाहरण से यह Point स्पष्ट है— ***(Jan. 2011)***

```
#include<stdio.h>
main( )
{
  struct WORDREGS
  {
  unsigned int ax, bx, cx, dx, di, si, cflag, flags;
  }
};
struct BYTEREGS
 {
 unsigned char a1, ab, b1, bh, c1, ch, d1, dh;
 };
union REGS
 {
   struct WORDREGS x;
   struct BYTEREGS h;
 };
union REGS inregs, intregs;
int memsize;
```

Scanned by CamScanner

```
int 86(18, & inregs, & outregs);
memsize=(outregs.x.ax);
printf("total memory %d", memsize);
getch( );
}
```

प्रश्न 7 – आप C प्रोग्राम में माउस तथा की-बोर्ड को कैसे कंट्रोल करेंगे? उदाहरण सहित समझाइये।
How do you control a Mouse and Keyboard in C program? Explain with example.

(Jan. 2011)

उत्तर–

```
int intmouse();
void show_cursor();
void hide_cursor();
void row cal():
int initmouse()
{
union REGS regs1, regs2;
regs1.x.ax=0;
int 86(0×33, & regs1, & regs2);
return (regs2.x.ax);
}
void show_cursor()
{
union REGS regs1, regs2;
regs1.x.ax=1;
int 86(0×33, & regs1, & regs2);
}
void hide_cursor()
{
   union REGS regs1, regs2;
   regs1.x.ax=2;
   int 86(0×33, & regs1, & regs2);
   };
   void row cal (int * row, int * cal)
   {
   int k;
   union REGS regs1, regs2;
   regs1.x.ax=3;
   int 86(0×33, & regs1, & regs2);
   * row=regs2.x.dx;
   *cal=regs2.x.cx;
 }
```

Scanned by CamScanner

प्रश्न 8 – *ROM BIOS routine क्या है?*
What is ROM BIOS routine? **(Jan. 2015; June 2011, 13, 14, 15)**

उत्तर–'कम्प्यूटर' हार्डवेयर तथा सॉफ्टवेयर से मिलकर बना होता है। कम्प्यूटर पर कार्य करने के लिये विभिन्न प्रकार की Peripheral Devices जुड़ी होती हैं; जैसे–Keyboard, Monitor, Pointer इत्यादि। कम्प्यूटर में कार्य करने के लिये सर्वप्रथम ऑपरेटिंग सिस्टम को Install करते हैं। ऑपरेटिंग सिस्टम एक सॉफ्टवेयर है जो Input-Output Devices, Memory Management के कार्य को पूरा करता है। ऑपरेटिंग सिस्टम में विभिन्न प्रकार के Control फंक्शन्स होते हैं जो ऑपरेटिंग सिस्टम की Services को पूरा करते हैं। PC में Routines को Run करने के लिये विभिन्न फंक्शन्स होते हैं। ये Routines तब क्रियान्वित होते हैं जब ऑपरेटिंग सिस्टम के Prompt पर कमाण्ड दी जाती है; जैसे–copy, type, time इत्यादि। इन built-in-routines का प्रयोग करते हैं और आप इन्हें ऑपरेटिंग सिस्टम Prompt पर प्रयोग करते हैं। इन Routines को ROM BIOS Routines कहते हैं। C compiler के पास इन Routines में एक्सेस करने की Authority होती है और इन Routines का प्रयोग करता है। यह आपको पहले से बने हुए प्रोग्रामों में एक्सेस कराता है। BIOS Routines आपके प्रोग्राम को तेजी से क्रियान्वित कराते हैं। जब BIOS Routines को Call किया जाता है तो Input Parameters Stacks में Place नहीं किये जाते हैं बल्कि ये Hardware Device में Place किये जाते हैं, इन्हें Registers कहते हैं। माइक्रोप्रोसेसर में BIOS Routines में Arguments को Pass करने के लिये चार Registers प्रयोग किये जाते हैं।

