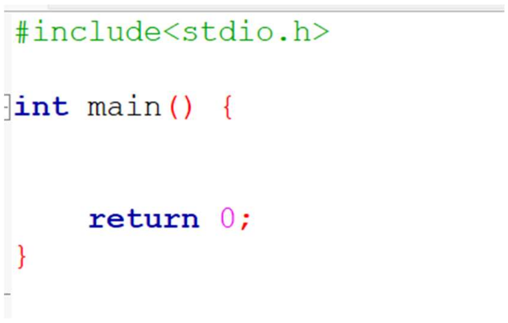
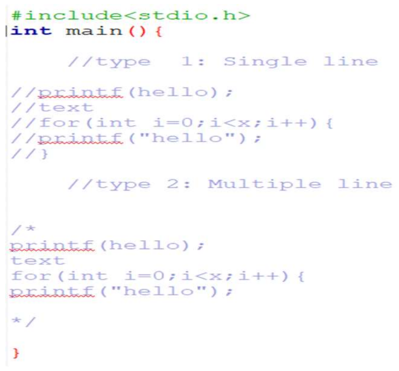
C

Boilerplate



```c
#include<stdio.h>

int main()  {


    return 0;
}
```

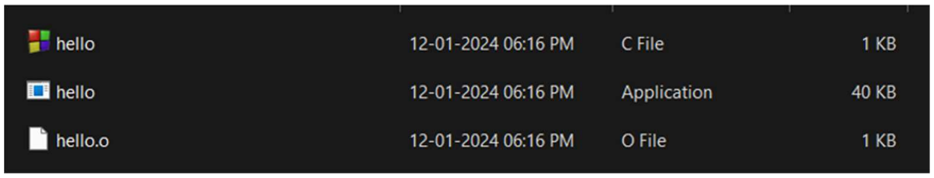
Comments



```c
#include<stdio.h>
int main(){

    //type  1: Single line

//printf(hello);
//text
//for(int i=0;i<x;i++){
//printf("hello");
//}

    //type 2: Multiple line


/*
printf(hello);
text
for(int i=0;i<x;i++){
printf("hello");

*/

}
```

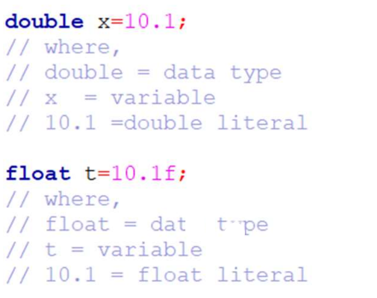
Whenever any file is compiled

- .c file
- .exe file(executable)
- .o file(object)

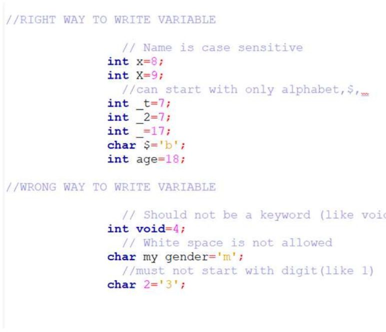


```
hello      12-01-2024 06:16 PM   C File          1 KB
hello      12-01-2024 06:16 PM   Application    40 KB
hello.o    12-01-2024 06:16 PM   O File          1 KB
```

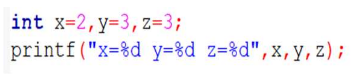
Data Types and variable



```c
double x=10.1;
// where,
// double = data type
// x  = variable
// 10.1 =double literal

float t=10.1f;
// where,
// float = dat  t pe
// t = variable
// 10.1 = float literal
```



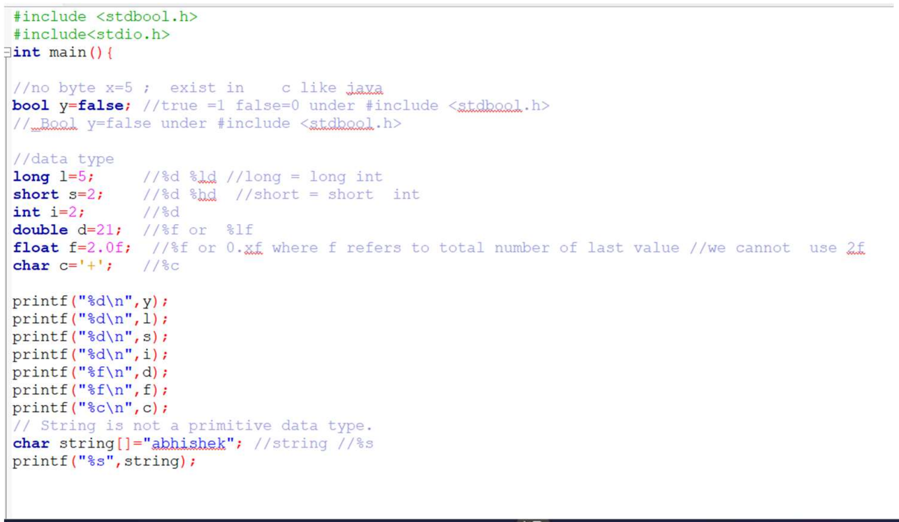
```c
//RIGHT WAY TO WRITE VARIABLE

              // Name is case sensitive
            int x=8;
            int X=9;
              //can start with only alphabet,$,_
            int _t=7;
            int _2=7;
            int _=17;
            char $='b';
            int age=18;

//WRONG WAY TO WRITE VARIABLE

              // Should not be a keyword (like void )
            int void=4;
              // White space is not allowed
            char my gender='m';
              //must not start with digit(like 1)
            char 2='3';
```



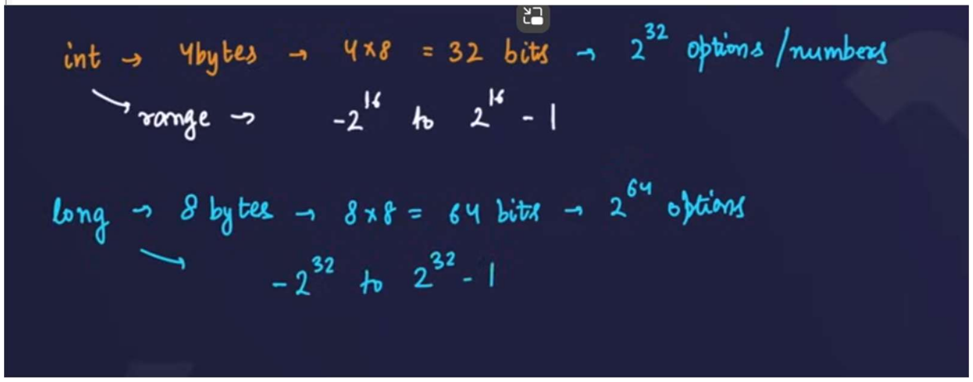
```c
int x=2,y=3,z=3;
printf("x=%d y=%d z=%d",x,y,z);
```

initializing multiple similar datatype variable

Primitive Data Type



```c
#include <stdbool.h>
#include<stdio.h>
int main(){

//no byte x=5 ;  exist in    c like java
bool y=false; //true =1 false=0 under #include <stdbool.h>
//_Bool y=false under #include <stdbool.h>

//data type
long l=5;      //%d %ld //long = long int
short s=2;     //%d %hd  //short = short  int
int i=2;       //%d
double d=21;   //%f or  %lf
float f=2.0f;   //%f or 0.xf where f refers to total number of last value //we cannot  use 2f
char c='+';    //%c

printf("%d\n",y);
printf("%d\n",l);
printf("%d\n",s);
printf("%d\n",i);
printf("%f\n",d);
printf("%f\n",f);
printf("%c\n",c);
// String is not a primitive data type.
char string[]="abhishek"; //string //%s
printf("%s",string);
```



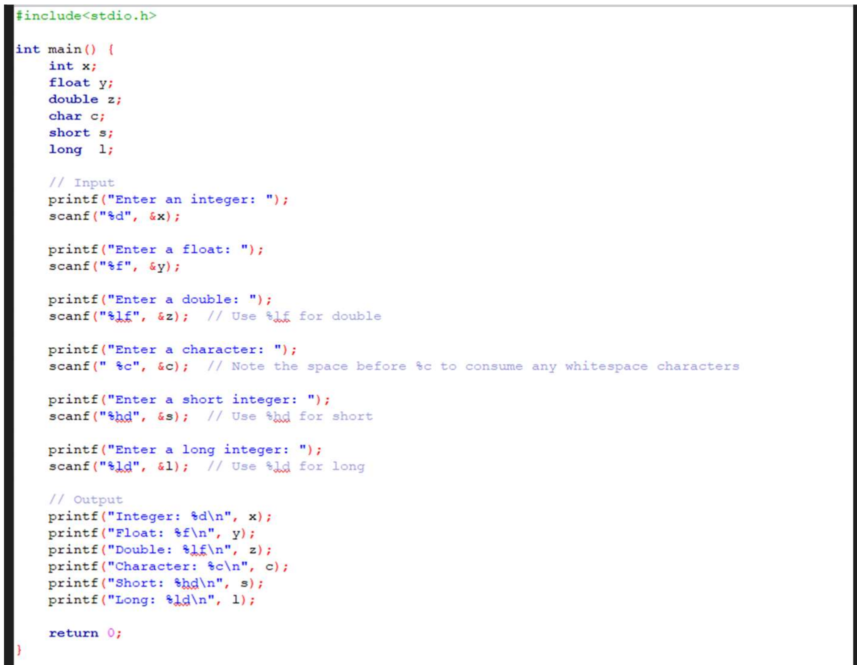
int → 4bytes → 4×8 = 32 bits → 2^{32} options/numbers

range → -2^{16} to $2^{16} - 1$

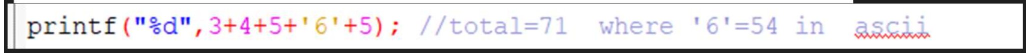
long → 8 bytes → 8×8 = 64 bits → 2^{64} options

