Bangladesh University of Business and Technology

Department of Computer science and Engineering

MD: Rakib Rahman

C programing

starter.c

1#include<stdio.h> [s=Standrad i=input o=output h=Hader file]

2

3 int main(){ [main - Function][void is not return=0]

4 printf("Hello world"); [printf- outout Result]

5 return 0;

6 }

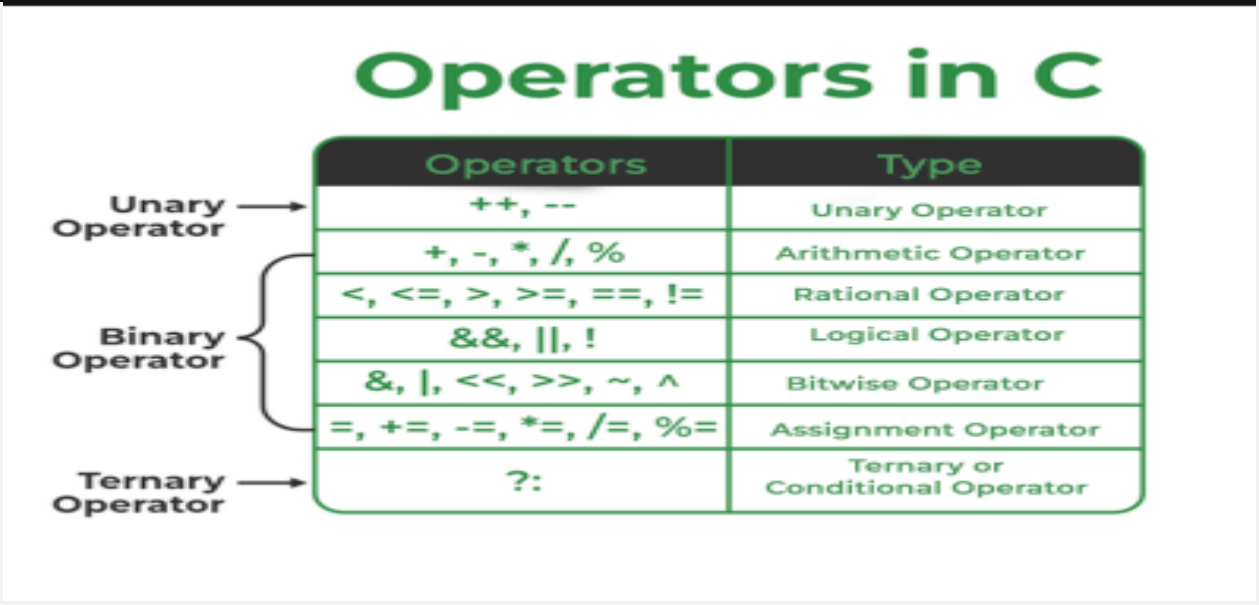
C Tokens

Keywords	constants	strings
float	-16.5	“ABC”
while	100	“Year”

identifers	operators	symbols
Main	+ , -	[]
Amount	*,/	{}

Data Types

Type	formate	Size byte
char	%c	1
int	%d	4
float	%f	4
Double	%lf	8
Long double	%lf	



```c
#include <stdio.h>

int main()
{
 int a = 9, b = 65;

 printf(" AND a&b = %d \n", a & b);
 printf(" OR a|b = %d \n", a | b);
 printf(" EXCLUSIVE OR a^b = %d \n", a ^ b);
 printf(" NOT ~a = %d \n", a = ~a);

 printf(" LEFT SHIFT a<<1 = %d \n", a << 1);
 printf(" RIGHT SHIFT b>>1 = %d \n", b >> 1);

 return 0;
}
```

AND Operation = a & b
0001001 & 1000001 = 0000001 = 1

OR Operation = a || b
0001001 || 1000001 = 1001001 = 73

Next, Exclusive OR Operation = a ^ b
0001001 ^ 1000001 = 1001000 = 72

Left Shift Operation = b >> 1
1000001 >> 1 = 0100000 = 32

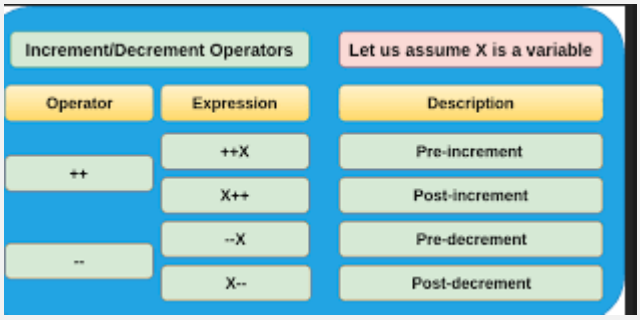
Logical Operators

operator	Meaning
&&	Logical AND
!	Logical NOT
||	Logical OR

Assignment Operators

Simple assignment	Shorthand operator
a=a+1	a+=1
a=a-1	a-=1
a=a*(m+n)	a*=m+n
a=a/(m+n)	a/=m+n
a=a%b	a%=b

Increment & Decrement operators



1. Increment x++ Note: x++ -post[1+]

   ++x -pre [1+]

2.Decrement x-- Note: x-- -post[1-]

   --x -pre [1-]

Ex: Evaluate the expression when a=4

b=a- ++a

=a-5

=5-5

=0

If condision

Expression is True

```
int test = 5;
if (test < 10 )
{
  // body of if
}
else
{
  // body of else
}
```

Expression is False

```
int test = 5;
if (test > 10 )
{
  // body of if
}
else
{
  // body of else
}
```

Expression is True

```
int test = 5;
if (test < 10 )
{
  // body of if
}
else
{
  // body of else
}
```

Expression is False

```
int test = 5;
if (test > 10 )
{
  // body of if
}
else
{
  // body of else
}
```

Expression is true

```
// codes before if-else

if (number < 5)
{
      number += 5;
}
else
{
      number -= 5;
}

// codes after if-else
```

Expression is false

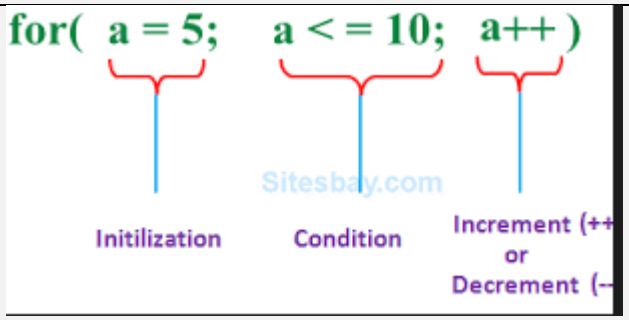
```
// codes before if-e

if (number < 5)
{
      number += 5;
}
else
{
      number -= 5;
}

// codes after if-el
```