इन Registers को AX, BX, CX तथा DX कहते हैं। Register में Store किया गया वेरियेबल Stack में Store करने की तुलना में शीघ्रता से एक्सेस किया जाता है। ऐसा इसलिये होता है क्योंकि BIOS Routines आपके प्रोग्राम को अत्यधिक तेज क्रियान्वित करता है।

प्रश्न 9 – *Keyboard तथा Mouse को कौन-सा function support करता है?*
Which function supports keyboard and mouse? **(Jan. 2011)**

उत्तर–Mouse तथा Keyboard को अपने प्रोग्राम में कॉल करने के लिये int86() फंक्शन का प्रयोग किया जाता है। यह फंक्शन das.h हैडर फाइल में परिभाषित है। इस फंक्शन का प्रारूप निम्नलिखित रूप में होता है–

```
int86(int_no, &regs1, &regs2);
```

यहां पर int_no एक इंटीजर नम्बर हैं, जिसे Interrupt number कहते हैं। int86() फंक्शन की परफॉरमेन्स interrupt number पर निर्भर करती है। हम जिस Peripheral device को प्रोग्राम में कॉल करना चाहते हैं, उसका Interrupt number इस फंक्शन में कॉल करते हैं। जैसे Mouse को इनिशियलाइज करने के लिये 33 Interrupt No. को int86() फंक्शन में कॉल किया जाता है इसी तरह Keyboard का Interrupt Number 22 है।

उदाहरण 1 : निम्नलिखित फंक्शन में int86() फंक्शन में Mouse को Initiate करने के लिये इसके Interrupt number को Input argument के रूप में Pass किया गया है–

```
int initmouse()
    {
    union REGS regs1, regs2;
    regs1.x.ax = 0;
     int 86(OX 33, &regs1, & regs2);          // function to call initiate mouse
    return(regs2. x. ax);
    }
```

उदाहरण 2 : निम्नलिखित फंक्शन में int86() फंक्शन में Keyboard को Initiate करने के लिये इसके Interrupt number को Input argument के रूप में Pass किया गया है।

```
int init keyboard()
    {
        union REGS regs1, regs2;
        regs1.x.ax = 0;
        int 86(OX22, &regs1, &regs2);          // Function to initiate Keyboard
        return(regs2.x.ax);
    }
```

Scanned by CamScanner

प्रश्न 10 – *Screen को साफ तथा मेमोरी की size का पता int86() function का उपयोग करके कैसे किया जाता है?*
How to clear the screen using int86() function and how to find the memory size by using int86()? ***(Jan. 2012, 16)***

उत्तर–निम्नलिखित प्रोग्राम के द्वारा int86() फंक्शन का प्रयोग करके screen को clear किया जाता है।

```
#include<stdio.h>
#include<conio.h>
#include<dos.h>
void main()
{
union REGS regs1, regs2;
regs1.h.ah = 6;
regs1.h.al = 0;
regs1.h.ch = 0;
regs1.h.cl =0;
regs1.h.dh = 24;
regs1.h.dl = 79;
regs1.h.bh = 7;
int 86 (0×10, & regs1, & regs2);
getch();
}
```

int86() फंक्शन का प्रयोग करके ROM BIOS की सहायता से कम्प्यूटर की मैमोरी का साईज प्राप्त कर सकते हैं। निम्नलिखित प्रोग्राम के द्वारा int86() फंक्शन का प्रयोग करके मैमोरी का size प्राप्त कर सकते हैं।

```
#include<stdio.h>
#include<conio.h>
#include<dos.h>
int main()
{
struct WORDREGS
{
unsigned int ax, bx, cx, dx, si, di, a flag, b flag;
};
struct BYTEREGS
{
unsigned char al, ah, bl, bh, cl, ch, dl, dh;
};
union REGS
{
struct WORDREGS x;
struct BYTEREGS h;
};
union REGS regs1, regs2;
int memorysize;
int 86 (18, & regs1, & regs2);
memorysize = regs2.x.ax;
printf("memory size of pc %d", memorysize);
getch();
}
```

❑

Scanned by CamScanner

प्रश्न 11 – Intdos() का उपयोग करते हुये file के साथ निम्नलिखित कार्यों को समझाइये :
Explain the following using intdos() : **(Jan. 2010, 11, 12; June 11)**
Renaming file, deleting file, creating file