-2^{32} to $2^{32} - 1$

Scanner Class

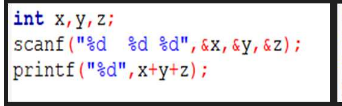


```c
#include<stdio.h>

int main() {
    int x;
    float y;
    double z;
    char c;
    short s;
    long  l;

    // Input
    printf("Enter an integer: ");
    scanf("%d", &x);

    printf("Enter a float: ");
    scanf("%f", &y);

    printf("Enter a double: ");
    scanf("%lf", &z);  // Use %lf for double

    printf("Enter a character: ");
    scanf(" %c", &c);  // Note the space before %c to consume any whitespace characters

    printf("Enter a short integer: ");
    scanf("%hd", &s);  // Use %hd for short

    printf("Enter a long integer: ");
    scanf("%ld", &l);  // Use %ld for long

    // Output
    printf("Integer: %d\n", x);
    printf("Float: %f\n", y);
    printf("Double: %lf\n", z);
    printf("Character: %c\n", c);
    printf("Short: %hd\n", s);
    printf("Long: %ld\n", l);

    return 0;
}
```



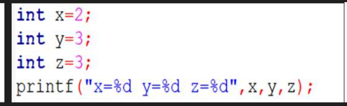
```c
printf("%d",3+4+5+'6'+5); //total=71  where '6'=54 in  ascii
```

Multiple scan in one time



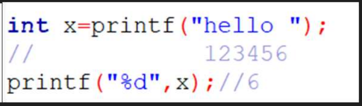
```c
int x,y,z;
scanf("%d  %d %d",&x,&y,&z);
printf("%d",x+y+z);
```

Multiple printing in one time



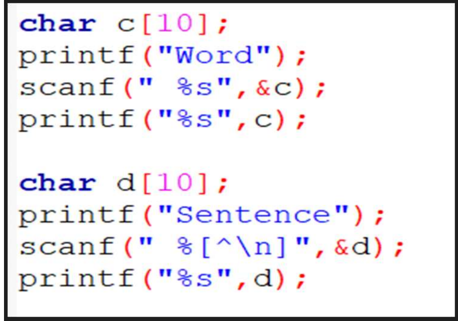
```c
int x=2;
int y=3;
int z=3;
printf("x=%d y=%d z=%d",x,y,z);
```

concept count in printf

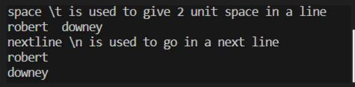


```c
int x=printf("hello ");
//           123456
printf("%d",x);//6
```

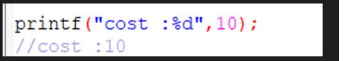
Scanning word vs sentence



```c
char c[10];
printf("Word");
scanf(" %s",&c);
printf("%s",c);

char d[10];
printf("Sentence");
scanf(" %[^\n]",&d);
printf("%s",d);
```

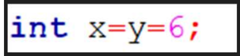


```
space \t is used to give 2 unit space in a line
robert  downey
nextline \n is used to go in a next line
robert
downey
```



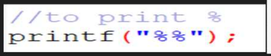
```c
printf("cost :%d",10);
//cost :10
```

Wrong format to initialize



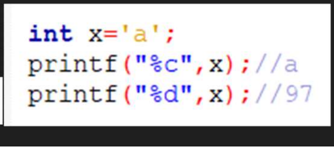
```c
int x=y=6;
```

to print % in sentence

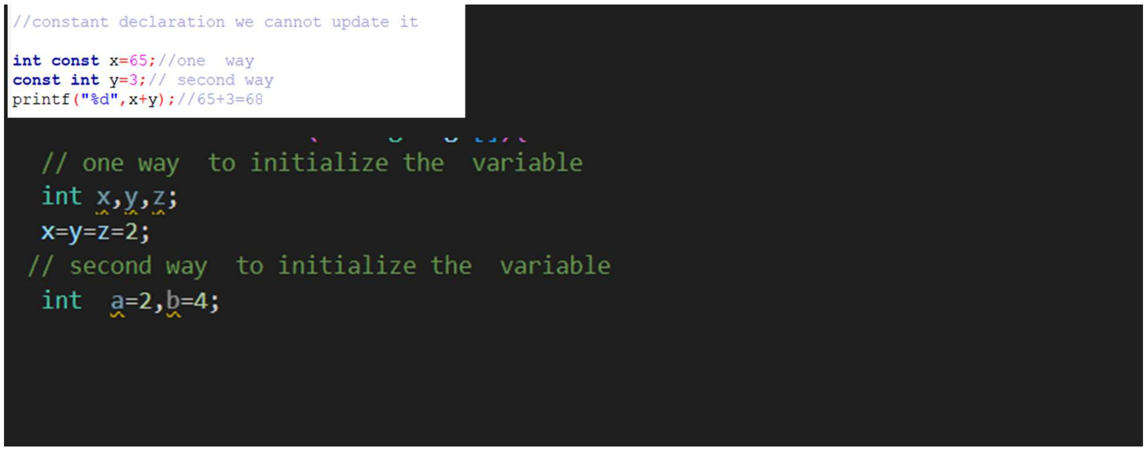


```c
//to print %
printf("%%");
```

ascii value



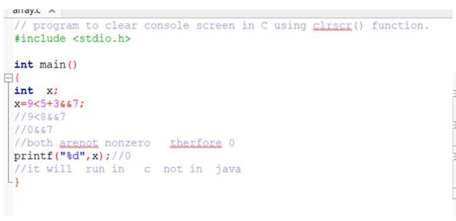
```c
int x='a';
printf("%c",x);//a
printf("%d",x);//97
```



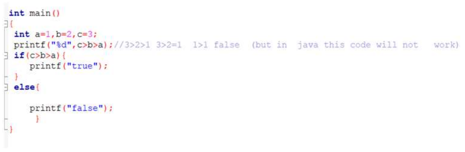
```
//constant declaration we cannot update it

int const x=65;//one  way
const int y=3;// second way
printf("%d",x+y);//65+3=68
```

```
  // one way  to initialize the  variable
  int x,y,z;
  x=y=z=2;
 // second way  to initialize the  variable
  int  a=2,b=4;
```

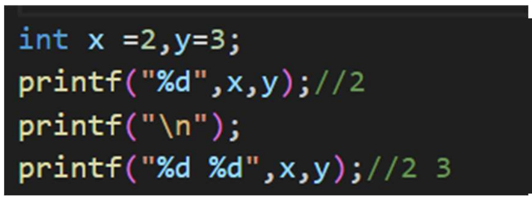
Operator Boolean not change into int in java but in c Boolean convert into int by default



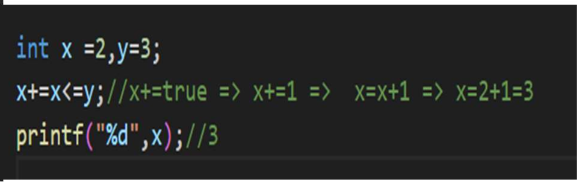
```
// program to clear console screen in C using clrscr() function.
#include <stdio.h>

int main()
{
int  x;
x=9<5+3&&7;
//9<8&&7
//0&&7
//both arenot nonzero   therfore 0
printf("%d",x);//0
//it will  run in   c  not in  java
}
```



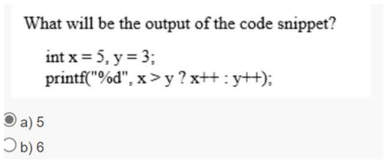
```
int main()
{
 int a=1,b=2,c=3;
 printf("%d",c>b>a);//3>2>1 3>2=1  1>1 false  (but in  java this code will not   work)
 if(c>b>a){
    printf("true");
 }
 else{

    printf("false");
     }
}
```



```
int x =2,y=3;
printf("%d",x,y);//2
printf("\n");
printf("%d %d",x,y);//2 3
```



```
int x =2,y=3;
x+=x<=y;//x+=true => x+=1 =>  x=x+1 => x=2+1=3
printf("%d",x);//3
```

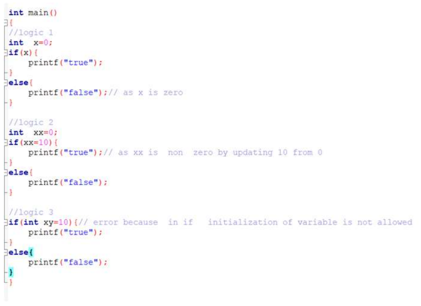


What will be the output of the code snippet?

```
int x = 5, y = 3;
printf("%d", x > y ? x++ : y++);
```