Switch case

```
#include<stdio.h>
 int main(){
  int n=1;
  switch(n){
  case 0:
       printf("Friday");
       break;
   case 1:
        printf("Saturday");
        break;
   case 2:
         printf("sunday");
        break;
    default:
         printf("please Try Again");
}
  Return 0;
}
```

Loop

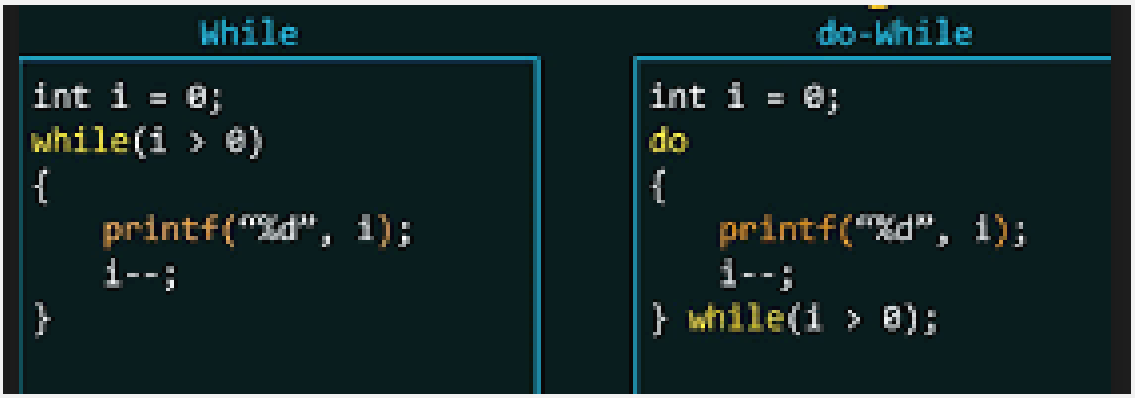
```
#include<stdio.h>
int main(){
int i;
for(i=1;i<10;i++)
{
 printf("Rakib\n");          note: new line "\n"
}                                tab line "\t"
return 0;
}
```

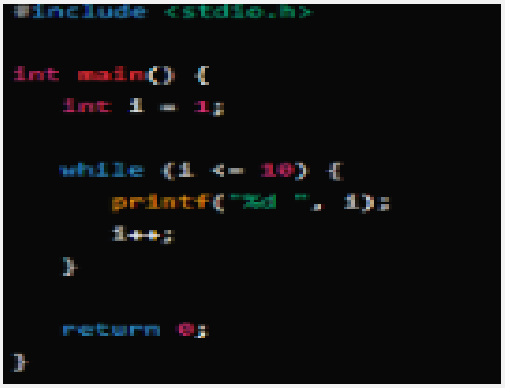



```
int main()
{
    int i, n;
    printf("Enter the number of integers to be printed
    scanf("%d", &n);
    for(i=1;i<=n;i++)
    {
        printf("%d\t", i);
    }
    return 0;
}
```

While loop and do while loop

```
#include<stdio.h>
Int main(){
  int i=10;
  while( I < 12 ){
   printf("Hi\n");
  i++;
  }
```



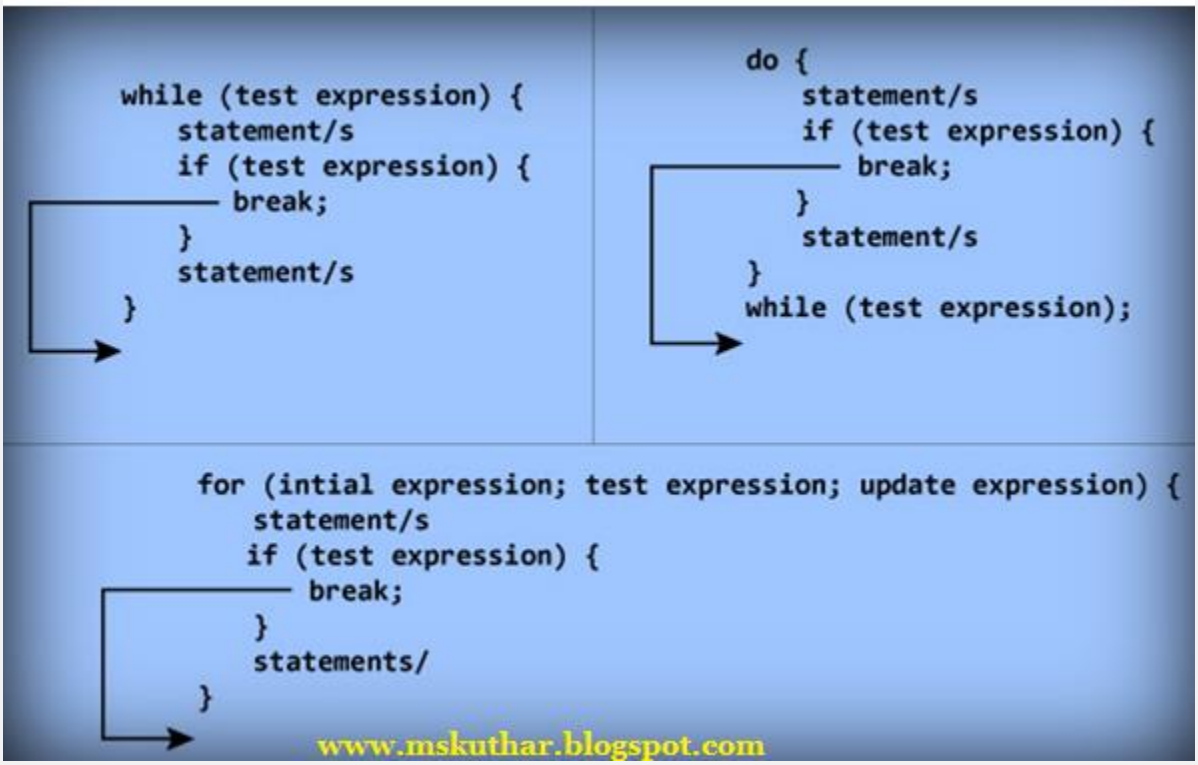


Continue Loop

```
#include<stdio.h>
 int main(){
 int i;
 for(i=1;i<=10;i++){
 if( i%3 == 0 ){
  continue;
}
 printf("%d",i);
}
 return 0;
}
```

Break

```
#include<stdio.h>
int main(){
int i;
for( I = 1 ; I < = 10 ; i++ ){
 if( I == 5 ){
  break;
}
printf("%d",i);
}
return 0;
}
```




```c
#include <stdio.h>

int main() {
    int n, i, j;
    scanf("%d", &n);
    for (i = 1; i <= n; i++) {
        for (j = 1; j <= 2 * n - 1; j++) {
            if (j >= n - (i - 1) && j <= n + (i - 1)) {
                printf("*");
            } else {
                printf(" ");
            }
        }
        printf("\n");
    }
    return 0;
}
```

```
C:\Users\User\Documents\20244103417.exe
5
    *
   ***
  *****
 *******
*********

Process returned 0 (0x0)   execut
```

