QUIZ 4

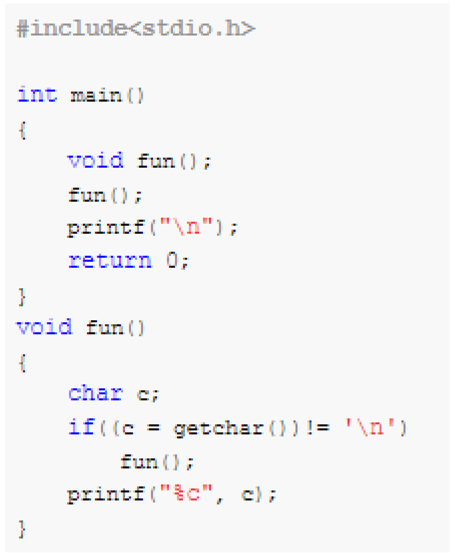
Prepared by : ENG. Marco Magdy

QUESTIONS

Q1)A FUNCTION CANNOT BE DEFINED INSIDE ANOTHER FUNCTION (TRUE OR FALSE)

- **Answer:** Option **A (True)**
- **Explanation:**
- A function cannot be defined inside the another function, but a function can be called inside a another function.

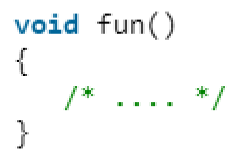
Q2) WHAT WILL BE THE OUTPUT OF THE PROGRAM IF CHARACTERS 'A', 'B' AND 'C' ENTER ARE SUPPLIED AS INPUT?



```c
#include<stdio.h>

int main()
{
    void fun();
    fun();
    printf("\n");
    return 0;
}
void fun()
{
    char c;
    if((c = getchar())!= '\n')
        fun();
    printf("%c", c);
}
```

- **Answer:** Option **D (CBA)**
- **Explanation:**
- **Step 1**: void fun(); This is the prototype for the function fun().
- **Step 2**: fun(); The function fun() is called here.
- The function fun() gets a character input and the input is terminated by an enter key(New line character). It prints the given character in the reverse order.
- The given input characters are "abc"

Q3)IN C, WHAT IS THE MEANING OF FOLLOWING FUNCTION PROTOTYPE WITH EMPTY PARAMETER LIST



```
void fun()
{
    /* .... */
}
```

- **Answer:** Option **B (Function can be called with any number of parameters of any types)**
- **Explanation:**
- Empty list in C mean that the parameter list is not specified and function can be called with any parameters. In C, to declare a function that can only be called without any parameter, we should use "void fun(void)"
- As a side note, in C++, empty list means function can only be called without any parameter. In C++, both void fun() and void fun(void) are same.

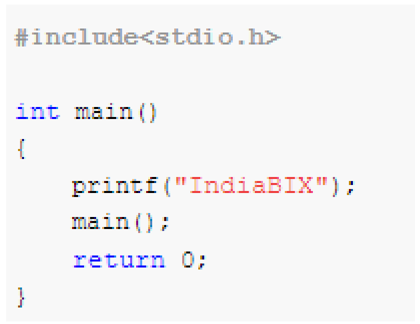
Q4)THE SIZEOF() OPERATOR IS EVALUATED AT

- **Answer:** Option **C (Run Time)**
- **Explanation:**
- As of C99, sizeof is evaluated at runtime .
- Example. int a[b], where b is not known at compile time. In this case, sizeof(a) is evaluated at runtime and its result is the size (in bytes) of the entire array, i.e. the size of all elements in the array, combined.

Q5)THE KEYWORD USED TO TRANSFER CONTROL FROM A FUNCTION BACK TO THE CALLING FUNCTION IS.......

- **Answer:** Option **D (return)**
- **Explanation:**
- The keyword *return* is used to transfer control from a function back to the calling function.

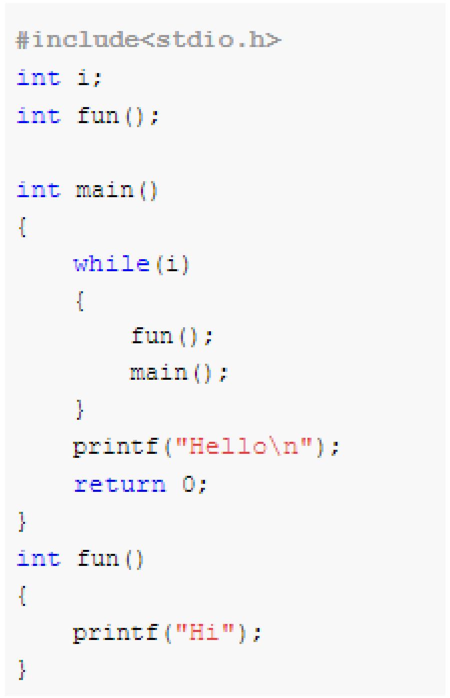
Q6)HOW MANY TIMES THE PROGRAM WILL PRINT "INDIABIX" ?



```
#include<stdio.h>

int main()
{
    printf("IndiaBIX");
    main();
    return 0;
}
```

- **Answer:** Option **D (Till stack overflows)**
- **Explanation:**
- A call stack or function stack is used for several related purposes, but the main reason for having one is to keep track of the point to which each active subroutine should return control When it finishes executing.
- Here function main() is called repeatedly and its return address is stored in the stack. After stack memory is full. It shows stack overflow error.
- A stack overflow occurs when too much memory is used on the call stack.

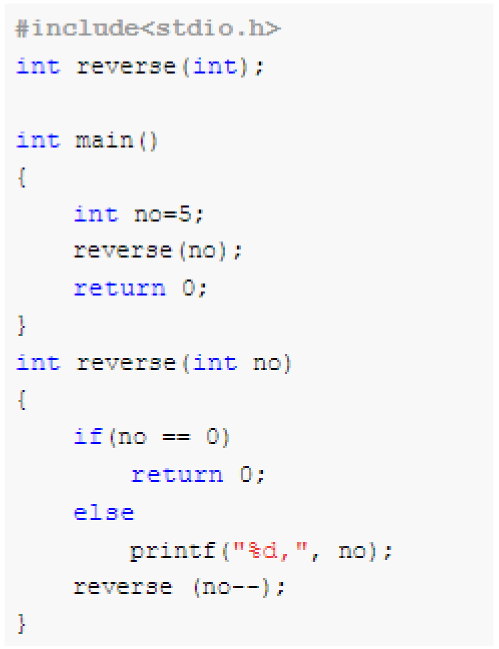
Q7)WHAT WILL BE THE OUTPUT OF THE PROGRAM?



```c
#include<stdio.h>
int i;
int fun();

int main()
{
    while(i)
    {
        fun();
        main();
    }
    printf("Hello\n");
    return 0;
}
int fun()
{
    printf("Hi");
}
```

- **Answer:** Option **A (Hello)**
- **Explanation:**
- Step 1: int **i**; The variable i is declared as an integer type.
- Step 2: **int fun();** This prototype tells the compiler that the function fun() does not accept any arguments and it returns an integer value.
- Step 3: **while(i)** The value of i is not initialized so this while condition is failed. So, it does not execute the while block.
- Step 4: **printf("Hello\n");** It prints "Hello".
- Hence the output of the program is "Hello".

Q8)WHAT WILL BE THE OUTPUT OF THE PROGRAM?