[✓ a) 5]

b) 6



```
int main()
{
//logic 1
int  x=0;
if(x){
    printf("true");
}
else{
    printf("false");// as x is zero
}

//logic 2
int  xx=0;
if(xx=10){
    printf("true");// as xx is  non  zero by updating 10 from 0
}
else{
    printf("false");
}

//logic 3
if(int xy=10){// error because  in if   initialization of variable is not allowed
    printf("true");
}
else{
    printf("false");
}
}
```

Data Type Conversion /widening/implicit conversion

short→int→ long→ float→double

Type Casting / Narrowing /Explicit Conversion

```
float x=5.3;
int z=x;
printf("%d",z);//5
```

```
printf("%f ",(float)3/4);//0.75 3 become float first
printf("%f ",(float)(3/4));//0.0 because(3/4) is int solved first
printf("%f ",(3.0/4));//0.75 because(3.0/4) 3.0 is float
printf("%f ",(3/4.0));//0.75 because(3/4.0) 4.0 is float
```

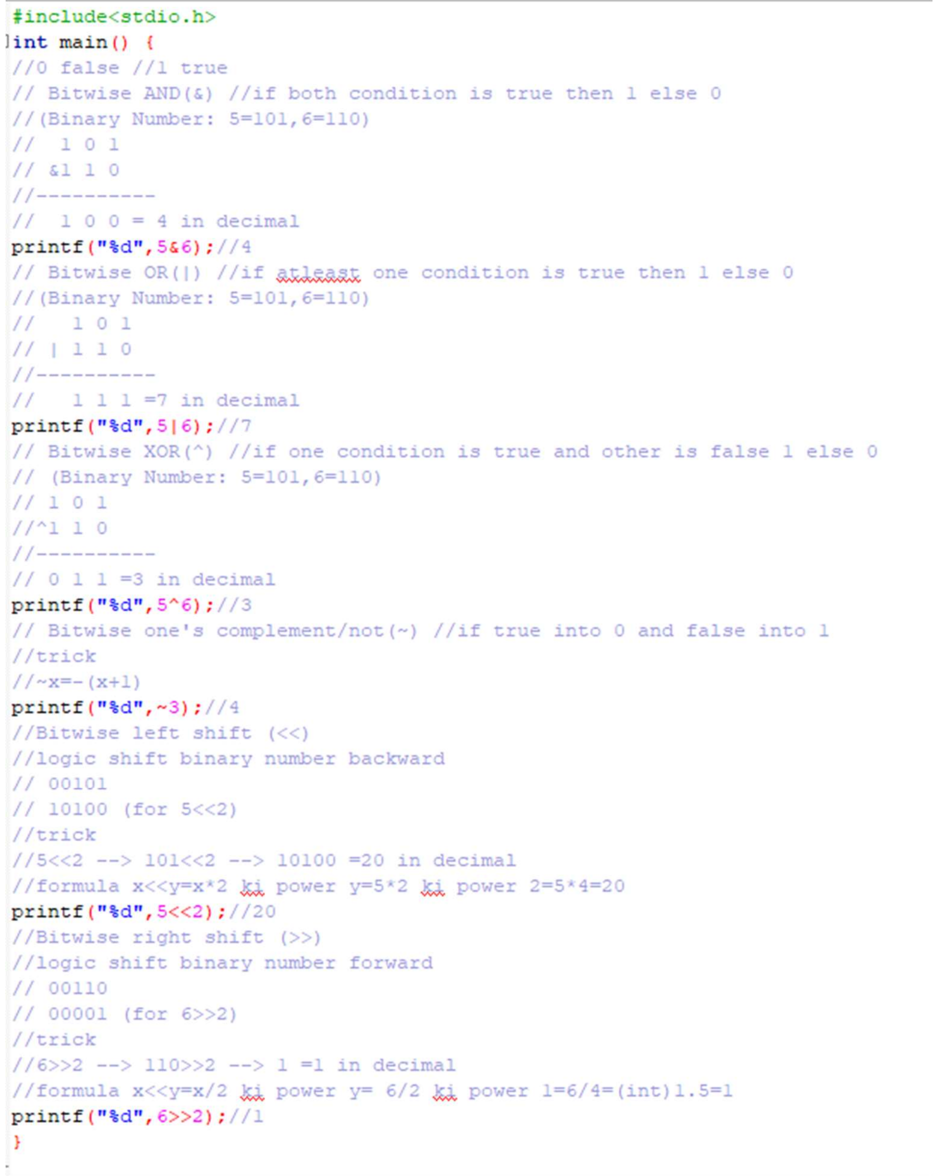
Operators

1. Arithmetic
   a. Binary (mathematical) operator: + - * / %
   b. Unary operator: ++x x++ --x x—
   c. Ternary operator: ?:

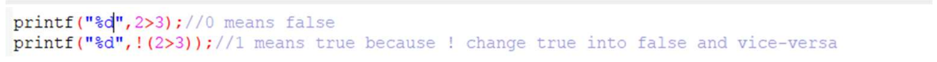
```
    int        x     =     5>2        ?   3  :  4;
//datatype variable = condition     ? true :false
```

2. Relational (== != > >= < <=)
3. Logical (&& || !)
4. Assignment (= += -= *= %= /=)
5. Bitwise (
   a. &(bitwise and)
   b. |(bitwise or)
   c. <<(shift left)
   d. >>(shift right)
   e. ~(one's complement)
   f. ^(bitwise exclusive or))

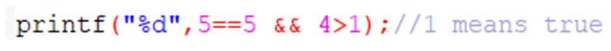
Bitwise Operator in detail



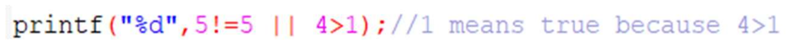
```
#include<stdio.h>
int main() {
//0 false //1 true
// Bitwise AND(&) //if both condition is true then 1 else 0
//(Binary Number: 5=101,6=110)
//  1 0 1
// &1 1 0
//----------
//  1 0 0 = 4 in decimal
printf("%d",5&6);//4
// Bitwise OR(|) //if atleast one condition is true then 1 else 0
//(Binary Number: 5=101,6=110)
//   1 0 1
// | 1 1 0
//----------
//   1 1 1 =7 in decimal
printf("%d",5|6);//7
// Bitwise XOR(^) //if one condition is true and other is false 1 else 0
// (Binary Number: 5=101,6=110)
// 1 0 1
//^1 1 0
//----------
// 0 1 1 =3 in decimal
printf("%d",5^6);//3
// Bitwise one's complement/not(~) //if true into 0 and false into 1
//trick
//~x=-(x+1)
printf("%d",~3);//4
//Bitwise left shift (<<)
//logic shift binary number backward
// 00101
// 10100 (for 5<<2)
//trick
//5<<2 --> 101<<2 --> 10100 =20 in decimal
//formula x<<y=x*2 ki power y=5*2 ki power 2=5*4=20
printf("%d",5<<2);//20
//Bitwise right shift (>>)
//logic shift binary number forward
// 00110
// 00001 (for 6>>2)
//trick
//6>>2 --> 110>>2 --> 1 =1 in decimal
//formula x<<y=x/2 ki power y= 6/2 ki power 1=6/4=(int)1.5=1
printf("%d",6>>2);//1
}
```



```
printf("%d",2>3);//0 means false
printf("%d",!(2>3));//1 means true because ! change true into false and vice-versa
```



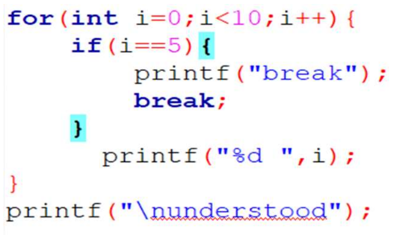
```
printf("%d",5==5 && 4>1);//1 means true
```



```
printf("%d",5!=5 || 4>1);//1 means true because 4>1
```

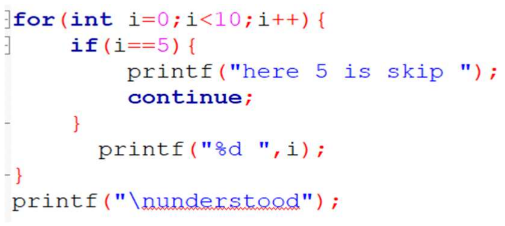
Break and Continue Statement

Break(to exit loop)



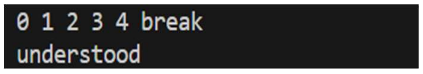
```c
for(int i=0;i<10;i++){
    if(i==5){
        printf("break");
        break;
    }
      printf("%d ",i);
}
printf("\nunderstood");
```

Continue(to skip specific condition iteration)



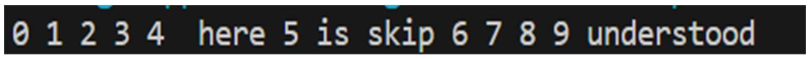
```c
for(int i=0;i<10;i++){
    if(i==5){
        printf("here 5 is skip ");
        continue;
    }
      printf("%d ",i);
}
printf("\nunderstood");
```