Nested loop

```
#include<stdio.h>
int main(){
int  i , j ;
for( i =1; I < = 5 ; i++ )
 {
 for( j = 1 ; j < = 5 ; j++ )
 {
 printf("%d",i);
}
printf("\n");
}
return 0;
}
```

Array operations in c

Case:1

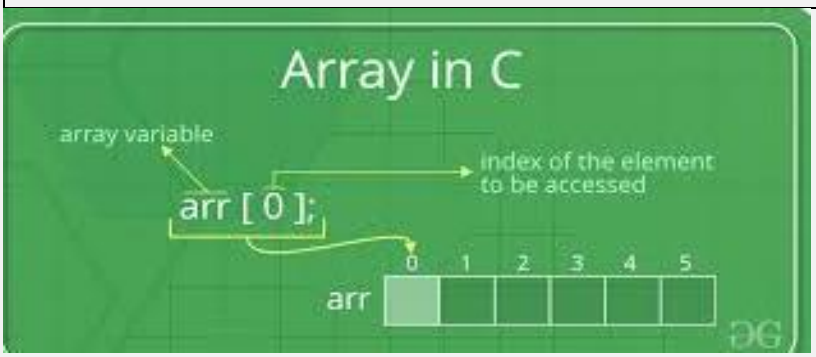
```
Int  ara [ ]={10,20,30,40};
printf("%d\n",ara[0]);
printf("%d\n",ara[1]);
printf("%d\n",ara[2]);
printf("%d",ara[3]);
```

case:2

```
#include<stdio.h>
Int main(){
int ara[4];
ara[0] = 10;
ara[1] = 20;
ara[2] = 30;
ara[3] = 40;
printf("%d\n",ara[0]);
printf("%d\n",ara[1]);
printf("%d\n",ara[2]);
printf("%d",ara[3]);
return 0;
}
```

for loop array

```
#include<stdio.h>
Int main(){
int n;
scanf("%d",&n);
int ara[n];
for(int i=0;i<n;i++)
{
scanf("%d",&ara[i]);
}
for(int i=0;i<n;i++)
{
printf("Output: %d\n",ara[i]);
}
return 0;
}
```



- Function:

Syntax: data _type function _name (data_ type _parameters parameters) { //Code

Example1:

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

 printf (“%d”, a) ;

 }

 int   main ( ){
 int a=5;
 display(a);
}
 return 0;
}
```

Example2:

```
#include<stdio.h>
 Int main(){
Int   display (int a[], int n)
{
for(int i=0; i<n; i++)
{
printf("%d ",a[i]);
  }
}
int main ()
{
 int ara[5]= {5,4};
display(ara,5);
 }
 return 0;
 }
```

• 2D Array: Syntax: data_type array_name [row_size][column_size]

Example:

int ara[2][3];

ara[0][0]	ara[0][1]	ara[0][2]
ara[1][0]	ara[1][1]	ara[1][2]

• Declare and Initialize 2D Array:

Example-1:

```
#include Int main(){
int ara [] [3] = {{1,2,3} , {4,5,6}};
printf("%d ",ara[0][0]);
printf("%d ",ara[0][1]);
printf("%d ",ara[0][2]);
printf("%d ",ara[1][0]);
printf("%d ",ara[1][1]);

printf("%d ",ara[1][2]);

return 0;

}
```

Example-2

```
#include<stdio.>
Int main(){
int ara[][3] = {{1,2,3},{4,5,6}};
for(int i=0; i<2; i++)
  {
     for(int j=0; j<3; j++)
  {
  printf("%d ",ara[i][j]);
}
printf("\n");
}
Return 0;
}
```

Example-3:

```
 int n,m;
 scanf("%d %d",&n,&m);
 int ara[n][m];
 for(int i=0; i<n; i++)
 {
 for(int j=0; j<m; j++)
 {
 scanf("%d",&ara[i][j]);
    }
  }
for(int i=0; i<n; i++){
 for(int j=0; j<m; j++)
 {
 printf("%d ",ara[i][j]);
 }
 printf("\n");
}
```

2 by 2

```c
#include<stdio.h>
int main(){
int A[3][3]={{4,4,3},{2,1,0}};
int B[3][3]={{5,3,4},{6,1,7}};
int C[3][3];

 for(int i=0; i<3; i++)
 {
  for(int j=0; j<3; j++)
   {
     C[i][j]=0;

     for(int k=0;k<3;k++){
        C[i][j] +=A[i][k] + B[k][j];
     }
    }
  }
   for(int i=0; i<3; i++){
    for(int j=0; j<3; j++)
    {
   printf("%d ",C[i][j]);
 }
 printf("\n");
}
return 0;
}
```

3 by 3

```
#include <stdio.h>
int main(){
    int A[3][3];
    int B[3][3];
    int C[3][3];
    int i,j;
    for(i=0; i<3; i++)
    {
        for(j=0; j<3; j++)
        {
            scanf("%d", &A[i][j]);
        }
    }
    for(i=0; i<3; i++)
    {
        for(j=0; j<3; j++)
        {
            scanf("%d", &B[i][j]);
```

```
        }
    }
    for(i=0; i<3; i++)
    {
        for(j=0; j<3; j++)
        {
            C[i][j] = A[i][j] - B[i][j];
        }
    }
    for(i=0; i<3; i++)
    {
        for(j=0; j<3; j++)
        {
            printf("%d ", C[i][j]);
        }
        printf("\n");
    }
    return 0;
}
```

```c
#include <stdio.h>

#define SIZE 3 // Matrix size

int main()
{
    int A[SIZE][SIZE];
    int row, col, sum = 0;
    for(row=0; row<SIZE; row++)
    {
        for(col=0; col<SIZE; col++)
        {
            scanf("%d", &A[row][col]);
        }
    }
    for(row=0; row<SIZE; row++)
    {
        sum = sum + A[row][row];
    }

    printf("Sum of main diagonal elements = %d\n", sum);

    return 0;
}
```

1 2 3

4 5 6

7 8 9

- Passing 2D Array in Function:

```
#include<stdio.h>
Int main(){
int m;
void display(int ara[][m],int n)
{
 for(int i=0; i<n; i++)
 {
 for(int j=0; j<m; j++)
 {
 printf("%d ",ara[i][j]);
 }
 printf("\n");
  }
}
int main()
{
 int n;
scanf("%d %d",&n,&m);
int ara[n][m];
 for(int i=0; i<n; i++)
  {
 for(int j=0; j<m; j++)
 {
scanf("%d",&ara[i][j]);
  }