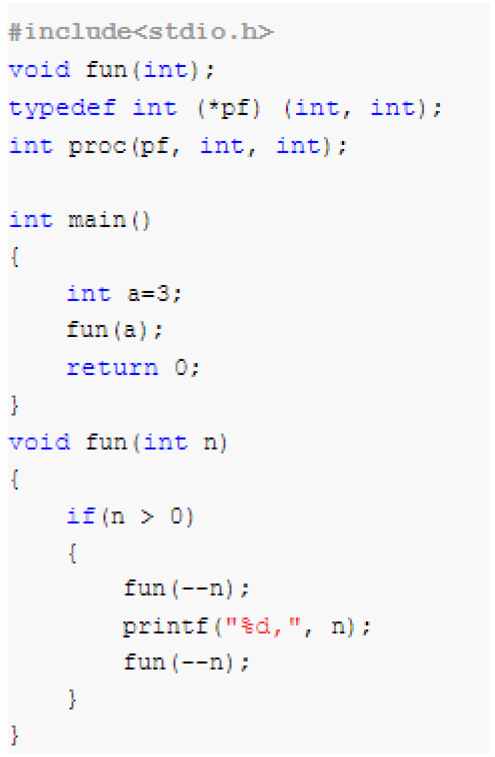


```
#include<stdio.h>
int reverse(int);

int main()
{
    int no=5;
    reverse(no);
    return 0;
}
int reverse(int no)
{
    if(no == 0)
        return 0;
    else
        printf("%d,", no);
    reverse (no--);
}
```

- **Answer:** Option **D (Infinite loop)**
- **Explanation:**
- **Step 1**: int no=5; The variable no is declared as integer type and initialized to 5.
- **Step 2**: reverse(no); becomes reverse(5); It calls the function reverse() with '5' as parameter.
- The function reverse accept an integer number 5 and it returns '0'(zero) if(5 == 0) if the given number is '0'(zero) or else printf("%d,", no); it prints that number 5 and calls the function reverse(5);.
- The function runs infinetely because the there is a post-decrement operator is used. It will not decrease the value of 'n' before calling the reverse() function. So, it calls reverse(5) infinitely.
- Note: If we use pre-decrement operator like reverse(--n), then the output will be 5, 4, 3, 2, 1. Because before calling the function, it decrements the value of 'n'.

Q9)WHAT WILL BE THE OUTPUT OF THE PROGRAM?

- **Answer:** Option **D (0, 1, 2, 0,)**
- **Explanation:**
- if(3>0) True. Step:1 fun(2).
- if(2>0) True. Step:2 fun(1).
- if(1>0) True. Step:3 fun(0).
- if(0>0) False.
- Print 0 from Step:3, fun(-1) False.
- Print 1 from Step:2, fun(0) False.
- Print 2 from Step:1, fun(1)->fun(0).
- Print 0 from fun (0) at second last line.

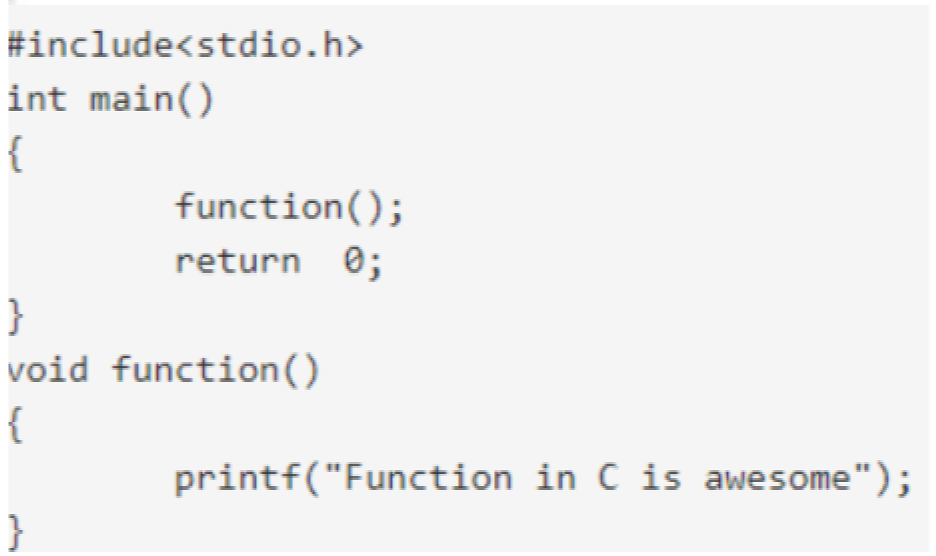


```
#include<stdio.h>
void fun(int);
typedef int (*pf) (int, int);
int proc(pf, int, int);

int main()
{
    int a=3;
    fun(a);
    return 0;
}
void fun(int n)
{
    if(n > 0)
    {
        fun(--n);
        printf("%d,", n);
        fun(--n);
    }
}
```

Q10)IF RETURN TYPE FOR A FUNCTION IS NOT SPECIFIED, IT DEFAULTS TO INT (TRUE OR FALSE)

- **Answer:** Option **A (True)**
- **Explanation:**
- True, The default return type for a function is int.

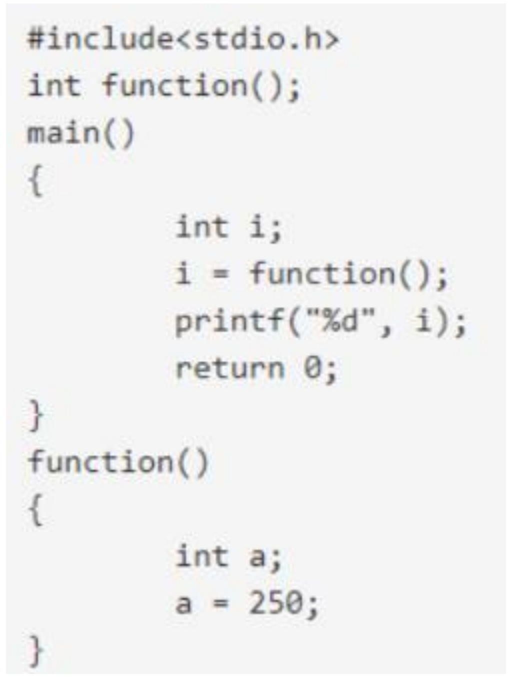
Q11)What will be the output of the C program?



```
#include<stdio.h>
int main()
{
        function();
        return  0;
}
void function()
{
        printf("Function in C is awesome");
}
```

- **Answer:** Option **D (Compilation error)**
- **Explanation:**
- Function definition should be a top of main() or the prototype should be above the main

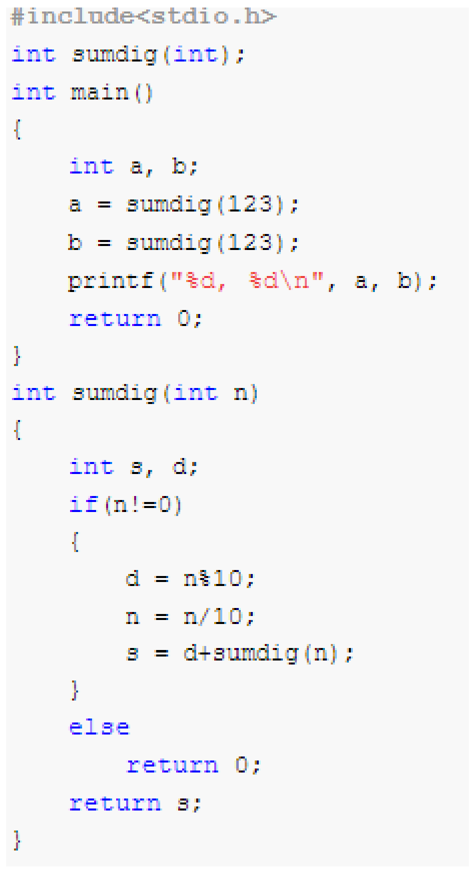
Q12)WHAT WILL BE THE OUTPUT OF THE C PROGRAM?



```
#include<stdio.h>
int function();
main()
{
        int i;
        i = function();
        printf("%d", i);
        return 0;
}
function()
{
        int a;
        a = 250;
}
```

- **Answer:** Option **C (1)**
- **Explanation:**
- Function executed successfully and we don't return anything, by default C compiler returns 1 for its successful execution.