Output: break



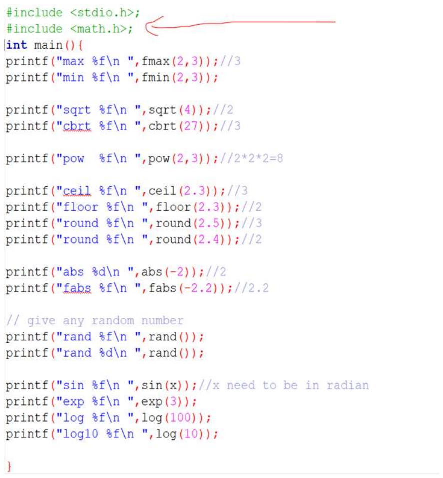
```
0 1 2 3 4 break
understood
```

Output: continue



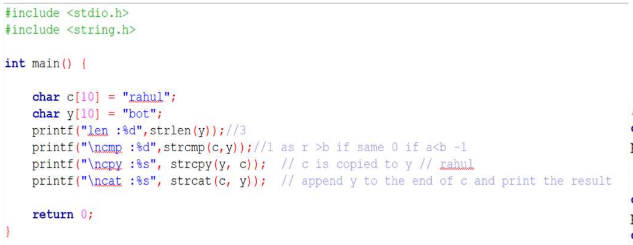
```
0 1 2 3 4  here 5 is skip 6 7 8 9 understood
```

Math function

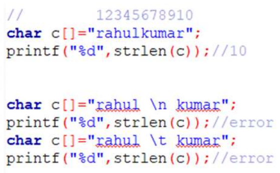


```c
#include <stdio.h>;
#include <math.h>;
int main(){
printf("max %f\n ",fmax(2,3));//3
printf("min %f\n ",fmin(2,3));

printf("sqrt %f\n ",sqrt(4));//2
printf("cbrt %f\n ",cbrt(27));//3

printf("pow  %f\n ",pow(2,3));//2*2*2=8

printf("ceil %f\n ",ceil(2.3));//3
printf("floor %f\n ",floor(2.3));//2
printf("round %f\n ",round(2.5));//3
printf("round %f\n ",round(2.4));//2

printf("abs %d\n ",abs(-2));//2
printf("fabs %f\n ",fabs(-2.2));//2.2

// give any random number
printf("rand %f\n ",rand());
printf("rand %d\n ",rand());

printf("sin %f\n ",sin(x));//x need to be in radian
printf("exp %f\n ",exp(3));
printf("log %f\n ",log(100));
printf("log10 %f\n ",log(10));

}
```

String Function



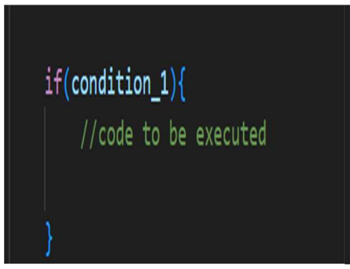
```c
#include <stdio.h>
#include <string.h>

int main() {

    char c[10] = "rahul";
    char y[10] = "bot";
    printf("len :%d",strlen(y));//3
    printf("\ncmp :%d",strcmp(c,y));//1 as r >b if same 0 if a<b -1
    printf("\ncpy :%s", strcpy(y, c));  // c is copied to y // rahul
    printf("\ncat :%s", strcat(c, y));  // append y to the end of c and print the result

    return 0;
}
```



```c
//          12345678910
char c[]="rahulkumar";
printf("%d",strlen(c));//10


char c[]="rahul \n kumar";
printf("%d",strlen(c));//error
char c[]="rahul \t kumar";
printf("%d",strlen(c));//error
```

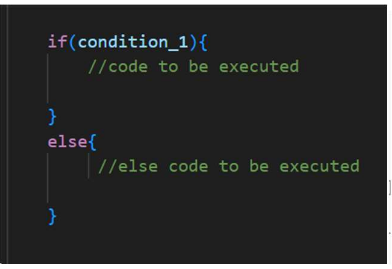
Conditional Branching/Selectional Control/Decision Making

If Statement



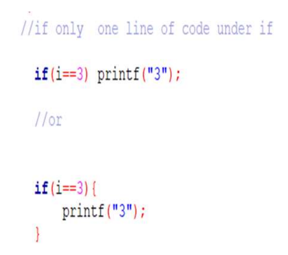
```
if(condition_1){
    //code to be executed

}
```

if else Statement



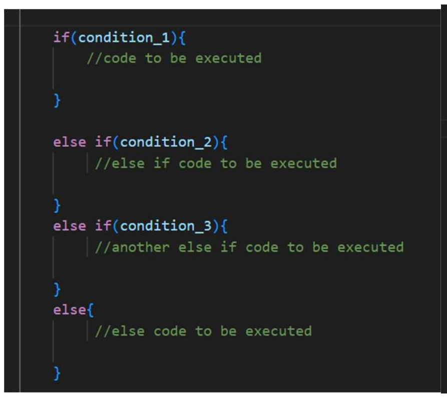
```
if(condition_1){
    //code to be executed

}
else{
    //else code to be executed

}
```

if only one statement



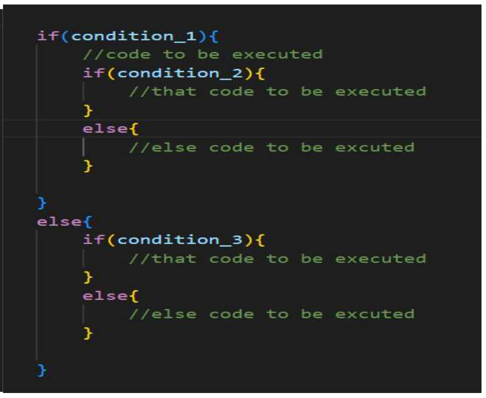
```
//if only  one line of code under if

  if(i==3) printf("3");

  //or


  if(i==3){
      printf("3");
  }
```

Else if Statement



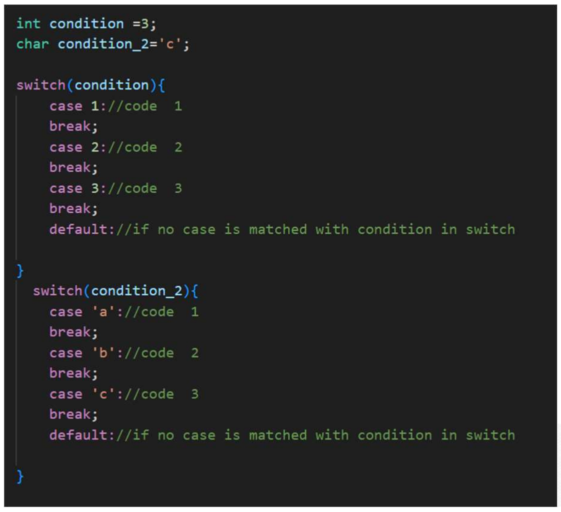
```
if(condition_1){
    //code to be executed

}

else if(condition_2){
    //else if code to be executed

}
else if(condition_3){
    //another else if code to be executed

}
else{
    //else code to be executed

}
```

Nested if Statement

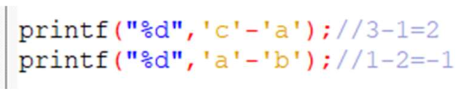


```
if(condition_1){
    //code to be executed
    if(condition_2){
        //that code to be executed
    }
    else{
        //else code to be excuted
    }

}
else{
    if(condition_3){
        //that code to be executed
    }
    else{
        //else code to be excuted
    }

}
```

Switch statement



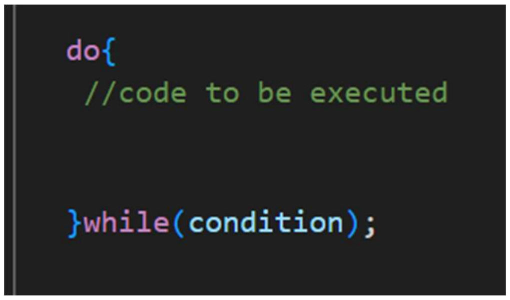
```
int condition =3;
char condition_2='c';

switch(condition){
    case 1://code  1
    break;
    case 2://code  2
    break;
    case 3://code  3
    break;
    default://if no case is matched with condition in switch

}
  switch(condition_2){
    case 'a'://code  1
    break;
    case 'b'://code  2
    break;
    case 'c'://code  3
    break;
    default://if no case is matched with condition in switch

}
```



```
printf("%d",'c'-'a');//3-1=2
printf("%d",'a'-'b');//1-2=-1
```

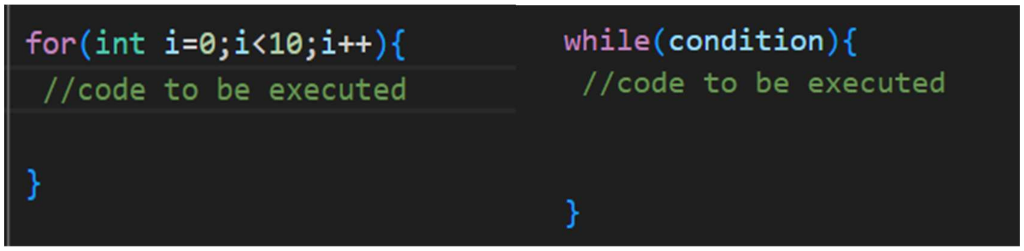
Loop Statement

1)Exit Controlled Loop/post tested loop (do while loop)



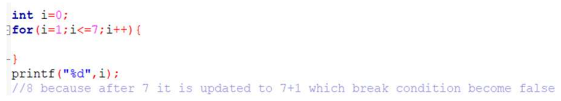
```
do{
 //code to be executed


}while(condition);
```