 }

 display(ara,n);
 return 0;
 }
```

string

- Declare and Initialization:case:1

```
#include<stdio.h>
int main(){
char s[] = {'a','b','c','\0'};
printf("%s\n",s);
puts(s);
return 0;
}
```

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

Case:2

```
char s[10],s1[10];
scanf("%s",&s);
gets(s1)
printf("%s\n",s);
puts(s1);
return 0;
}
```

case:3

```
#include<stdio.h>
int main()
{
char s[10];
int n;
scanf("%d",&n);
gets(s);
printf("%d\n",n);
puts(s);
return 0;
}
```

- String function:

```
#include<stdio.h>
#include<string.h>
int main()
{
char s[10];
gets(s);
char s2[10];
gets(s2);
printf("s= %s \ns2= %s \n",s,s2);

printf("string length of s= %d\n",strlen(s));

printf("string copy from s to s2 = %s\n",strcpy(s2,s));
printf("string merge both  s2 = %s\n",strcat(s,s2));

printf("string compare  s and s2 =  %d\n",strcmp(s,s2));

return 0;

}
```

Recursion

. A recursion is a function which is call itself.

Case:1

```
#include<stdio.h>
int fun(int n)
{
 If(n==1) return 1;
 return 1 + fun(n-1);
}
int main()
{
 int n=3;

 printf("%d",fun(n));
 return 0;
}
```

Case:2

```
#include<stdio.h>
int power(int a,int n)
{
 if(n==0) return 1;
 return a * power (a, n-1);
}
int main ()
{
 int n=3,a=3;
 printf ("%d", power (a,n));
 return 0;
}
```

Case:3

```
#include<stdio.h>
int fact(int n)
{
 if(n==0)return 1;
 return n * fact(n-1);
}
int main()
{
 int n=5;
 printf("%d",fact(n));
 return 0;
}
```

Case:4

```
#include<stdio.h>
int sum (int n)
{
 if(n==0)return 0;
 return n + sum(n-1);
}
int main()
{
 int n=10;
 printf("%d",sum(n));
 return 0;
}
```

Case:5

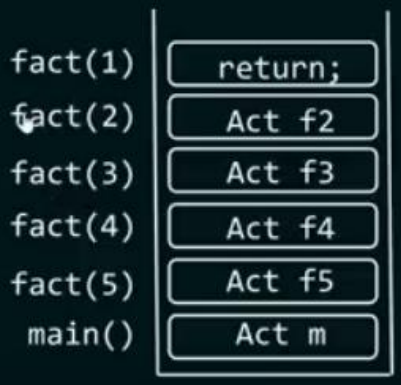
```
#include<stdio.h>
int sum(int ara[], int n)
{
 if(n==0)return ara[0];

 return ara[n] + sum(ara,n-1);
}

int main()

{

 int ara[]={1,2,3,4};

 printf("%d",sum(ara,3));

 return 0;

}
```

Iterative

```
int fact(int n) {
    int res=1;
    while(n!=0) {
        res = res*n;
        n--;
    }
    return res;
}
int main() {
    printf("%d", fact(5));
    return 0;
}
```

Recursive

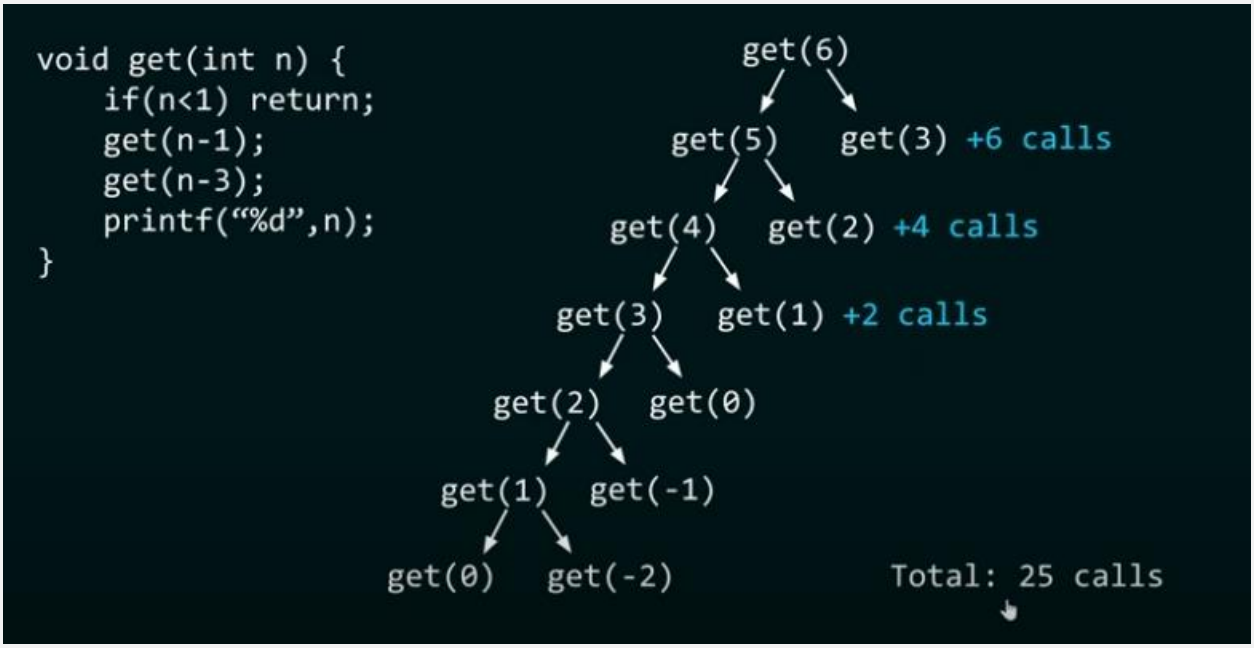
```
int fact(int n) {
    if(n==1)
        return 1;
    else
        return n*fact(n-1);
}
int main() {
    printf("%d", fact(5));
    return 0;
}
```



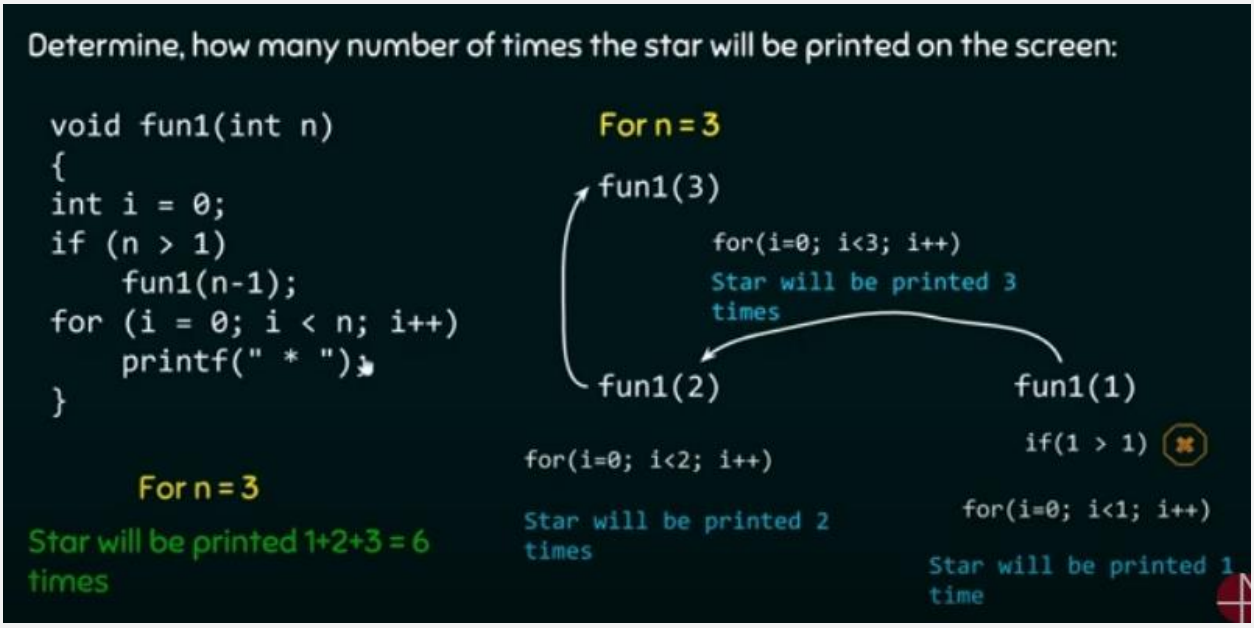