उत्तर–[नोट–इसके उत्तर के लिए **पृष्ठ 85** पर **प्रश्न–5** के उत्तर का अध्ययन करें।]

Renaming File–Dos में file का नाम बदलने के लिये rename कमाण्ड का प्रयोग dos prompt पर करते हैं। यदि हमें service No. पता हो तो intdos() फंक्शन के द्वारा file का नाम निम्नलिखित प्रोग्राम के माध्यम से rename कर सकते हैं–

```
#include<stdio.h>
#include<conio.h>
    void main()
    {
    union REGS i, 0;
    char filename[90];
    puts("enter file name");
    gets(file name);
    i.h.ah = 0 × 42           /*service number */
    i.x.dx = file name        /*offset address of filename */
    intdos(&i, &0);
    if (0.x.c flag = = 0)
    printf("% s file successfully renamed", filename);
    else
    printf("% s file could not be renamed", filename);
    getch();
}
```

प्रश्न 12 – PC में इन्स्टॉल की गयी मेमोरी के आकार (साइज) को हम किस प्रकार ज्ञात कर सकते हैं, समझाइये।
Explain how can we find the size of memory installed in a PC. **(June 2012; Jan. 2016)**

उत्तर–ROM-BIOS में उपलब्ध फंक्शनस् में से एक फंक्शन Machine में Installed base memory के Size को Return करता है। यह utility तब अधिक उपयोगी होती है जब आप यह नहीं जानते हैं कि जिस Machine को आप use कर रहे हैं वह Machine कितनी memory का प्रयोग करती है? यह उस प्रोग्रामों के लिये उपयोगी हो सकती है जो अलग-अलग Computers पर मैमोरी की अधिक मात्रा का प्रयोग करते है तथा भिन्न-भिन्न Memory size के Computers पर run होते है। ROM-BIOS फंक्शन को क्रियान्वित करने के लिए int86() फंक्शन निम्नलिखित प्रोग्राम के अनुसार arguments को pass करके Memory size को Kilobytes में find करते हैं–

```
/* finding memory size */
#include<stdio.h>
main( )
{
struct WORDREGS
{
unsigned int ax, bx, cx, dx, si, di, cflag, flags;
};
struct BYTEREGS
{
unsigned char al, ab, bl, bh, cl, ch, dl, dh;
};
union REGS
{
```

```
struct WORDREGS x;
struct BYTEREGS h;
};
union REGS inregs, intregs;
int memsize;
int86(18, & inregs, & outregs);
memsize=outregs.x.ax;
printf("/n total memory=%d", memsize);
getch( );
}
```

First structure WORDREGS में सभी registers होते हैं। Second structure BYTEREGS में Ax, Bx, Cx तथा Dx registers हैं। ये दोनों registers REGS union में डिफाइन्ड हैं। inregs तथा outregs REGS union type के variables है। इन variables में access करने के लिये हम दो बार डॉट ऑपरेटर (.) का प्रयोग करते हैं, जैसे–

```
outregs.x.ax
inregs.h.cl
```

WORDREGS तथा BYTEREGS स्ट्रक्चर्स और Regs union dos.h है जो फाइल में पहले से ही परिभाषित हैं, इस तरह यदि हम इस हैडर फाइल को प्रोग्राम में include करते हैं तो program का प्रारूप निम्नलिखित रूप में होता है–

```
# include<stdio.h>
#include<dos.h>
void main( )
{
union REGS inregs, outregs;
int memsize;
int86(18, &inregs, &outregs);
memsize = outregs.x.ax;
printf("total memory=%d", memsize);
getch( );
}
```

❑

प्रश्न 13 – निम्नलिखित पर संक्षिप्त टिप्पणियां लिखिए :

Write short notes on the following :