2)Entry Controlled Loop/pre tested loop (for loop, while loop)

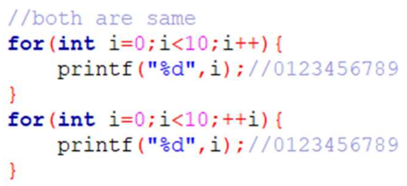


```
for(int i=0;i<10;i++){
 //code to be executed


}
```

```
while(condition){
 //code to be executed


}
```



```
int i=0;
for(i=1;i<=7;i++){

}
printf("%d",i);
//8 because after 7 it is updated to 7+1 which break condition become false
```



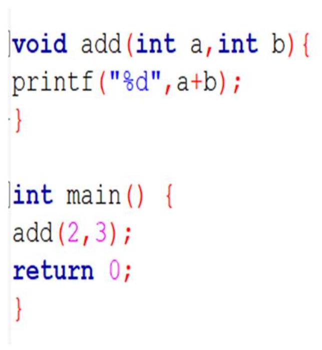
```
//both are same
for(int i=0;i<10;i++){
    printf("%d",i);//0123456789
}
for(int i=0;i<10;++i){
    printf("%d",i);//0123456789
}
```

Function

- function overloading (same name different parameter or same name but parameter datatype different) is not supported in c

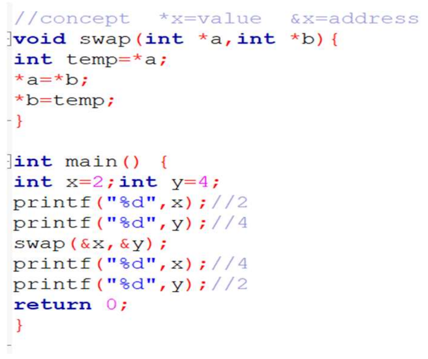
Here value is passed in function by call by value and call by reference.

Call by value



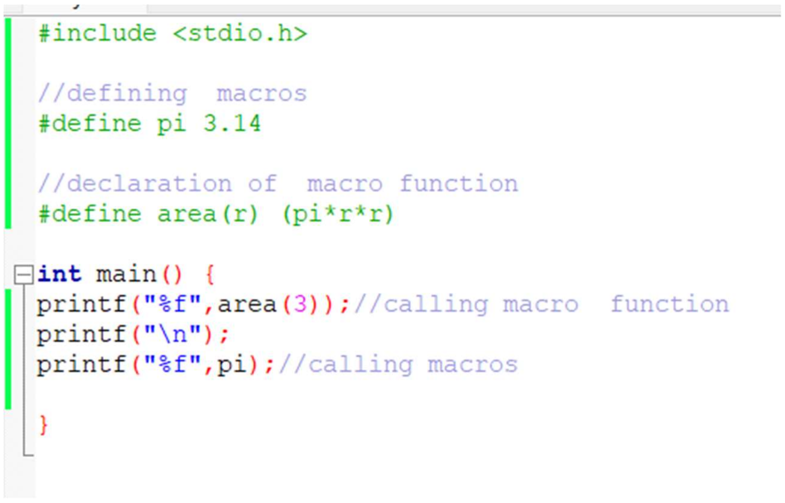
```
void add(int a,int b){
printf("%d",a+b);
}

int main() {
add(2,3);
return 0;
}
```

Call by reference



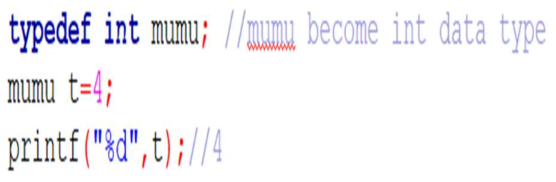
```
//concept  *x=value  &x=address
void swap(int *a,int *b){
int temp=*a;
*a=*b;
*b=temp;
}

int main() {
int x=2;int y=4;
printf("%d",x);//2
printf("%d",y);//4
swap(&x,&y);
printf("%d",x);//4
printf("%d",y);//2
return 0;
}
```

Macros and macro functions



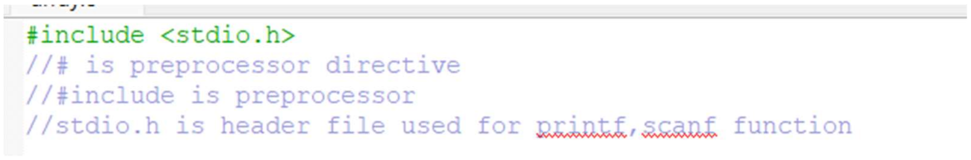
```
#include <stdio.h>

//defining  macros
#define pi 3.14

//declaration of  macro function
#define area(r) (pi*r*r)

int main() {
printf("%f",area(3));//calling macro  function
printf("\n");
printf("%f",pi);//calling macros

}
```

typedef



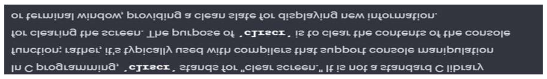
```
typedef int mumu; //mumu become int data type
mumu t=4;
printf("%d",t);//4
```

#preprocessor directive

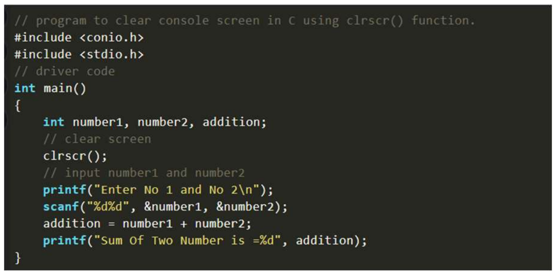


```
#include <stdio.h>
//# is preprocessor directive
//#include is preprocessor
//stdio.h is header file used for printf,scanf function
```

Clear console screen function

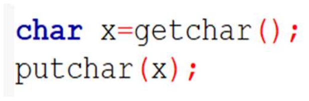


In C programming, `clrscr` stands for "clear screen." It is not a standard C library function; rather, it's typically used with compilers that support console manipulation for clearing the screen. The purpose of `clrscr` is to clear the contents of the console or terminal window, providing a clean slate for displaying new information.

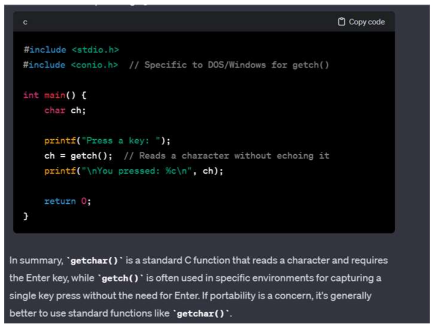


```
// program to clear console screen in C using clrscr() function.
#include <conio.h>
#include <stdio.h>
// driver code
int main()
{
    int number1, number2, addition;
    // clear screen
    clrscr();
    // input number1 and number2
    printf("Enter No 1 and No 2\n");
    scanf("%d%d", &number1, &number2);
    addition = number1 + number2;
    printf("Sum Of Two Number is =%d", addition);
}
```

getchar () and putchar () function for scanning and printing character



```
char x=getchar();
putchar(x);
```



c

```
#include <stdio.h>
#include <conio.h>  // Specific to DOS/Windows for getch()

int main() {
    char ch;

    printf("Press a key: ");
    ch = getch();  // Reads a character without echoing it
    printf("\nYou pressed: %c\n", ch);

    return 0;
}
```

In summary, `getchar()` is a standard C function that reads a character and requires the Enter key, while `getch()` is often used in specific environments for capturing a single key press without the need for Enter. If portability is a concern, it's generally better to use standard functions like `getchar()`.

Array

For understanding only

1. *Int dim[row]*
2. *Int dim2[row][column]*
3. *Int dim3[depth][row][column] (generally not used but can be)*

- *Row=sizeof(dim)/sizeof(dim[0])*
- *Column= sizeof(dim[0])/sizeof(dim[0][0])*

One dimensional Array

```
//one way
int arr[4];
arr[0]=1;
arr[1]=31;
arr[2]=21;
arr[3]=31;


//second way
int arr2[]={1,2,3,4,5};//1 2 3 4

//third way
int arr3[5]={1,2,3};// 1 2 3 0 0

//fourth way
int arr4[5];
scanf("%d",&arr4[0]);
printf("%d",arr4[0]);
```

```
//concept
int arr[3]={1,2,3,4,5};
// 1 2 3 x y where x y refer to any random number
```

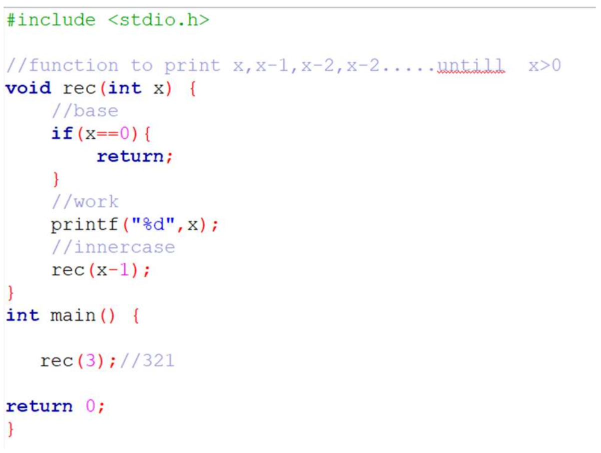
Multidimensional Array

2d and 3d array and many more

```
//one way
int arr[2][2];
arr[0][0]=1;
arr[0][1]=31;
arr[1][0]=21;
arr[1][1]=31;

//second way
int arr3[5][2]={{1,2},{3,4},{5,6},{7,8},{9,2}};// 1 2 3 0 0


//third way
int arr4[5][3];
scanf("%d",&arr4[0][0]);
printf("%d",arr4[0][0]);
```