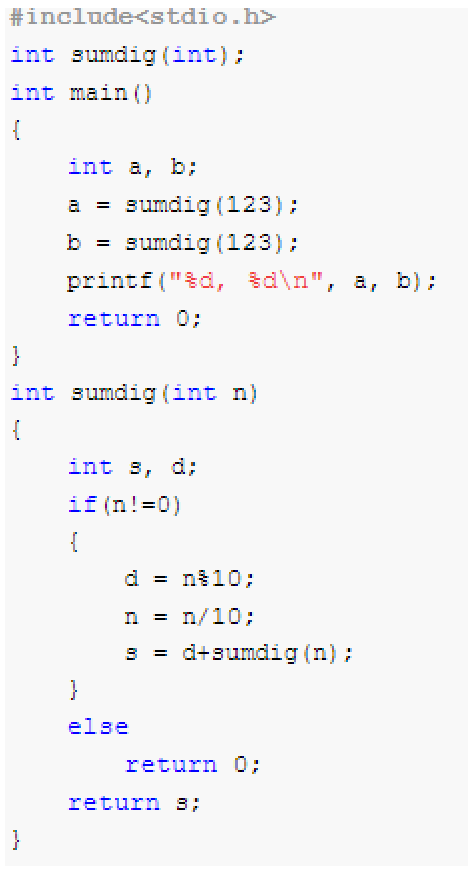
Q13)WHAT WILL BE THE OUTPUT OF THE C PROGRAM?



```c
#include<stdio.h>
int sumdig(int);
int main()
{
    int a, b;
    a = sumdig(123);
    b = sumdig(123);
    printf("%d, %d\n", a, b);
    return 0;
}
int sumdig(int n)
{
    int s, d;
    if(n!=0)
    {
        d = n%10;
        n = n/10;
        s = d+sumdig(n);
    }
    else
        return 0;
    return s;
}
```

- **Answer:** Option **C (6,6)**
- **Explanation:**
- **Step 1 :** call to function sumdig with parameter 123,now n=123
- **Step 2 :** function starts
- **Step 3 :** declaration of s & d and contains GARBAGE value
- **Step 4 :** 123!=0 condition true, loop starts
- **Step 5 :** d=123%10, d=3
- **Step 6 :** n=123/10, n=12
- **Step 7 :** s=d+sumdig(12),call to function sumdig with parameter 12, n=12 & s=3+sumdig(12)
- **Step 8 :** Step 1 with value of n=12
- **Step 9 :** finally s=3+2+1=6
- **Step 10 :** now (n!=0) condition false come out of loop and return value 6
- **Step 11 :** same execution for b=sumdig(123)

Q13)



```
#include<stdio.h>
int sumdig(int);
int main()
{
    int a, b;
    a = sumdig(123);
    b = sumdig(123);
    printf("%d, %d\n", a, b);
    return 0;
}
int sumdig(int n)
{
    int s, d;
    if(n!=0)
    {
        d = n%10;
        n = n/10;
        s = d+sumdig(n);
    }
    else
        return 0;
    return s;
}
```

1st time

- D = 3
- N = 12
- S = 3+sumdig(n)

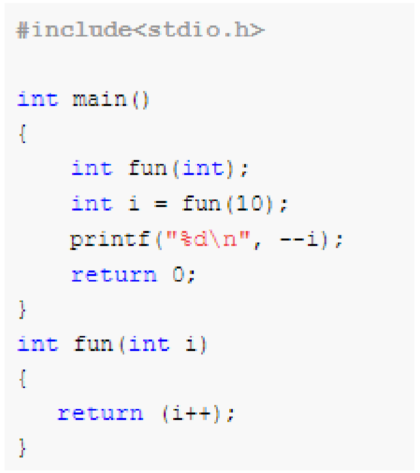
2nd time

- D = 2
- N = 1
- S = 2+sumdig(n)

3rd time

- D = 1
- N = 0
- S = 1+0=1

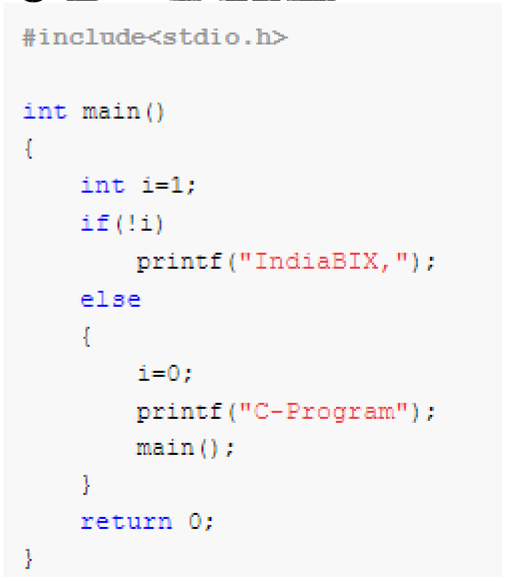
Q14)WHAT WILL BE THE OUTPUT OF THE C PROGRAM?



```
#include<stdio.h>

int main()
{
    int fun(int);
    int i = fun(10);
    printf("%d\n", --i);
    return 0;
}
int fun(int i)
{
   return (i++);
}
```

- **Answer:** Option **A (9)**
- **Explanation:**
- **Step 1**: int fun(int); Here we declare the prototype of the function fun().
- **Step 2**: int i = fun(10); The variable i is declared as an integer type and the result of the fun(10) will be stored in the variable i.
- **Step 3**: int fun(int i){ return (i++); } Inside the fun() we are returning a value return(i++). It returns 10. because i++ is the post-increment operator.
- **Step 4**: Then the control back to the main function and the value 10 is assigned to variable i.
- **Step 5**: printf("%d\n", --i); Here --i denoted pre-increment. Hence it prints the value 9.

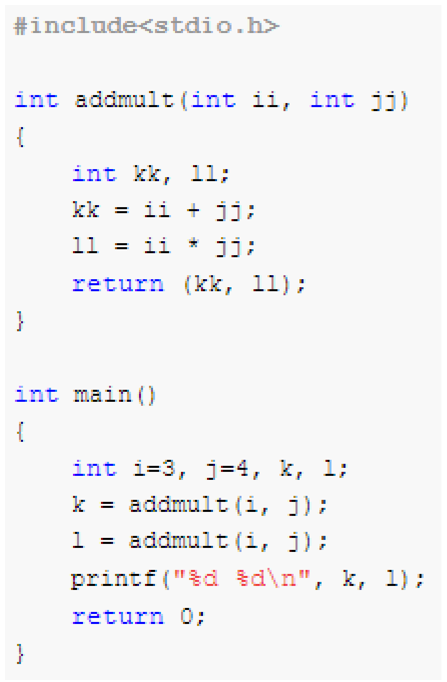
Q15)WHAT WILL BE THE OUTPUT OF THE PROGRAM?



```
#include<stdio.h>

int main()
{
    int i=1;
    if(!i)
        printf("IndiaBIX,");
    else
    {
        i=0;
        printf("C-Program");
        main();
    }
    return 0;
}
```

- **Answer:** Option **B (prints "C-Program" infinitely)**
- **Explanation:**
- **Step 1**: int i=1; The variable i is declared as an integer type and initialized to 1(one).
- **Step 2**: if(!i) Here the !(NOT) operator reverts the i value 1 to 0. Hence the if(0) condition fails. So it goes to else part.
- **Step 3**: else { i=0; In the else part variable i is assigned to value 0(zero).
- **Step 4**: printf("C-Program"); It prints the "C-program".
- **Step 5**: main(); Here we are calling the main() function.
- After calling the function, the program repeats from **Step 1** to **Step 5** infinitely.

Q16)WHAT WILL BE THE OUTPUT OF THE PROGRAM?

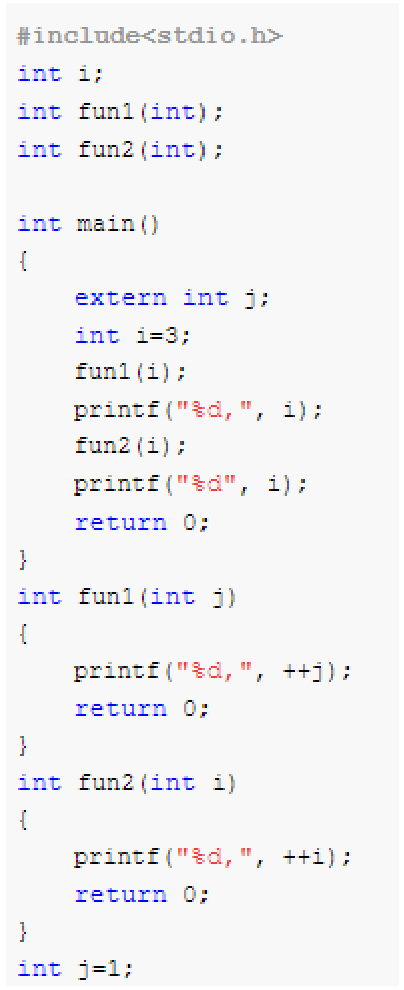


```
#include<stdio.h>

int addmult(int ii, int jj)
{
    int kk, ll;
    kk = ii + jj;
    ll = ii * jj;
    return (kk, ll);
}

int main()
{
    int i=3, j=4, k, l;
    k = addmult(i, j);
    l = addmult(i, j);
    printf("%d %d\n", k, l);
    return 0;
}
```

- **Answer:** Option **A (12,12)**
- **Explanation:**
- **Step 1**: int i = 3,j=4,k,l; The variable i is declared as an integer type and its value = 3 , and j = 4 , k and l = garbage value.
- **Step 2**: k = addmult(i,j); The variable k is declared as an integer type and the result of the addmult(i,j) will be stored in the variable k.
- **Step 3**: int addmult(int ii,int jj); { int kk,ll; Inside the int addmult() The variable kk,ll are declared as an integer type.
- **Step 4**: kk = ii + jj; ll = ii * jj; The variable kk = 3+4 =7 and ll= 3*4 =12.
- **Step 5**: Then the control back to the main function and the value 12 is assigned to variable ll and the left operand has no effect.
- **Step 6**: printf("%d %d\n", k, l); Hence it prints the value 12,12.