```
#include<stdio.h>
int sum(int roll[], int n){
if(n==0)
   return roll[0];
return roll[n] + sum(roll,n-1);
}
```

```
int main(){
 int
roll[]={2,0,2,4,4,1,0,3,4,1,7};
 printf("%d\n",sum(roll,10));
 return 0;
}
```

Q:1


void get(int n) {
if(n<1) return;
get(n-1);
get(n-3);
printf("%d",n);
}
get(6)
get(5)
get(3) +6 calls
get(4)
get(2) +4 calls
get(3)
get(1) +2 calls
get(2)
get(0)
get(1)
get(-1)
get(0)
get(-2)
Total: 25 calls

Q:2


Determine, how many number of times the star will be printed on the screen:
void fun1(int n)
{
int i = 0;
if (n > 1)
fun1(n-1);
for (i = 0; i < n; i++)
printf(" * ");
}
For n = 3
Star will be printed 1+2+3 = 6 times
For n = 3
fun1(3)
for(i=0; i<3; i++)
Star will be printed 3 times
fun1(2)
fun1(1)
for(i=0; i<2; i++)
Star will be printed 2 times
if(1 > 1)
for(i=0; i<1; i++)
Star will be printed 1 time

Q:3

Consider the C function given below:

```
int f(int j)
{
  static int i = 50;
  int k;
  if (i == j)
  {
    printf("something");
    k = f(i);
    return 0;
  }
  else return 0;
}
```

Which one of the following is TRUE?

a) The function returns 0 for all values of j.
b) The function prints the string "something" for all values of j.
c) The function returns 0 when j = 50.
d) The function will exhaust the runtime stack or run into an infinite loop when j = 50.

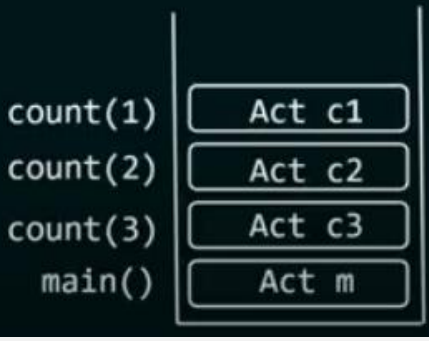
[GATE 2014 (Set 2)]

Q:4

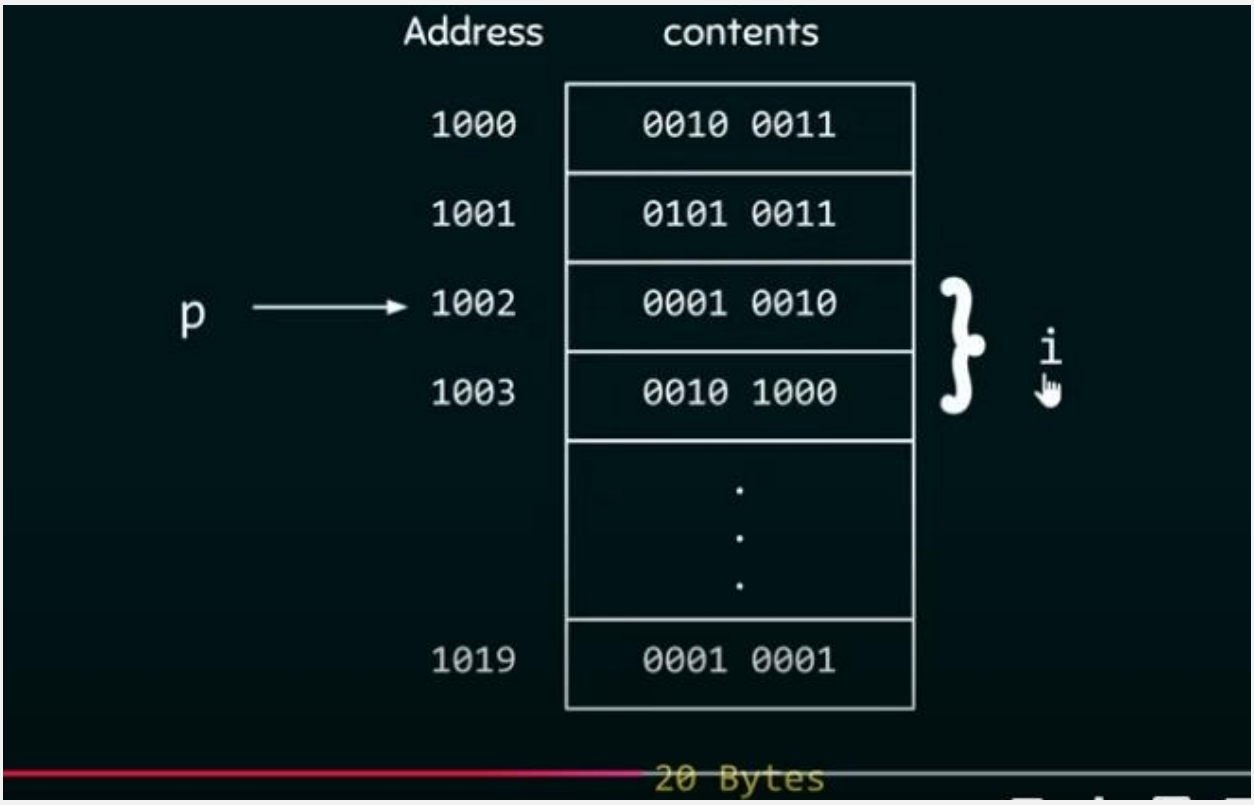
What will be the output of the following C program?

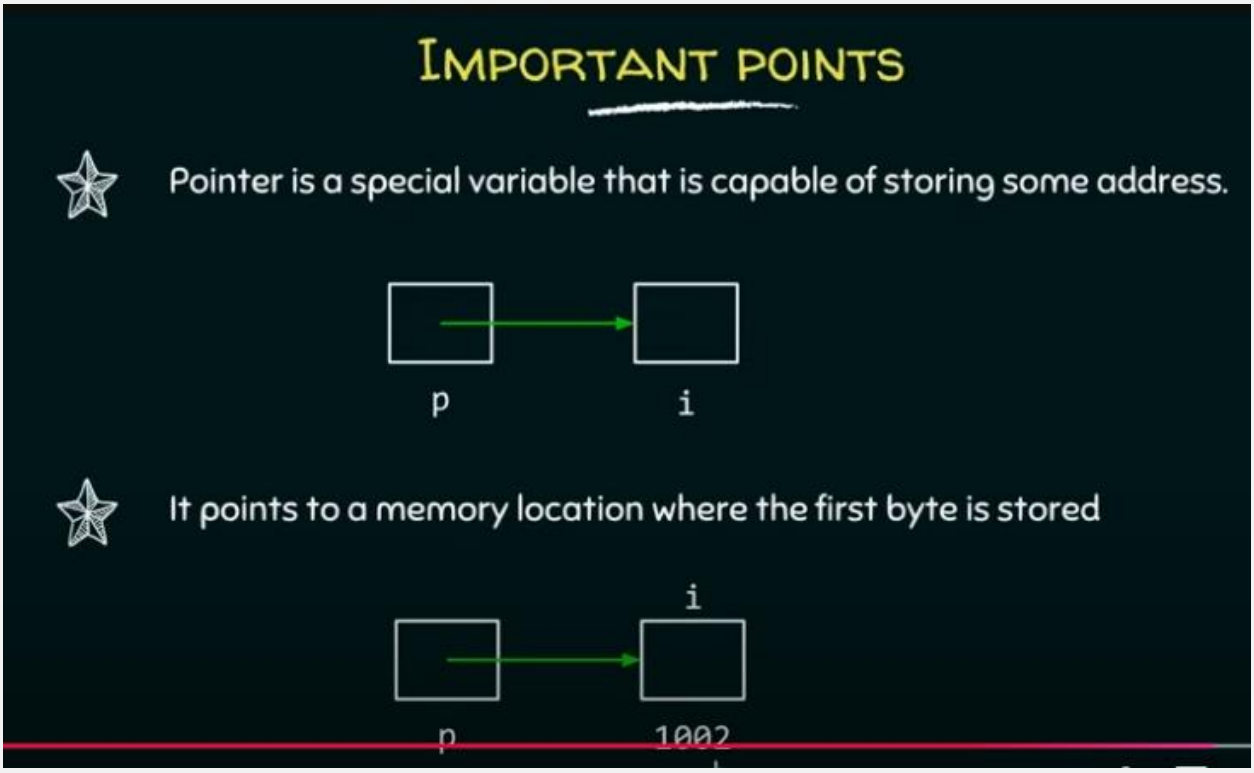
```
void count(int n)
{
    static int d = 1;
    printf("%d ", n);