Recursion

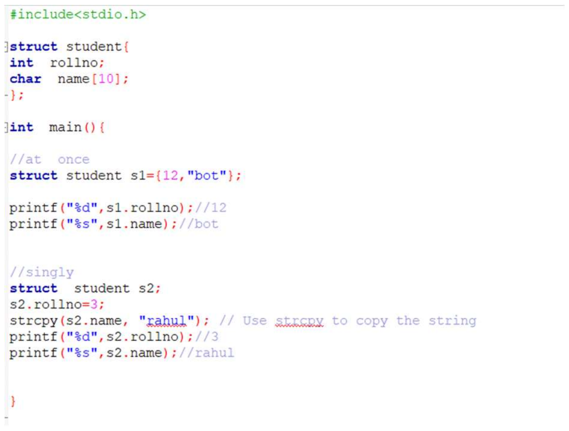
1. Base case
2. Work
3. Inner case



```c
#include <stdio.h>

//function to print x,x-1,x-2,x-2.....untill  x>0
void rec(int x) {
    //base
    if(x==0){
        return;
    }
    //work
    printf("%d",x);
    //innercase
    rec(x-1);
}
int main() {

   rec(3);//321

return 0;
}
```

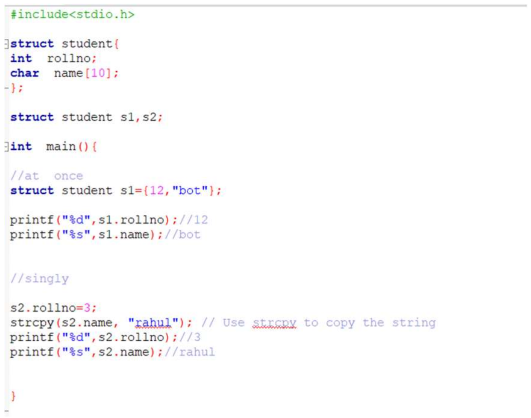
Structure

In main



```c
#include<stdio.h>

struct student{
int  rollno;
char  name[10];
};

int  main(){

//at  once
struct student s1={12,"bot"};

printf("%d",s1.rollno);//12
printf("%s",s1.name);//bot


//singly
struct  student s2;
s2.rollno=3;
strcpy(s2.name, "rahul"); // Use strcpy to copy the string
printf("%d",s2.rollno);//3
printf("%s",s2.name);//rahul


}
```

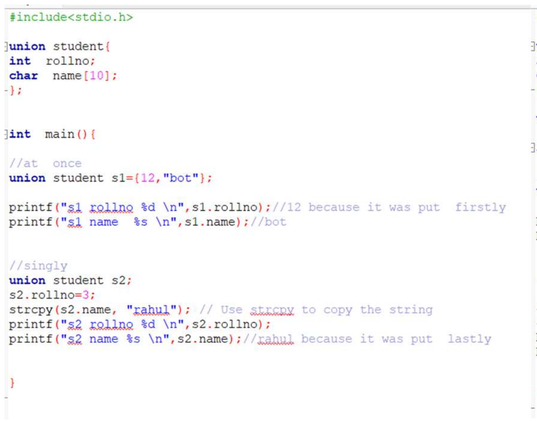
Globally



```c
#include<stdio.h>

struct student{
int  rollno;
char  name[10];
};

struct student s1,s2;

int  main(){

//at  once
struct student s1={12,"bot"};

printf("%d",s1.rollno);//12
printf("%s",s1.name);//bot


//singly

s2.rollno=3;
strcpy(s2.name, "rahul"); // Use strcpy to copy the string
printf("%d",s2.rollno);//3
printf("%s",s2.name);//rahul


}
```

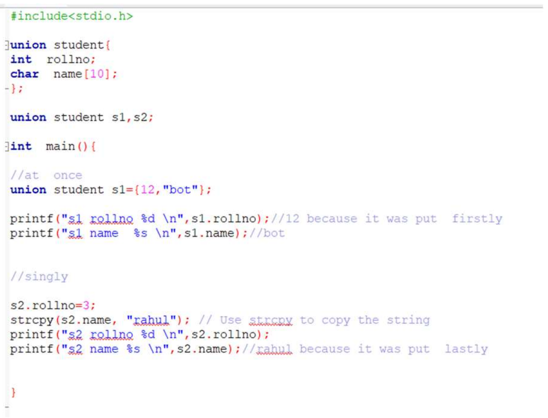
Union

In main



```c
#include<stdio.h>

union student{
int  rollno;
char  name[10];
};


int  main(){

//at  once
union student s1={12,"bot"};

printf("s1 rollno %d \n",s1.rollno);//12 because it was put  firstly
printf("s1 name  %s \n",s1.name);//bot


//singly
union student s2;
s2.rollno=3;
strcpy(s2.name, "rahul"); // Use strcpy to copy the string
printf("s2 rollno %d \n",s2.rollno);
printf("s2 name %s \n",s2.name);//rahul because it was put  lastly


}
```

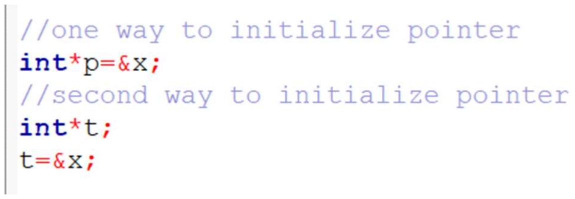
Globally



```c
#include<stdio.h>

union student{
int  rollno;
char  name[10];
};

union student s1,s2;

int  main(){

//at  once
union student s1={12,"bot"};

printf("s1 rollno %d \n",s1.rollno);//12 because it was put  firstly
printf("s1 name  %s \n",s1.name);//bot


//singly

s2.rollno=3;
strcpy(s2.name, "rahul"); // Use strcpy to copy the string
printf("s2 rollno %d \n",s2.rollno);
printf("s2 name %s \n",s2.name);//rahul because it was put  lastly


}
```

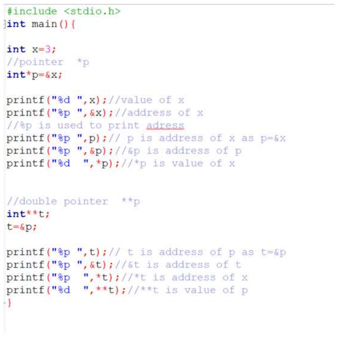
Pointer

Declaration

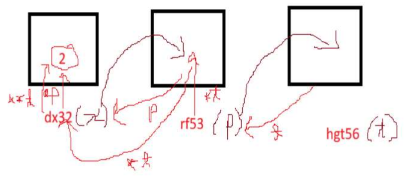


```
//one way to initialize pointer
int*p=&x;
//second way to initialize pointer
int*t;
t=&x;
```

Type of pointer: Pointer and Double Pointer



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

int x=3;
//pointer  *p
int*p=&x;

printf("%d ",x);//value of x
printf("%p ",&x);//address of x
//%p is used to print adress
printf("%p ",p);// p is address of x as p=&x
printf("%p ",&p);//&p is address of p
printf("%d  ",*p);//*p is value of x


//double pointer  **p
int**t;
t=&p;

printf("%p ",t);// t is address of p as t=&p
printf("%p ",&t);//&t is address of t
printf("%p  ",*t);//*t is address of x
printf("%d  ",**t);//**t is value of p
}
```

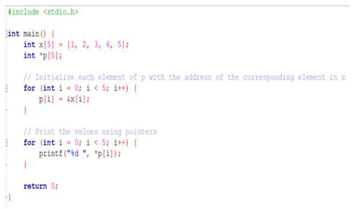


*p mean value of &x(address x)

*t mean value of &p(address p)

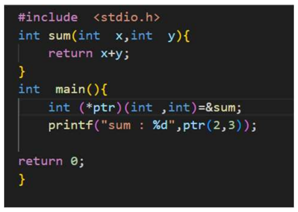
**t mean value of &x(address x)

Pointer to array



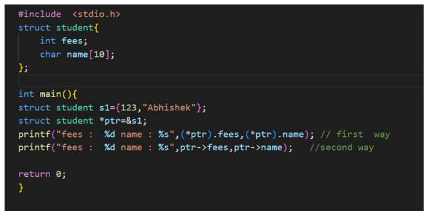
```
#include <stdio.h>

int main() {
    int x[5] = {1, 2, 3, 4, 5};
    int *p[5];

    // Initialize each element of p with the address of the corresponding element in x
    for (int i = 0; i < 5; i++) {
        p[i] = &x[i];
    }