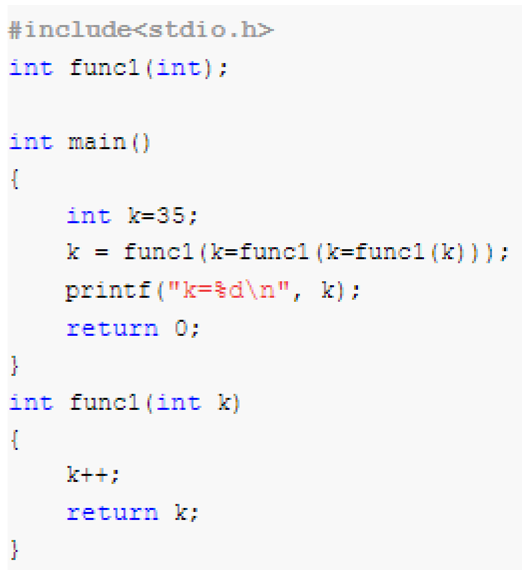
Q17)What will be the output of the program?

- **Answer:** Option **B (4, 3, 4, 3)**
- **Explanation:**
- **Step 1**: int i; The variable i is declared as an global and integer type.
- **Step 2**: int fun1(int); This prototype tells the compiler that the fun1() accepts the one integer parameter and returns the integer value.
- **Step 3**: int fun2(int); This prototype tells the compiler that the fun2() accepts the one integer parameter and returns the integer value.
- **Step 4**: extern int j; Inside the main function, the extern variable j is declared and defined in another source file.
- **Step 5**: int i=3; The local variable i is defined as an integer type and initialized to 3.
- **Step 6**: fun1(i); The fun1(i) increments the given value of variable i prints it. Here fun1(i) becomes fun1(3) hence it prints '4' then the control is given back to the main function.
- **Step 7**: printf("%d,", i); It prints the value of local variable i. So, it prints '3'.
- **Step 8**: fun2(i); The fun2(i) increments the given value of variable i prints it. Here fun2(i) becomes fun2(3) hence it prints '4' then the control is given back to the main function.
- **Step 9**: printf("%d,", i); It prints the value of local variable i. So, it prints '3'.
- Hence the output is "4 3 4 3".



```
#include<stdio.h>
int i;
int fun1(int);
int fun2(int);

int main()
{
    extern int j;
    int i=3;
    fun1(i);
    printf("%d,", i);
    fun2(i);
    printf("%d", i);
    return 0;
}
int fun1(int j)
{
    printf("%d,", ++j);
    return 0;
}
int fun2(int i)
{
    printf("%d,", ++i);
    return 0;
}
int j=1;
```

Q18)WHAT WILL BE THE OUTPUT OF THE PROGRAM?



```
#include<stdio.h>
int func1(int);

int main()
{
    int k=35;
    k = func1(k=func1(k=func1(k)));
    printf("k=%d\n", k);
    return 0;
}
int func1(int k)
{
    k++;
    return k;
}
```

- **Answer:** Option **D (K=38)**
- **Explanation:**
- **Step 1**: int k=35; The variable k is declared as an integer type and initialized to 35.
- **Step 2**: k = func1(k=func1(k=func1(k))); The func1(k) increment the value of k by 1 and return it. Here the func1(k) is called 3 times.
  Hence it increments value of k = 35 to 38. The result is stored in the variable k = 38.
- **Step 3**: printf("k=%d\n", k); It prints the value of variable k "38".

Q18)

1st time

- k = 35 in main
- k = 36 return of function

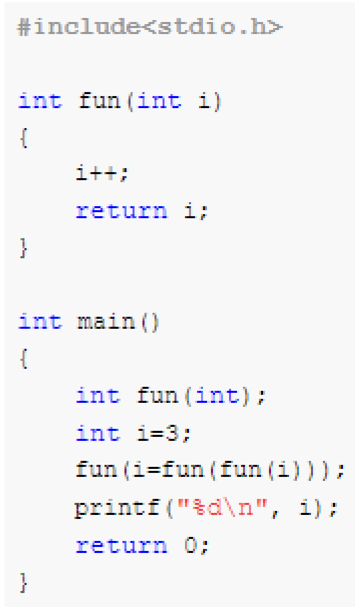
2nd time

- k = 36 in main
- k = 37 return of function

3rd time

- k = 37 in main
- k = 38 return of function

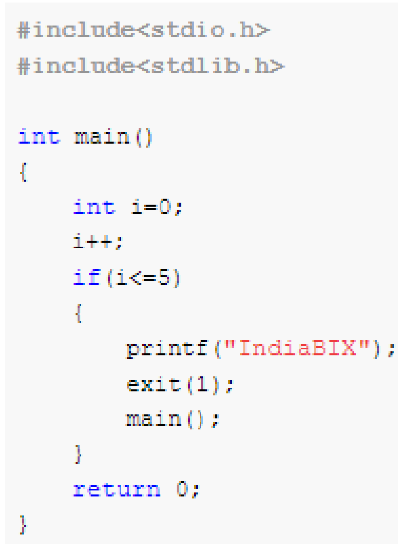
Q19)WHAT WILL BE THE OUTPUT OF THE PROGRAM?



```
#include<stdio.h>

int fun(int i)
{
    i++;
    return i;
}

int main()
{
    int fun(int);
    int i=3;
    fun(i=fun(fun(i)));
    printf("%d\n", i);
    return 0;
}
```

- **Answer:** Option **A (5)**
- **Explanation:**
- **Step 1**: int fun(int); This is prototype of function fun(). It tells the compiler that the function fun() accept one integer parameter and returns an integer value.
- **Step 2**: int i=3; The variable i is declared as an integer type and initialized to value 3.
- **Step 3**: fun(i=fun(fun(i)));. The function fun(i) increments the value of i by 1(one) and return it.
- Lets go Step by Step ,
- => fun(i) becomes fun(3) is called and it returns 4.
- => i = fun(fun(i)) becomes i = fun(4) is called and it returns 5 and stored in variable i.(i=5)
- => fun(i=fun(fun(i))); becomes fun(5); is called and it return 6 and nowhere the return value is stored.
- **Step 4**: printf("%d\n", i); It prints the value of variable i.(6)
- Hence the output is '5'.

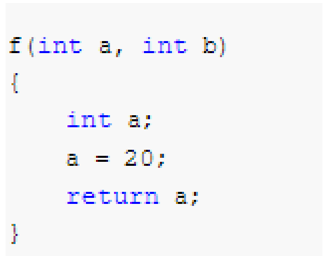
Q20)WHAT WILL BE THE OUTPUT OF THE PROGRAM?