    printf("%d ", d);
    d++;
    if(n > 1) count(n-1);
    printf("%d ", d);
}
int main()
{
    count(3);
}
```

a) 3 1 2 2 1 3 4 4 4
b) 3 1 2 1 1 1 2 2 2
c) 3 1 2 2 1 3 4
d) 3 1 2 1 1 1 2



Pointers


Address
contents
1000
0010 0011
1001
0101 0011
p
1002
0001 0010
1003
0010 1000
i
1019
0001 0001
20 Bytes


IMPORTANT POINTS
Pointer is a special variable that is capable of storing some address.
p
i
It points to a memory location where the first byte is stored
i
p
1002

Syntax for declaring pointer variables

General syntax for declaring pointer variables:

```
data_type *pointer_name
```

Here data type refers to the type of the value that the pointer will point to.

For example:

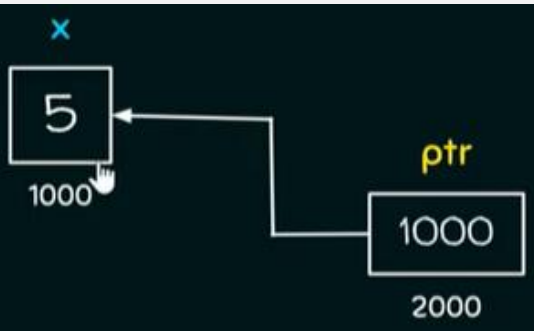
```
int *ptr;
```
Points to integer value

```
char *ptr;
```
Points to character value

```
float *ptr;
```
Points to float value

```
int x = 5;

int *ptr;

ptr = &x;
```

address of operator



```
int x = 5;

int *ptr;

ptr = &x;
```

is equivalent to

```
int x = 5;

int *ptr;

ptr = &x;
```

is equivalent to

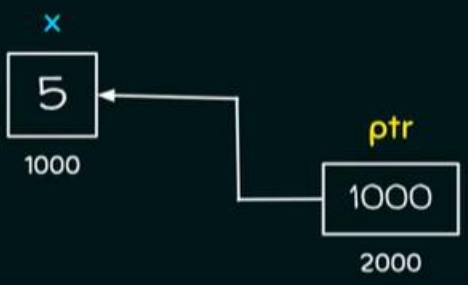
```
int x = 5, *ptr = &x;
```

Value of operator/indirection operator/dereference operator is an operator that is used to access the value stored at the location pointed by the pointer.

```
int x = 5;

int *ptr;

ptr = &x;

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

It says go to the address of object and take what is stored in the object

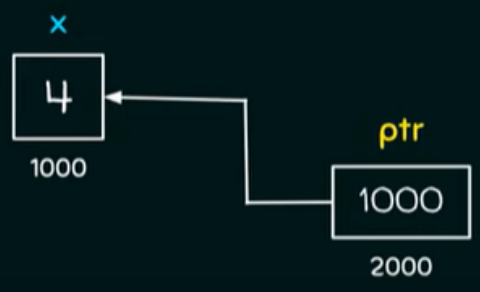
VALUE OF OPERATOR



We can also change the value of the object pointed by the pointer.