    // Print the values using pointers
    for (int i = 0; i < 5; i++) {
        printf("%d ", *p[i]);
    }

    return 0;
}
```

Pointer to function



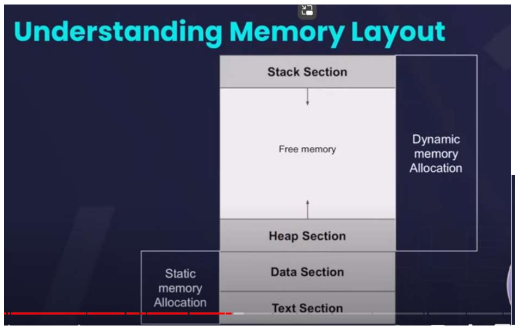
```
#include  <stdio.h>
int sum(int  x,int  y){
    return x+y;
}
int  main(){
    int (*ptr)(int ,int)=&sum;
    printf("sum : %d",ptr(2,3));

return 0;
}
```

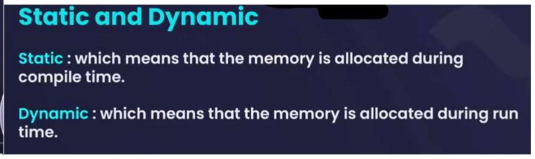
Pointer to structure



```
#include  <stdio.h>
struct student{
    int fees;
    char name[10];
};

int main(){
struct student s1={123,"Abhishek"};
struct student *ptr=&s1;
printf("fees :  %d name : %s",(*ptr).fees,(*ptr).name); // first  way
printf("fees :  %d name : %s",ptr->fees,ptr->name);   //second way

return 0;
}
```

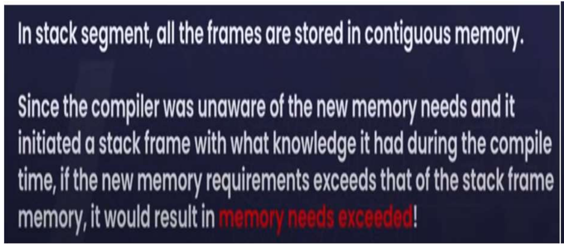
Memory allocation


Understanding Memory Layout

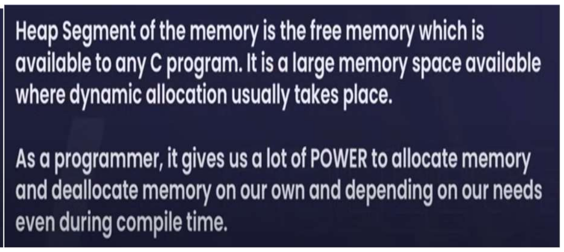

Static and Dynamic

Static : which means that the memory is allocated during compile time.

Dynamic : which means that the memory is allocated during run time.

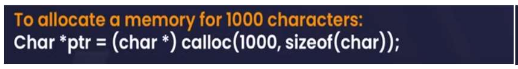

In stack segment, all the frames are stored in contiguous memory.

Since the compiler was unaware of the new memory needs and it initiated a stack frame with what knowledge it had during the compile time, if the new memory requirements exceeds that of the stack frame memory, it would result in memory needs exceeded!

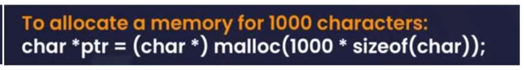

Heap Segment of the memory is the free memory which is available to any C program. It is a large memory space available where dynamic allocation usually takes place.

As a programmer, it gives us a lot of POWER to allocate memory and deallocate memory on our own and depending on our needs even during compile time.

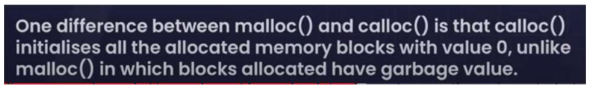
Allocating dynamic memory


To allocate a memory for 1000 characters:

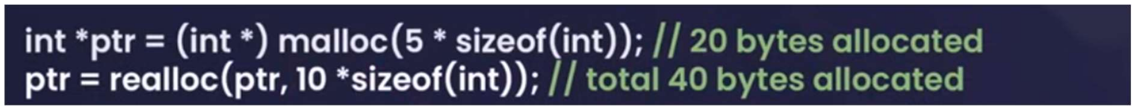
```
Char *ptr = (char *) calloc(1000, sizeof(char));
```


To allocate a memory for 1000 characters:

```
char *ptr = (char *) malloc(1000 * sizeof(char));
```

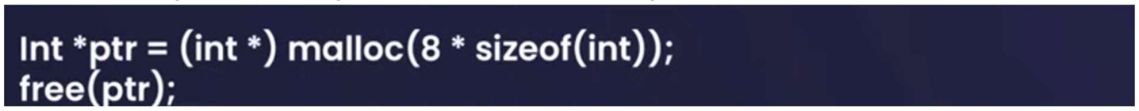

One difference between malloc() and calloc() is that calloc() initialises all the allocated memory blocks with value 0, unlike malloc() in which blocks allocated have garbage value.

Changing Size of dynamically allocated

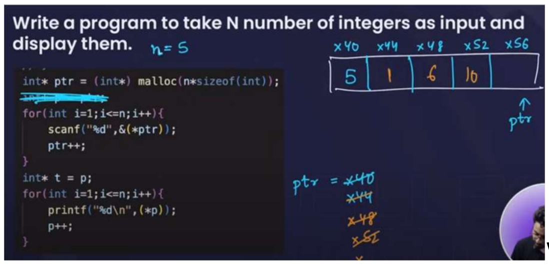


```
int *ptr = (int *) malloc(5 * sizeof(int)); // 20 bytes allocated
ptr = realloc(ptr, 10 *sizeof(int)); // total 40 bytes allocated
```

Free the dynamically allocated memory



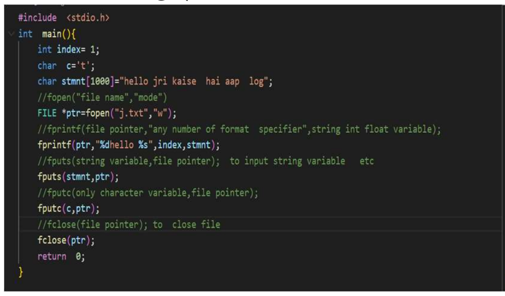
```
Int *ptr = (int *) malloc(8 * sizeof(int));
free(ptr);
```


Write a program to take N number of integers as input and display them.

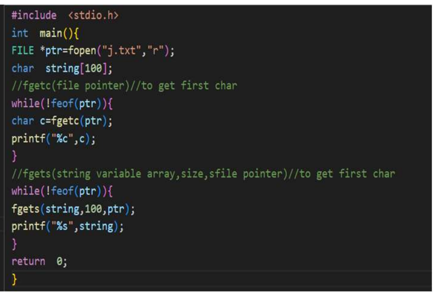
```
int* ptr = (int*) malloc(n*sizeof(int));
for(int i=1;i<=n;i++){
    scanf("%d",&(*ptr));
    ptr++;
}
int* t = p;
for(int i=1;i<=n;i++){
    printf("%d\n",(*p));
    p++;
}
```

we are using ptr as array

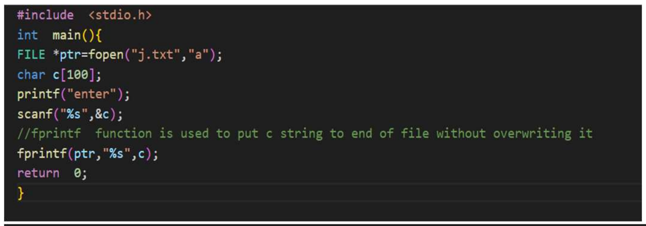
file handling (NOT USED IN TODAYS TIME BECAUSE OF MODERN TECHNOLOGY)



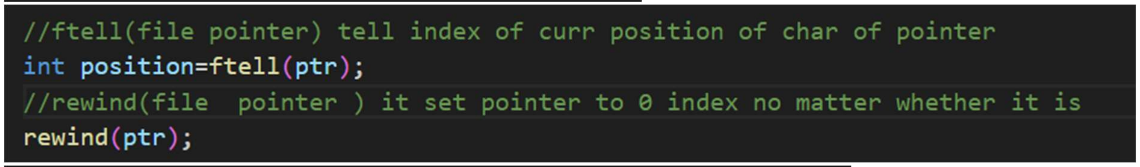
```c
#include  <stdio.h>
int  main(){
    int index= 1;
    char  c='t';
    char stmnt[1000]="hello jri kaise  hai aap  log";
    //fopen("file name","mode")
    FILE *ptr=fopen("j.txt","w");
    //fprintf(file pointer,"any number of format  specifier",string int float variable);
    fprintf(ptr,"%dhello %s",index,stmnt);
    //fputs(string variable,file pointer);  to input string variable   etc
    fputs(stmnt,ptr);
    //fputc(only character variable,file pointer);
    fputc(c,ptr);
    //fclose(file pointer); to  close file
    fclose(ptr);
    return  0;
}
```



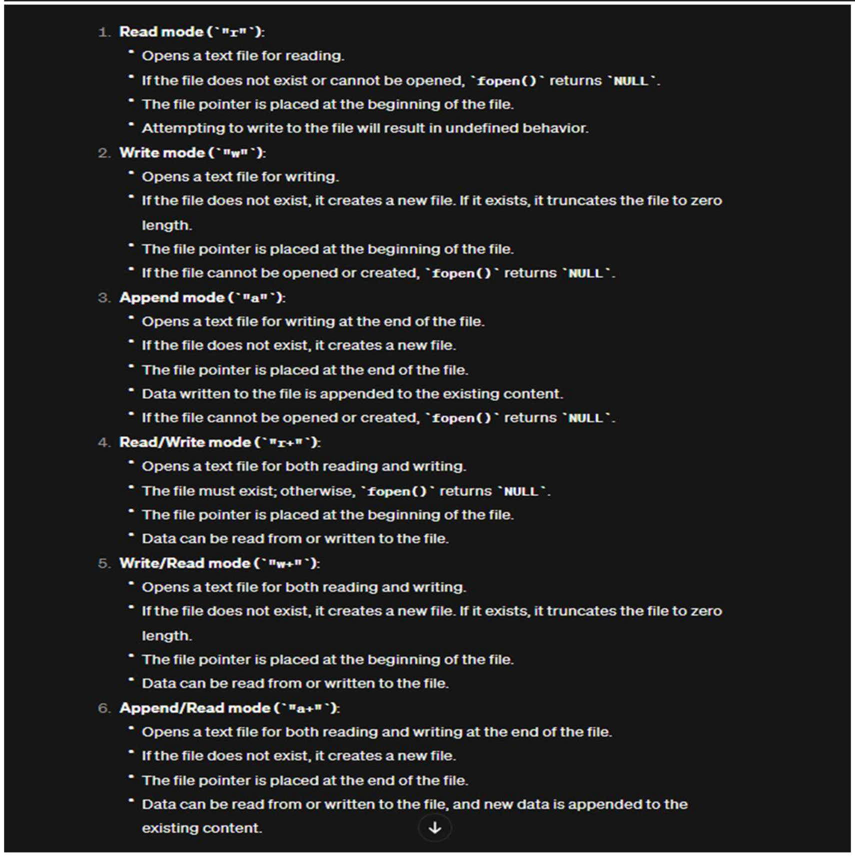
```c
#include  <stdio.h>
int  main(){
FILE *ptr=fopen("j.txt","r");
char  string[100];
//fgetc(file pointer)//to get first char
while(!feof(ptr)){
char c=fgetc(ptr);
printf("%c",c);
}
//fgets(string variable array,size,sfile pointer)//to get first char
while(!feof(ptr)){
fgets(string,100,ptr);
printf("%s",string);
}
return  0;
}
```