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

int main()
{
    int i=0;
    i++;
    if(i<=5)
    {
        printf("IndiaBIX");
        exit(1);
        main();
    }
    return 0;
}
```

- **Answer:** Option **D (IndiaBIx)**
- **Explanation:**
- **Step 1**: int i=0; The variable i is declared as in integer type and initialized to '0'(zero).
- **Step 2**: i++; Here variable i is incremented by 1. Hence i becomes '1'(one).
- **Step 3**: if(i<=5) becomes if(1 <=5). Hence the if condition is satisfied and it enter into if block statements.
- **Step 4**: printf("IndiaBIX"); It prints "IndiaBIX".
- **Step 5**: exit(1); This exit statement terminates the program execution.

Q21)Point out the error in the program



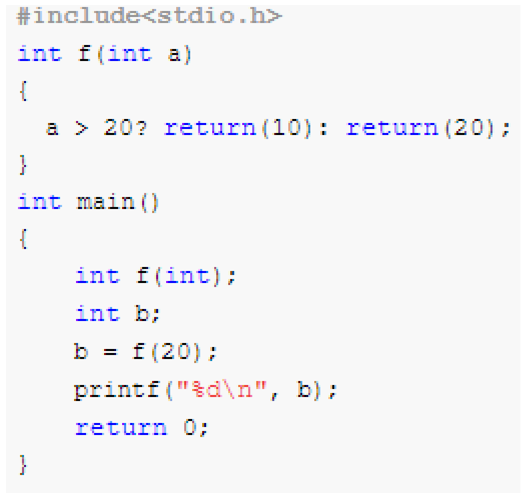
```
f(int a, int b)
{
    int a;
    a = 20;
    return a;
}
```

- **Answer:** Option **C (Redeclaration of a)**
- **Explanation:**
- f(int a, int b) The variable a is declared in the function argument statement.
- int a; Here again we are declaring the variable a. Hence it shows the error "Redeclaration of a“
- Function f is implicit by int

Q22) CAN ARRAY PASS BY VALUE

- **Answer:** Option **A (No, Arrays are pointers)**
- **Explanation:**
- In C, array name represents address and when we pass an array, we actually pass address and the parameter receiving function always accepts them as pointers (even if we use []).

Q24)POINT OUT THE ERROR IN THE PROGRAM

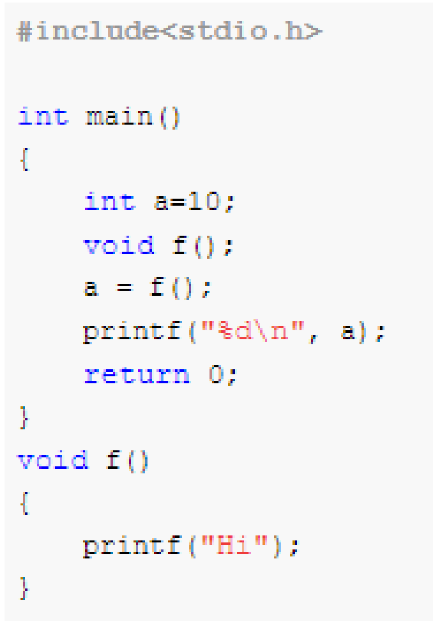


```
#include<stdio.h>
int f(int a)
{
  a > 20? return(10): return(20);
}
int main()
{
    int f(int);
    int b;
    b = f(20);
    printf("%d\n", b);
    return 0;
}
```

- **Answer:** Option **C (Error: return statement cannot be used with conditional operators)**
- **Explanation:**
- In a ternary operator, we cannot use the return statement. The ternary operator requires expressions but not code.

```
#include<stdio.h>
int f(int a){
if (a > 20)
      return 10;
else
      return 20;
}


int main()
{
int f(int);
int b;
b = f(20);
printf("%d\n", b);
return 0;
}
```

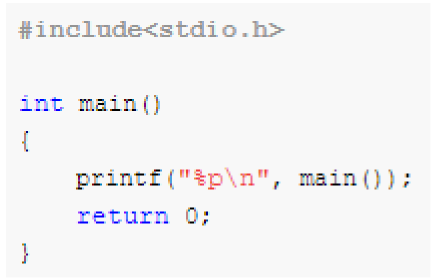
Q25)POINT OUT THE ERROR IN THE PROGRAM



```
#include<stdio.h>

int main()
{
    int a=10;
    void f();
    a = f();
    printf("%d\n", a);
    return 0;
}
void f()
{
    printf("Hi");
}
```

- **Answer:** Option **A (Error: Not allowed assignment)**
- **Explanation:**
- The function void f() doesn’t return any value so we can’t define a as return of this function .

Q26) WHICH OF THE FOLLOWING STATEMENTS ARE CORRECT ABOUT THE PROGRAM?

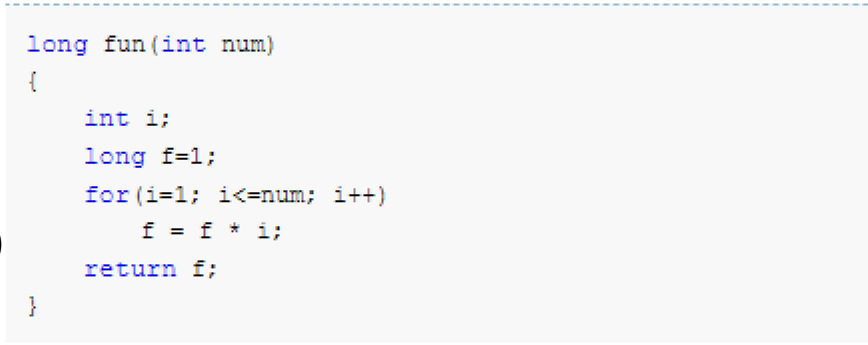


```
#include<stdio.h>

int main()
{
    printf("%p\n", main());
    return 0;
}
```

- **Answer:** Option **B (Runs infinitely without printing anything)**
- **Explanation:**
- In printf("%p\n", main ( )) ; it calls the main() function and then it repeats infinitely, until stack overflow
- To Print address of pointer of the main() function.

```
#include <stdio.h>
int main()
{
printf("%p", main);
return 0;
}
```

Q27)WHICH OF THE FOLLOWING STATEMENTS ARE CORRECT ABOUT THIS FUNCTION



```
long fun(int num)
{
    int i;
    long f=1;
    for(i=1; i<=num; i++)
        f = f * i;
    return f;
}
```

- **Answer:** Option **C (Calculate factorial value of integer)**
- **Explanation:**
- Suppose the user entered 6.
- Initially, fun() is called from main() with 6 passed as an argument.
- Then in for loop it will multiply the value of f to i in each iteration.
- Finally it will return the factorial of the number (f).

Q28)FUNCTIONS CANNOT RETURN MORE THAN ONE VALUE AT A TIME (TRUE OR FALSE)

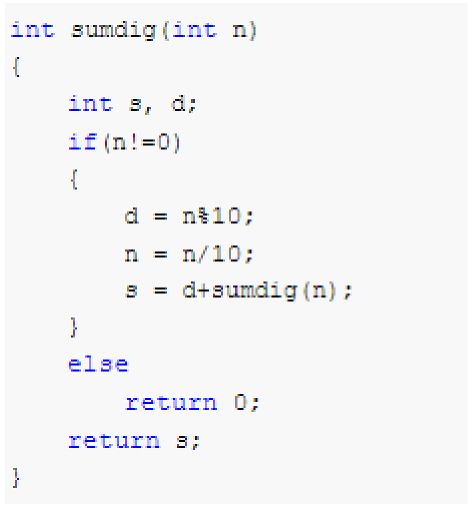
- **Answer:** Option **A (True)**
- **Explanation:**
- True, A function cannot return more than one value at a time. because after returning a value the control is given back to calling function.

Q29) FUNCTIONS CAN BE CALLED EITHER BY VALUE OR REFERENCE ? (TRUE OR FALSE)

- **Answer:** Option **A (True)**
- **Explanation:**
- Call by value means c = sub(a, b); here value of a and b are passed.
- Call by reference means c = sub(&a, &b); here address of a and b are passed.

Q30) A FUNCTION MAY HAVE ANY NUMBER OF RETURN STATEMENTS EACH RETURNING DIFFERENT VALUES (TRUE OR FALSE)

- **Answer:** Option **A (True)**
- **Explanation:**
- **Example :**



```
int sumdig(int n)
{
    int s, d;
    if(n!=0)
    {
        d = n%10;
        n = n/10;
        s = d+sumdig(n);
    }
    else
        return 0;
    return s;
}
```

Q31)NAMES OF FUNCTIONS IN TWO DIFFERENT FILES LINKED TOGETHER MUST BE UNIQUE (TRUE OR FALSE)

- **Answer:** Option **A (True)**
- **Explanation:**
- True, If two function are declared in a same name, it gives "Error: Multiple declaration of function_name())".

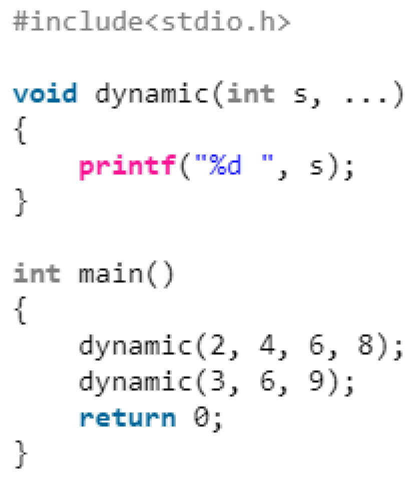
Q32)USUALLY RECURSION WORKS SLOWER THAN LOOPS. (TRUE OR FALSE)

- **Answer:** Option **A (True)**
- **Explanation:**
- When a recursive call is made, the function/process clones itself and then process that function. This leads to time and space constrains.
- In a loop, there is no recursive call involved that saves a lot of time and space too.

Q33)WHICH OF THE FOLLOWING IS TRUE ABOUT RETURN TYPE OF FUNCTIONS IN C?

- **Answer:** Option **B (Functions can return any type except array and functions)**
- **Explanation:**
- In C, functions can return any type except arrays and functions.
- We can get around this limitation by returning pointer to array or pointer to function.

Q34)OUTPUT OF FOLLOWING PROGRAM?

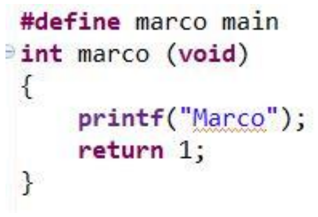