(i) DOS Routines *(June 2012, 13, 15)*

(ii) Creating Directory *(June 2012, 15; Jan. 2013)*

(iii) Delete Directory. *(June 2012; Jan. 2013)*

उत्तर–(i) DOS Routines– ROM-BIOS की विभिन्न Services होती हैं। ये Services.one interrupt के अंतर्गत आपस में सम्बन्धित होती हैं; जैसे–ROM-BIOS के फंक्शनस VDU के interrupt 16 के अंतर्गत आपस में एक-दूसरे से Interact करते हैं। Keyboard

Scanned by CamScanner

की Services interrupt 22 के अंतर्गत आती हैं। निम्नलिखित Figure में इनको दर्शाया गया है–

Interrupt Numbers		Purpose
Decimal	**Hexadecimal**	
Peripheral Device Services		
16	10	Video display services
19	13	Diskette services
20	14	Communication services
21	15	Cassette tape services
22	16	Keyboard services
23	17	Printer services
Equipment Status Services		
17	11	Equipment list services
18	12	Memory size service
Time/Date Services		
26	1A	Time and date services
Special Services		
5	5	Print screen services
24	18	Activate Rom BASIC
25	19	Activate boot step loader routine

Fig : Rom-BIOS/DOS Services

अधिकतर DOS Services interrupt number 0×21 (decimal 33) के अंतर्गत आती है। intdos( ) फंक्शन DOS Services को interrupt numbers 0×21 के अंतर्गत exist करता है। intdos( ) फंक्शन भी int86( ) फंक्शन की तरह ही dos.h हैडर फाइल में डिफाइन है। intdos( ) फंक्शन का प्रयोग DOS Routines में किया जाता है।

(ii) Creating Directory—हम अक्सर MD कमाण्ड का प्रयोग DOS prompt पर करते हैं। यदि हमें service number पता चल जाये तो MD कमाण्ड के स्थान इस service के माध्यम से डायरेक्टरी को क्रिएट कर सकते हैं। जैसा कि निम्नलिखित प्रोग्राम के माध्यम से डायरेक्टरी को क्रिएट किया है–

```
#include<stdio.h>
#include<conio.h>
void main( )
{
union REGS i, o;
struct SREGS s;
char far(dirname{120});
puts("\n enter directory name");
gets(dirname);
i.h.ah=0 × 39;      /*service no*/
s.ds=FP_SEG (dirname);
i.x.dx=FP_OFF (dirname);
intdosx(&i, &o,&s);
if(0.x.cflag==0)
printf("Directory successfully created");
```

Scanned by CamScanner

```
else
printf("could not create directory");
getch( );
}
```

(iii) Delete Directory—डायरेक्टरी को डिलीट करने के लिये DOS prompt पर RD कमाण्ड का प्रयोग करते हैं। Service No. के माध्यम से डायरेक्टरी को delete करने के लिये निम्नलिखित प्रोग्राम बनाते हैं–

```
#include<stdio.h>
#include<dos.h>
void main( )
{
union REGS i,o;
struct SREGS s;
char far(dirname[120]);
puts("\n enter directory name");
  gets(dirname);
  i.h.ah=0×40;          /*service no */
  s.ds=FP_SEG(dirname);
  i.x.dx=FP_OFF(dirname);
  intdosx(&i, &o, &s);
  if(0.x.cflag==0)
  printf("Directory successfully deleted");
  else
  printf("could not delete directory");
  getch( );
  }
```

❑

प्रश्न 14 – IVT पर संक्षिप्त टिप्पणी लिखिए:

Write short notes on IVT. ***(Jan. 2013, 15; June 2014)***

उत्तर–IVT : IVT का पूर्ण स्वरूप Integrating Vision Toolkit है। यह एक बहुत तेज तथा पॉवरफुल कम्प्यूटर विजन Library होती है। यह स्वयं का एक multi-platform GUI टूलकिट प्रदान करती है। यह लाइब्रेरी एक free सॉफ्टवेयर के रूप में उपलब्ध है। यह सभी ऑपरेटिंग सिस्टम्स (Windows Mac तथा Linux) के साथ कार्य करने में सक्षम है।

IVT को Karlsruhe Institute of Technology (KIT) में डेवलप किया गया था। IVT का प्रथम संस्करण 22 दिसम्बर 2005 को रिलीज किया गया था।

IVT के फीचर्स निम्नलिखित हैं–

1. Camera model (Monocular तथा Stereo)
2. Camera Calibration (using open (V.I.O.)
3. Filters
4. Color segmentation (HSV Color Space)
5. Hough
6. Undistortion
7. Rectification
8. Linear Least Squares
9. Data Structures
10. Particle filtering framework.