```c
#include  <stdio.h>
int  main(){
FILE *ptr=fopen("j.txt","a");
char c[100];
printf("enter");
scanf("%s",&c);
//fprintf  function is used to put c string to end of file without overwriting it
fprintf(ptr,"%s",c);
return  0;
}
```

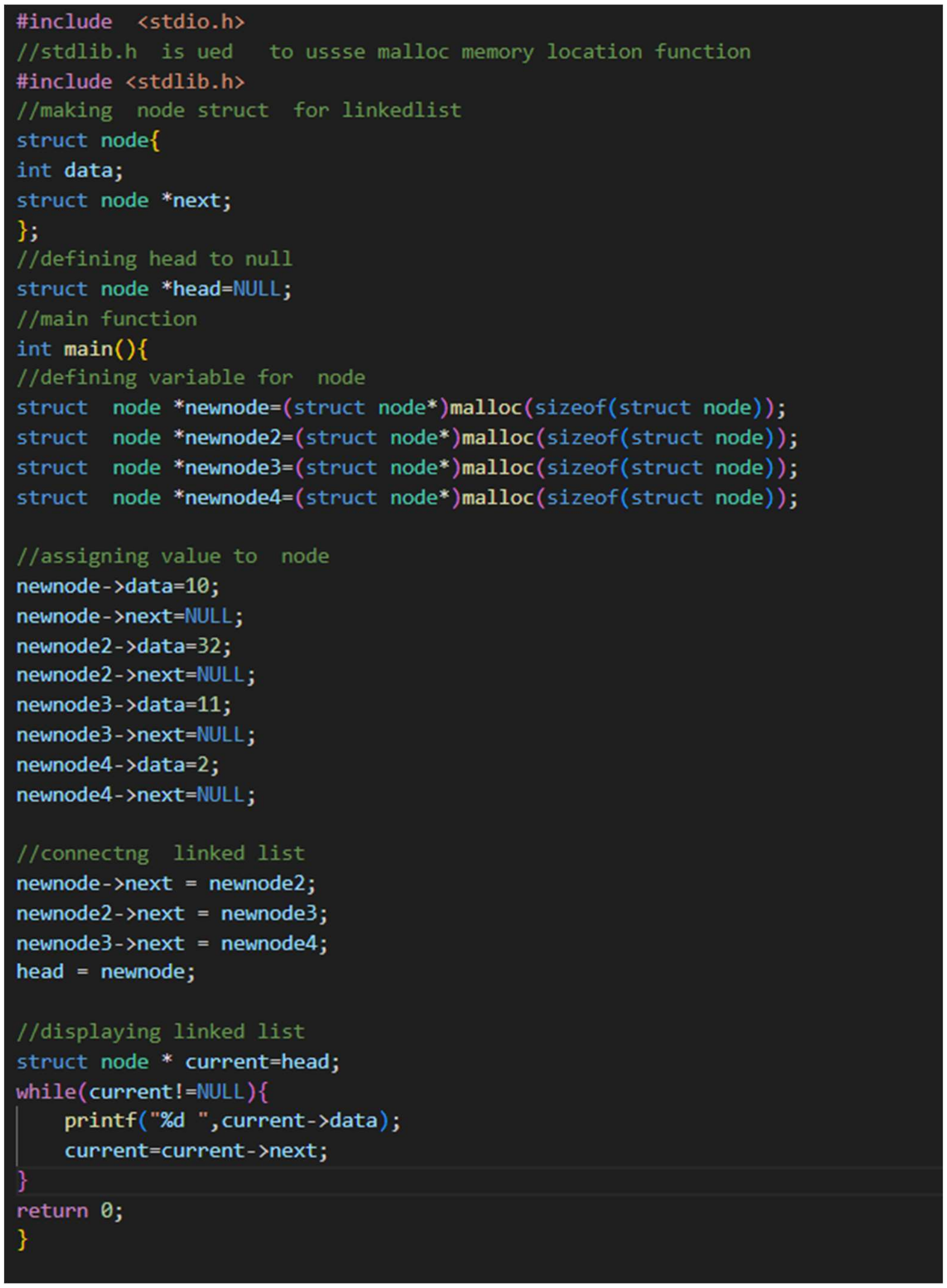


```c
//ftell(file pointer) tell index of curr position of char of pointer
int position=ftell(ptr);
//rewind(file  pointer ) it set pointer to 0 index no matter whether it is
rewind(ptr);
```



1. **Read mode (`"r"`):**
   - Opens a text file for reading.
   - If the file does not exist or cannot be opened, `fopen()` returns `NULL`.
   - The file pointer is placed at the beginning of the file.
   - Attempting to write to the file will result in undefined behavior.
2. **Write mode (`"w"`):**
   - Opens a text file for writing.
   - If the file does not exist, it creates a new file. If it exists, it truncates the file to zero length.
   - The file pointer is placed at the beginning of the file.
   - If the file cannot be opened or created, `fopen()` returns `NULL`.
3. **Append mode (`"a"`):**
   - Opens a text file for writing at the end of the file.
   - If the file does not exist, it creates a new file.
   - The file pointer is placed at the end of the file.
   - Data written to the file is appended to the existing content.
   - If the file cannot be opened or created, `fopen()` returns `NULL`.
4. **Read/Write mode (`"r+"`):**
   - Opens a text file for both reading and writing.
   - The file must exist; otherwise, `fopen()` returns `NULL`.
   - The file pointer is placed at the beginning of the file.
   - Data can be read from or written to the file.
5. **Write/Read mode (`"w+"`):**
   - Opens a text file for both reading and writing.
   - If the file does not exist, it creates a new file. If it exists, it truncates the file to zero length.
   - The file pointer is placed at the beginning of the file.
   - Data can be read from or written to the file.
6. **Append/Read mode (`"a+"`):**
   - Opens a text file for both reading and writing at the end of the file.
   - If the file does not exist, it creates a new file.
   - The file pointer is placed at the end of the file.
   - Data can be read from or written to the file, and new data is appended to the existing content.

Linked structure/self-referential structure



```
#include  <stdio.h>
//stdlib.h  is ued   to ussse malloc memory location function
#include <stdlib.h>
//making  node struct  for linkedlist
struct node{
int data;
struct node *next;
};
//defining head to null
struct node *head=NULL;
//main function
int main(){
//defining variable for  node
struct  node *newnode=(struct node*)malloc(sizeof(struct node));
struct  node *newnode2=(struct node*)malloc(sizeof(struct node));
struct  node *newnode3=(struct node*)malloc(sizeof(struct node));
struct  node *newnode4=(struct node*)malloc(sizeof(struct node));

//assigning value to  node
newnode->data=10;
newnode->next=NULL;
newnode2->data=32;
newnode2->next=NULL;
newnode3->data=11;
newnode3->next=NULL;
newnode4->data=2;
newnode4->next=NULL;

//connectng  linked list
newnode->next = newnode2;
newnode2->next = newnode3;
newnode3->next = newnode4;
head = newnode;

//displaying linked list
struct node * current=head;
while(current!=NULL){
    printf("%d ",current->data);
    current=current->next;
}
return 0;
}
```