```c
#include<stdio.h>

void dynamic(int s, ...)
{
    printf("%d ", s);
}

int main()
{
    dynamic(2, 4, 6, 8);
    dynamic(3, 6, 9);
    return 0;
}
```

- **Answer:** Option **A (2,3)**
- **Explanation:**
- In c three continuous dots is known as ellipsis which is variable number of arguments of function. The values to parameters are assigned one by one.

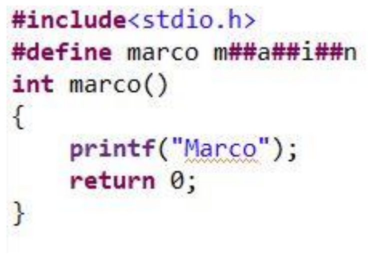
Q35) CAN A PROGRAM BE COMPILED WITHOUT A MAIN FUNCTION ? (TRUE OR FALSE)

Using a macro that defines main



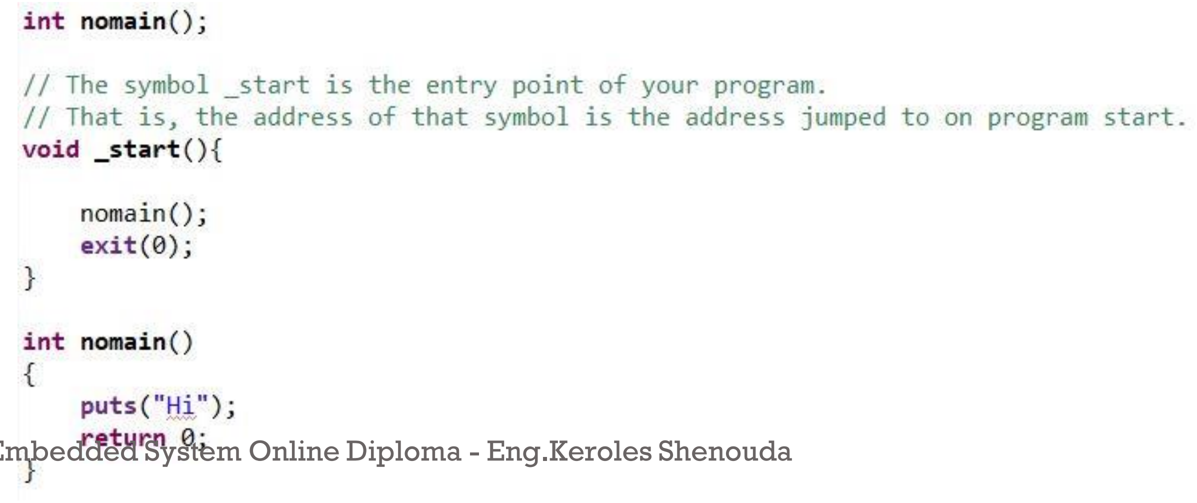
```c
#define marco main
int marco (void)
{
    printf("Marco");
    return 1;
}
```

Using Token-Pasting Operator



```c
#include<stdio.h>
#define marco m##a##i##n
int marco()
{
    printf("Marco");
    return 0;
}
```

Modify the entry point during compilation



```c
int nomain();

// The symbol _start is the entry point of your program.
// That is, the address of that symbol is the address jumped to on program start.
void _start(){

    nomain();
    exit(0);
}

int nomain()
{
    puts("Hi");
    return 0;
}
```

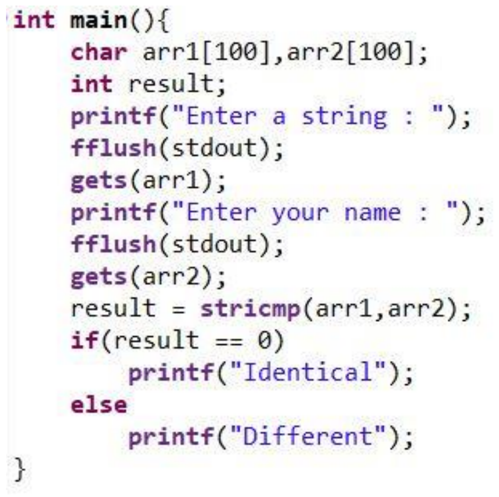
VOID MAIN() {… ; MAIN();} (WHAT HAPPEN)

- You must be thinking that when the main() function calls inside main(), recursion goes on infinitely.

CODES

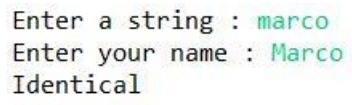
Q1)WRITE A C PROGRAM TAKES STRING FROM THE USER AND CHECK IF IT THE SAME USERNAME OR NOT.

Main Function



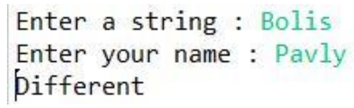
```
int main(){
    char arr1[100],arr2[100];
    int result;
    printf("Enter a string : ");
    fflush(stdout);
    gets(arr1);
    printf("Enter your name : ");
    fflush(stdout);
    gets(arr2);
    result = stricmp(arr1,arr2);
    if(result == 0)
        printf("Identical");
    else
        printf("Different");
}
```

Output if Identical



```
Enter a string : marco
Enter your name : Marco
Identical
```

Output if Different



```
Enter a string : Bolis
Enter your name : Pavly
Different
```

Q2)WRITE A C PROGRAM FOR SWAPPING 2 ARRAYS WITH DIFFERENT LENGHTS

Main Function

```c
int main() {
    int a[10],b[10],c[10],i;
    printf("Enter First array->");
    fflush(stdout);
    for (i=0;i<10;i++)
        scanf("%d",&a[i]);
    printf("\nEnter Second array->");
    fflush(stdout);
    for (i=0;i<5;i++)
        scanf("%d",&b[i]);
    printf("Arrays before swapping\nFirst array->\t");
    fflush(stdout);
    for (i=0;i<10;i++) {
        printf("\t%d",a[i]);
        fflush(stdout);
    }
    printf("\nSecond array->\t");
    fflush(stdout);
    for (i=0;i<5;i++) {
        printf("\t%d",b[i]);
        fflush(stdout);
    }
    for (i=0;i<10;i++) {
        c[i]=a[i];
        a[i]=b[i];
        b[i]=c[i];
    }
```

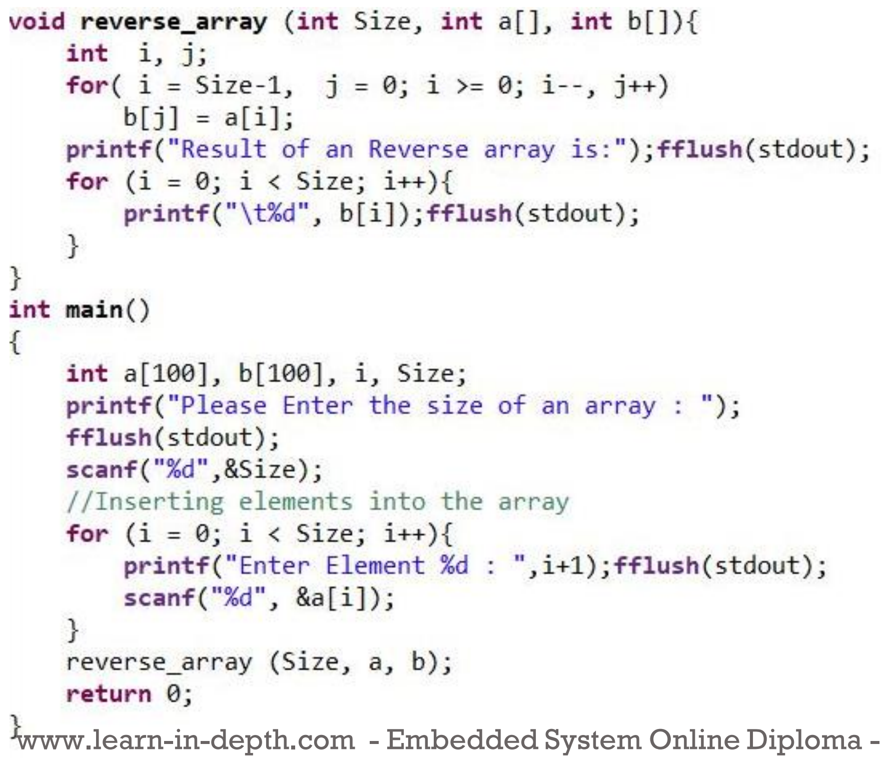
```c
    printf("\nArrays after swapping");
    fflush(stdout);
    printf("\nFirst array->\t"); fflush(stdout);
    for (i=0;i<5;i++) {
        printf("\t%d",a[i]);fflush(stdout);
    }
    printf("\nSecond array->\t");fflush(stdout);
    for (i=0;i<10;i++) {
        printf("\t%d",b[i]);fflush(stdout);
    }
    return 0;
}
```

Output