For example:

```
int x = 10;
int *ptr = &x;

*ptr = 4;

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

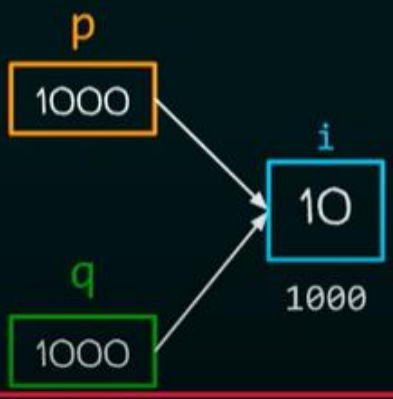


Output: 4

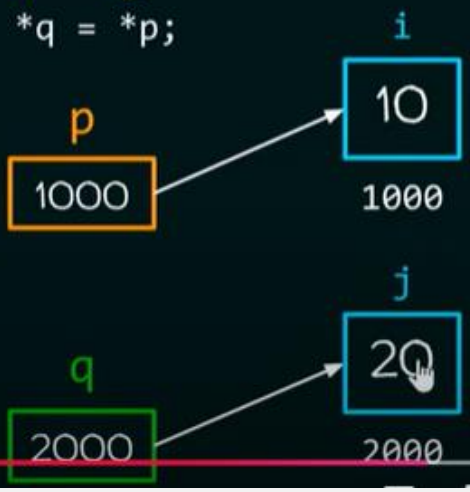
Type:4

NOTE: q = p is not same as *q = *p

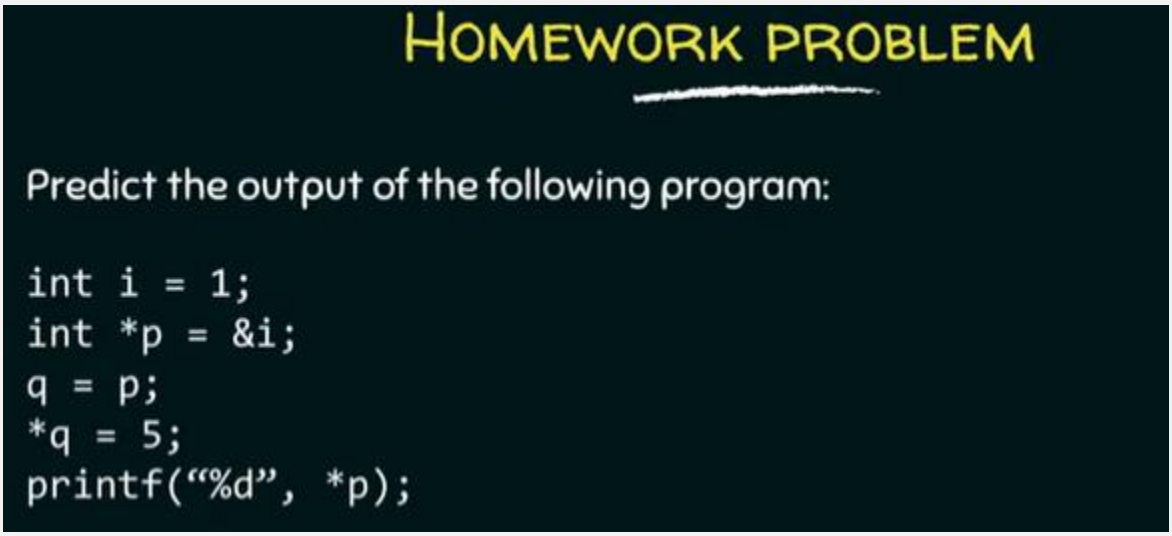
```
int i = 10;
int *p, *q;
p = &i;
q = p;
```



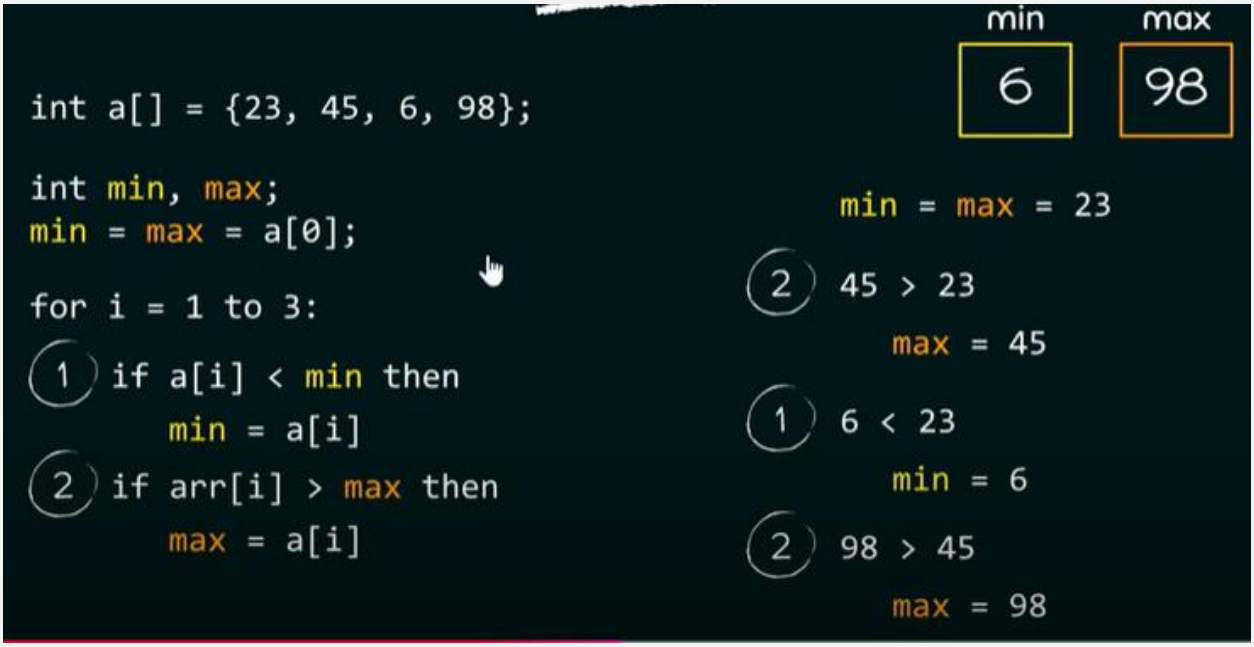