Scanned by CamScanner

प्रश्न 15 – *Dos Command 'dir' को बनाने के लिये C program लिखिये।*

Write a C program to create dos command 'dir'. ***(Jan. 2014)***

उत्तर–DOS Command 'dir' को C program द्वारा बनाने के लिये निम्न steps को follow करते हैं–

Step 1 : सबसे पहले निम्न Code को लिखते हैं :

```
#include<stdio.h>
#include<dos.h>
int main(int count,char *argv[]){
struct find_t q ;
  int a;
  if(count==1)
    argv[1]="*.*";
  a = _dos_findfirst(argv[1],1,&q);
if(a==0){
    while (!a){
      printf(" %s\n", q.name);
      a = _dos_findnext(&q);
    }
  }
  else{
    printf("File not found");
  }
  return 0;
}
```

Step 2 : अब इस प्रोग्राम को list.c के नाम से save करते हैं (कोई दूसरे नाम से भी save किया जा सकता है)।

Step 3 : File को Compile और execute करते हैं।

Step 4 : अब C से बाहर आ कर, Windows operating system के desktop पर My Computer पर Right Click करते हैं और प्रदर्शित सूचि से properties को select करते हैं।

Step 5 : अब properties window के Advanced tab को select करते हैं तथा उसमें Environment variables को select करते हैं।

Step 6 : अब प्रदर्शित Window में user variable के नीचे New button पर click करते हैं।

Step 7 : अब प्रदर्शित Window को निम्न प्रकार fill करते हैं।

Variable name: list
Variable value: c:\tc\bin\list.c (वह path जहां हमारा प्रोग्राम save है।)

Step 8 : अब Command prompt को खोलकर list type करते हैं तथा enter press करें।

इस प्रकार उपर्युक्त steps को follow करके Dos 'dir' command का C program लिखते है। ❑

प्रश्न 16 – *C language में virus create करने का program लिखिये।*

Write a program in C language to create virus. ***(Jan. 2014)***

उत्तर–C language में एक simple virus program computer को shutdown करने के लिये निम्न प्रकार बनाते हैं–

Step 1 : सर्वप्रथम नीचे दिये गये प्रोग्राम को C में लिखते है।

Scanned by CamScanner

```
#include<stdio.h>
#include<dos.h>
int mai (void){
  system("shutdown -s");
  return 0;
}
```

Step 2 : अब इस File को close.C के नाम से save करते हैं।

Step 3 : अब प्रोग्राम को केवल compile करते है, Run नहीं करते हैं।

Step 4 : अब C compiler को बंद कर देते हैं तथा windows operating system में उस directory को खोलते हैं जिसमें close.C को save किये हैं (आमतौर पर default directory c:\tc\bin होती है)।

Step 5 : अब इस प्रोग्राम की .exe file (close.exe) पर double click करने से कुछ समय पश्चात् windows operating system shut down हो जाता है। ❑

प्रश्न 17 – *DOS कमाण्ड 'Copy' को बनाने के लिए C प्रोग्राम लिखिए।*

Write a C program to create DOS command 'Copy'. ***(June 2014)***

उत्तर–

```
#include <stdio.h>
#include <string.h>
int main (int argc, char*argv[])
{
FILE *source,*target;
char ch;
/*If arguments are less then 3 then give an error*/
if (argc!=3)
{
printf("Command Error!! Insufficient argument given");
return;
}
source=fopen(argv[1], "r");
target = fopen (argv[2], "w");
if (source ==NULL || target == NULL)
{
printf ("Unable to open...ERROR in opening file!!n");
return;
}
while((ch = fgetc(source))!=EOF)
fputc(ch,target);        /*writing to the target file*/
printf("Copy is successful");
fclose(source);
fclose(target);
}
```

❑

Scanned by CamScanner