```
Arrays before swapping
First array->   1   2   3   4   5   6   7   8   9   10
Second array->  1   8   9   7   5
Arrays after swapping
First array->   1   8   9   7   5
Second array->  1   2   3   4   5   6   7   8   9   10
```

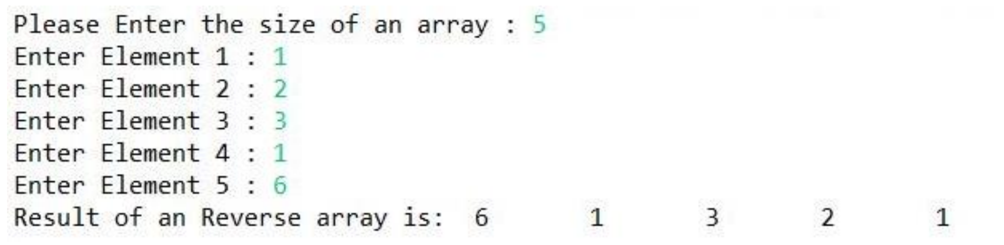
Q3)WRITE A C FUNCTION THAT REVERSE AN INPUT ARRAY

Main Function



```c
void reverse_array (int Size, int a[], int b[]){
    int  i, j;
    for( i = Size-1,  j = 0; i >= 0; i--, j++)
        b[j] = a[i];
    printf("Result of an Reverse array is:");fflush(stdout);
    for (i = 0; i < Size; i++){
        printf("\t%d", b[i]);fflush(stdout);
    }
}
int main()
{
    int a[100], b[100], i, Size;
    printf("Please Enter the size of an array : ");
    fflush(stdout);
    scanf("%d",&Size);
    //Inserting elements into the array
    for (i = 0; i < Size; i++){
        printf("Enter Element %d : ",i+1);fflush(stdout);
        scanf("%d", &a[i]);
    }
    reverse_array (Size, a, b);
    return 0;
}
```

Output



```
Please Enter the size of an array : 5
Enter Element 1 : 1
Enter Element 2 : 2
Enter Element 3 : 3
Enter Element 4 : 1
Enter Element 5 : 6
Result of an Reverse array is:  6       1       3       2       1
```

Q5)WRITE A C FUCNTION THAT CLEARS BIT IN A GIVEN NUMBER

Main Function

```
int main() {
    int n,bit;
    printf("Input Number : ");
    fflush(stdout);
    scanf("%d",&n);
    printf("Bit Position : ");
    fflush(stdout);
    scanf("%d",&bit);
    printf("Result =  %d",clear_specifed_bit(n,bit));
}
```

Clear Bit Function

```
int clear_specifed_bit(int n,int bit){

    return n &= ~(1 << bit);
}
```

Explanation

```
3 => 11
1 << 0 =          0000 0001 = 1
~ 1 =             1111 1110
3 & ~ (1)
0000 0011 & 1111 1110 = 0000 0010 = 2
```

```
30 => 0001 1110
1 << 3 =          0000 1000 = 8
~ 0000 1000 =     1111 0111
30 & ~(8)
0001 1110 & 1111 0111 = 0001 0110 = 22
```

Output if Bit is Found

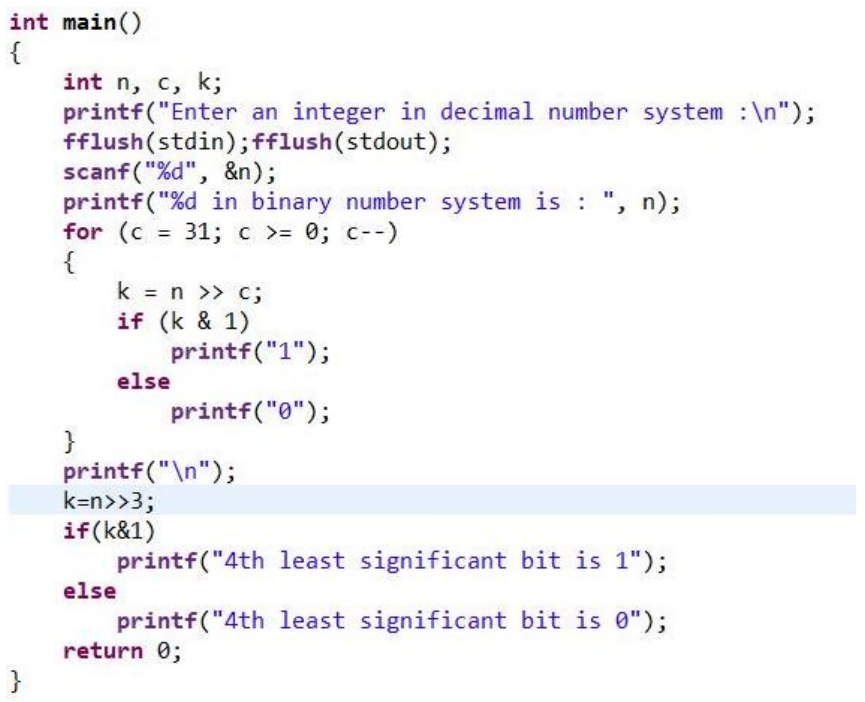
```
Input Number : 3
Bit Position : 0
Result =  2
```

Output if Bit isn't Found

```
Input Number : 3
Bit Position : 9
Result =  3
```

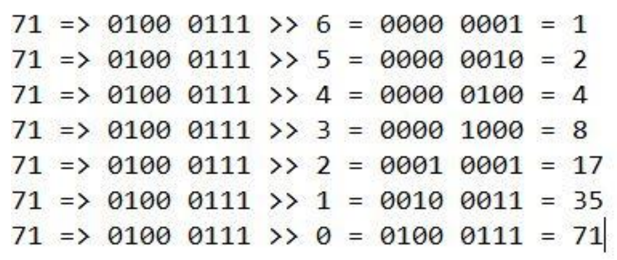
Q6)WRITE A C PROGRAM TO KNOW THE VALUE OF THE 4TH LEAST SIGNIFICANT BIT IN BINARY

Main Function



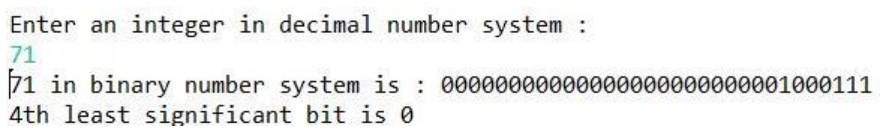
```c
int main()
{
    int n, c, k;
    printf("Enter an integer in decimal number system :\n");
    fflush(stdin);fflush(stdout);
    scanf("%d", &n);
    printf("%d in binary number system is : ", n);
    for (c = 31; c >= 0; c--)
    {
        k = n >> c;
        if (k & 1)
            printf("1");
        else
            printf("0");
    }
    printf("\n");
    k=n>>3;
    if(k&1)
        printf("4th least significant bit is 1");
    else
        printf("4th least significant bit is 0");
    return 0;
}
```

Explanation