```
int i = 10, j = 20;
int *p, *q;
p = &i;
q = &j;
*q = *p;
```



Homework



Type:5 Finding the largest and smallest pointers



Example: maximam and minimam pointers

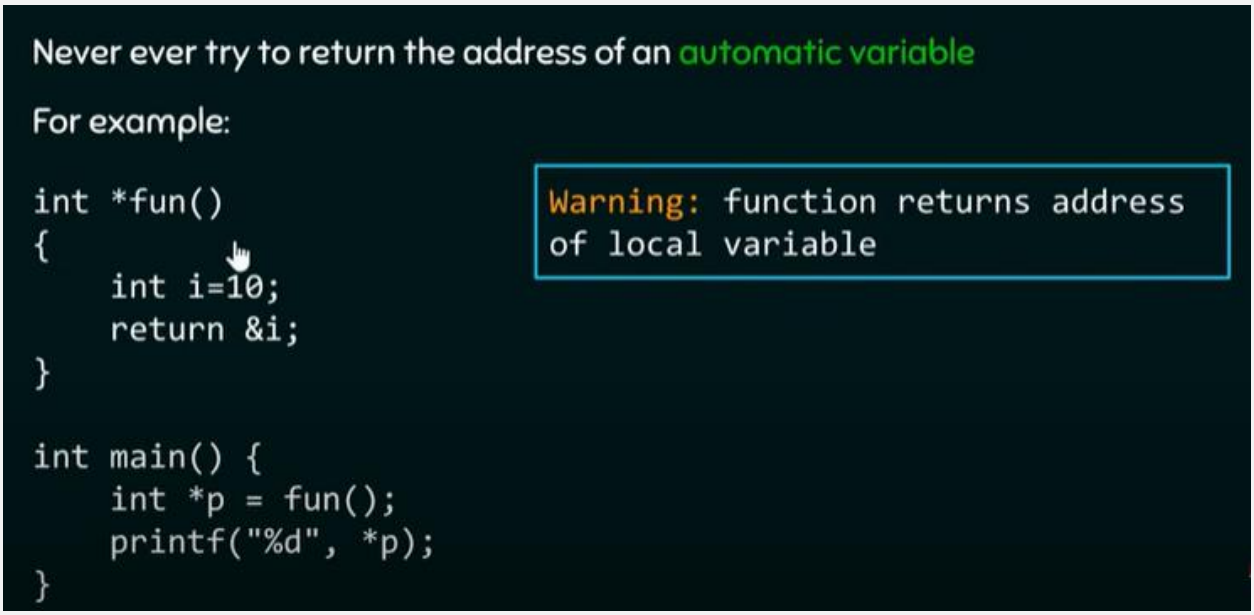
```c
#include <stdio.h>

void minMax(int arr[], int len, int *min, int *max)
{
    *min = *max = arr[0];
    int i;
    for(i=1; i<len; i++)
    {
        if(arr[i] > *max)
            *max = arr[i];
        if(arr[i] < *min)
            *min = arr[i];
    }
}

int main() {
    int a[] = {23, 4, 21, 98, 987, 45, 32, 10, 123, 986, 50, 3, 4, 5};
    int min, max;
    int len = sizeof(a)/sizeof(a[0]);
    minMax(a, len, &min, &max);
    printf("Minimum value in the array is: %d and Maximum value is: %d", min, max);
    return 0;
}
```

Type:6 Returning pointers

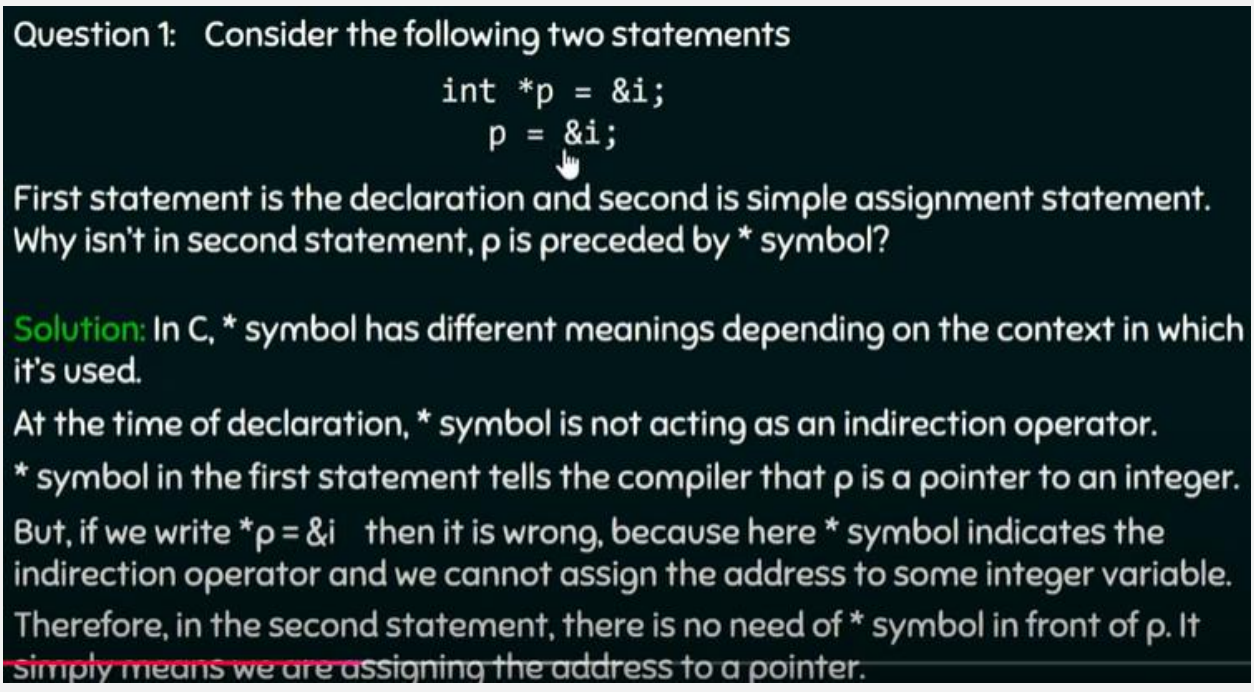
```c
int main()
{
    int a[] = {1, 2, 3, 4, 5};
    int n = sizeof(a)/sizeof(a[0]);
    int *mid = findMid(a, n);
    printf("%d", *mid);
    return 0;

}

int *findMid(int a[], int n)
{
    return &a[n/2];
}
```

OUTPUT: 3

Q:1



Type:7 Important question in pointers



Question 2: What is the output of the following program

```
void fun(const int *p)
{
    *p = 0;
}

int main() {
    const int i = 10;
    fun(&i);
    return 0;
}
```

Output:

Error: Assignment of read-only location *p

Question 3: How to print the address of a variable?

Solution: use %p as a format specifier in printf function

```
int main() {
    int i = 10;
    int *p = &i;
    printf("The address of variable i is %p", p);
    return 0;
}
```

```
Output: The address of variable i is 0x7ffd5b9a987c
```

Type:8 Arithmetic pointers-Addition