Notes

# Question:

Output of following program?

#include <stdio.h>

int main()

{

int i = 5;

printf("%d %d %d", i++, i++, i++);

return 0;

}

A 7 6 5

B 5 6 7

C 7 7 7

D Compiler Dependent

Explanation:

When parameters are passed to a function, the value of every parameter is evaluated before being passed to the function. What is the order of evaluation of parameters - left-to-right or right-to-left? If evaluation order is left-to-right, then output should be 5 6 7 and if the evaluation order is right-to-left, then output should be 7 6 5. Unfortunately, there is no fixed order defined by C standard. A compiler may choose to evaluate either from left-to-right. So the output is compiler dependent.

# Question:

#include <stdio.h>

int main()

{

printf("%d", main);

return 0;

}

A Address of main function

B Compiler Error

C Runtime Error

D Some random value

Explanation: Name of the function is actually a pointer variable to the function and prints the address of the function. Symbol table is implemented like this.

struct

{

char \*name;

int (\*funcptr)();

}

symtab[] = {

"func", func,

"anotherfunc", anotherfunc,

};

# Question:

Output?

#include <stdio.h>

int main()

{

int (\*ptr)(int ) = fun;

(\*ptr)(3);

return 0;

}

int fun(int n)

{

for(;n > 0; n--)

printf("GeeksQuiz ");

return 0;

}

A GeeksQuiz GeeksQuiz GeeksQuiz

B GeeksQuiz GeeksQuiz

C Compiler Error

D Runtime Error

Explanation: The only problem with program is fun is not declared/defined before it is assigned to ptr.

# Question:

What is the meaning of using extern before function declaration? For example following function sum is made extern

extern int sum(int x, int y, int z)

{

return (x + y + z);

}

A Function is made globally available

B extern means nothing, sum() is same without extern keyword.

C Function need not to be declared before its use

D Function is made local to the file.

Explanation:

extern keyword is used for global variables. Functions are global anyways, so adding extern doesn't add anything.

# Question:

What is the meaning of using static before function declaration? For example following function sum is made static

static int sum(int x, int y, int z)

{

return (x + y + z);

}

A Static means nothing, sum() is same without static keyword.

B Function need not to be declared before its use

C Access to static functions is restricted to the file where they are declared

D Static functions are made inline

Explanation:

In C, functions are global by default. Unlike global functions, access to static functions is restricted to the file where they are declared. We can have file level encapsulation using static variables/functions in C because when we make a global variable static, access to the variable becomes limited to the file in which it is declared.

# Question:

What’s going to happen when we compile and run the following C program snippet?

#include "stdio.h"

int main()

{

int a = 10;

int b = 15;

printf("=%d",(a+1),(b=a+2));

printf(" %d=",b);

return 0;

}

A =11 15=

B =11 12=

C Compiler Error due to (b=a+2) in the first printf().

D No compile error but output would be =11 X= where X would depend on compiler implementation.

Explanation:

As per C standard C11, all the arguments of printf() are evaluated irrespective of whether they get printed or not. That’s why (b=a+2) would also be evaluated and value of b would be 12 after first printf(). That’s why correct answer is B.

# Question:

Pick the best statement for the following program.

#include "stdio.h"

int foo(int a)

{

printf("%d",a);

return 0;

}

int main()

{

foo;

return 0;

}

A It’ll result in compile error because foo is used without parentheses.

B No compile error and some garbage value would be passed to foo function. This would make foo to be executed with output “garbage integer”.

C No compile error but foo function wouldn’t be executed. The program wouldn't print anything.

D No compile error and ZERO (i.e. 0) would be passed to foo function. This would make foo to be executed with output 0.

Explanation:

In C, if a function name is used without parentheses, the reference to the function name simply generates a pointer to the function, which is then discarded. So the above program would compile but won’t print anything.

# Question:

Given a boolean function f (x1, x2, ..., xn), which of the following equations is NOT true

A f (x1, x2, ..., xn) = x1'f(x1, x2, ..., xn) + x1f(x1, x2, ..., xn)

B f (x1, x2, ..., xn) = x2f(x1, x2, …, xn) + x2'f(x1, x2, …,xn)

C f (x1, x2, ..., xn) = xn'f(x1, x2, …, 0) + xnf(x1, x2, …,1)

D f (x1, x2, ..., xn) = f(0, x2, …, xn) + f(1, x2, .., xn)

Explanation:

Option A: f (x1, x2, …, xn) = x1’f(x1, x2, …, xn) + x1f(x1, x2, …, xn)

Case 1: taking x1=0 RHS = 1.f(x1, x2, …, xn) + 0.f(x1, x2, …, xn) RHS =f(x1, x2, …, xn).

Case 2: taking x1=1 RHS = 0.f(x1, x2, …, xn) + 1.f(x1, x2, …, xn) RHS =f(x1, x2, …, xn). In both cases RHS=LHS, so, (A) is true

Option B: f (x1, x2, …, xn) = x2f(x1, x2, …, xn) + x2’f(x1, x2, …, xn)

Case 1: taking x2=0 RHS= 0.f(x1, x2, …, xn) + 1.f(x1, x2…,xn) RHS =f(x1, x2, …, xn).

Case 2: taking x2=1 RHS = 1.f(x1, x2, …, xn) + 0.f(x1, x2, …, xn) RHS =f(x1, x2, …, xn). In both cases RHS=LHS, so, (B) is true.

Option C: f (x1, x2, …, xn) = xn’f(x1, x2, …, 0) + xnf(x1, x2, …,1)

Case 1: taking xn=0 RHS= 1.f(x1, x2, …, 0) + 0.f(x1, x2, …, 1) RHS =f(x1, x2, …, 0)

Case 2: taking xn=1 RHS = 0.f(x1, x2, …, 0) + 1.f(x1, x2, …, 1) RHS =f(x1, x2, …, 1)In both cases RHS=LHS, so, (C) is true.

Option D: f (x1, x2, …, xn) = f(0, x2, …, xn) + f(1, x2, .., xn) Here, no way to equate LHS and RHS so ‘NOT true’. NO term depends on value of ‘x1’.