```
71 => 0100 0111 >> 6 = 0000 0001 = 1
71 => 0100 0111 >> 5 = 0000 0010 = 2
71 => 0100 0111 >> 4 = 0000 0100 = 4
71 => 0100 0111 >> 3 = 0000 1000 = 8
71 => 0100 0111 >> 2 = 0001 0001 = 17
71 => 0100 0111 >> 1 = 0010 0011 = 35
71 => 0100 0111 >> 0 = 0100 0111 = 71
```

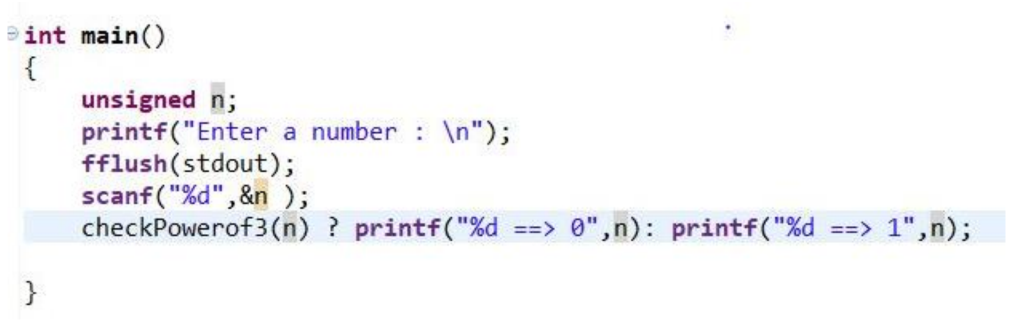
Output



```
Enter an integer in decimal number system :
71
71 in binary number system is : 00000000000000000000000001000111
4th least significant bit is 0
```

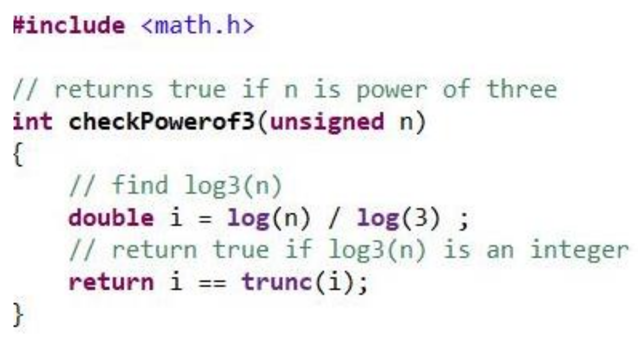
Q7)Write a C program to check if a given number is power of 3

Main Function



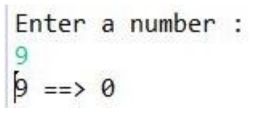
```
int main()
{
    unsigned n;
    printf("Enter a number : \n");
    fflush(stdout);
    scanf("%d",&n );
    checkPowerof3(n) ? printf("%d ==> 0",n): printf("%d ==> 1",n);

}
```

Check Power of 3 Function



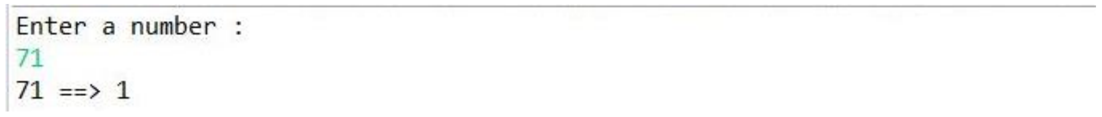
```
#include <math.h>

// returns true if n is power of three
int checkPowerof3(unsigned n)
{
    // find log3(n)
    double i = log(n) / log(3) ;
    // return true if log3(n) is an integer
    return i == trunc(i);
}
```

Output When True



```
Enter a number :
9
9 ==> 0
```

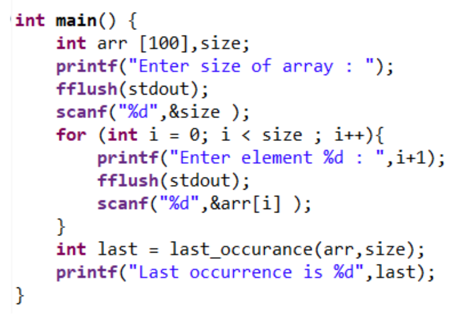
Output When False



```
Enter a number :
71
71 ==> 1
```

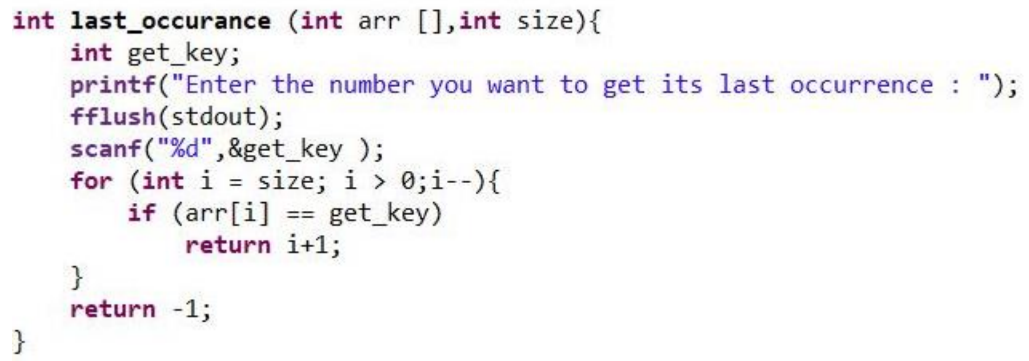
Q8) WRITE A C FUCNTION TO PRINT THE LAST OCCURANCE OF A NUMBER

Main Function



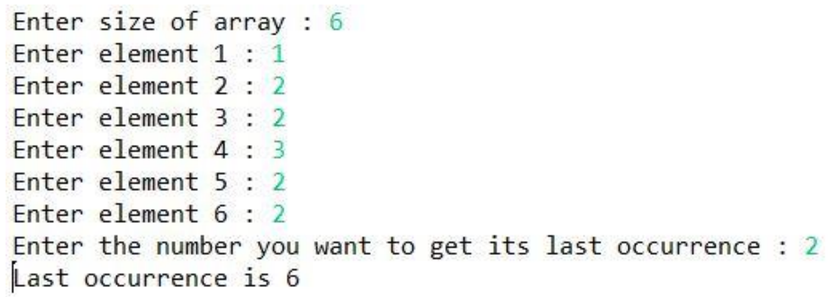
```c
int main() {
    int arr [100],size;
    printf("Enter size of array : ");
    fflush(stdout);
    scanf("%d",&size );
    for (int i = 0; i < size ; i++){
        printf("Enter element %d : ",i+1);
        fflush(stdout);
        scanf("%d",&arr[i] );
    }
    int last = last_occurance(arr,size);
    printf("Last occurrence is %d",last);
}
```

Last Occurrence Function



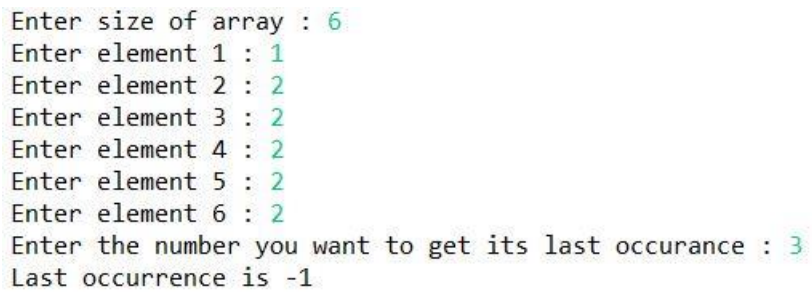
```c
int last_occurance (int arr [],int size){
    int get_key;
    printf("Enter the number you want to get its last occurrence : ");
    fflush(stdout);
    scanf("%d",&get_key );
    for (int i = size; i > 0;i--){
        if (arr[i] == get_key)
            return i+1;
    }
    return -1;
}
```

Output If Found



```
Enter size of array : 6
Enter element 1 : 1
Enter element 2 : 2
Enter element 3 : 2
Enter element 4 : 3
Enter element 5 : 2
Enter element 6 : 2
Enter the number you want to get its last occurrence : 2
Last occurrence is 6
```

Output If Not Found



```
Enter size of array : 6
Enter element 1 : 1
Enter element 2 : 2
Enter element 3 : 2
Enter element 4 : 2
Enter element 5 : 2
Enter element 6 : 2
Enter the number you want to get its last occurance : 3
Last occurrence is -1
```

ANY QUESTIONS ?

THANK YOU