**Rules:**

1. All the variable names or function names used in the program should be meaningful.

(Follow coding guidelines provided for reference)

1. Solve all the questions as a separate file
2. For all the questions, provide the solutions based on your understanding of the concepts and provide necessary explanation to justify your answers (Do not execute and generate the output)
3. All the answers for this set shall be submitted in ***GitHub*** as a ***single file***.

**Questions:**

1. The following will work on a UNIX machine but will fail on a PC

int zip; // zip code for current address

........

zip = 92126;

Why does this fail? What will be the result when run on a PC?

Answer:

C++:

On most UNIX machines, integers are 32 bits (4 bytes), providing a range of 2147483647 (231-1) to -2147483648. On the PC, most compilers use only 16 bits (2 bytes), so the range is 32767 (215-1) to -32768. These sizes are typical.

C:

The C standard does not specify the actual size of numbers. Programs that depend on an integer being a specific size (say 32 bits) frequently fail when moved to another machine

1. Why does this program print ''The value of 1/3 is 0"? What must be done to this

program to fix it?

#include <iostream.h>

float answer; // the result of the divide

main ()

{

answer = 1/3;

cout << "The value of 1/3 is " << answer << "\n";

return (0);

}

Answer:

The compiler will treat the evaluation 1/3 as 0 instead of 0.33 because the compiler will evaluate every equation as integer (i.e., the program returns the output as 0).

The evaluation equation should be type-casted in order to get the desired output as we desired.

Answer=(float) 1/3;

Same method can be used in both c and c++.

1. Example is designed to compute the area of a triangle, given its width

and height. For some strange reason, the compiler refuses to believe that we declared the variable width. The declaration is right there on line two, just after the definition of

height. Why isn't the compiler seeing it?

#include <iostream.h>

int height; /\* the height of the triangle

int width; /\* the width of the triangle \*/

int area; /\* area of the triangle (computed) \*/

main()

{

cout << "Enter width height? ";

cin >> width >> height;

area = (width \* height) / 2;

cout << "The area is " << area << '\n';

return (0);

}

Answer:

The compiler does not see width because the comment statement in line 2 does not end properly which end only in line 3. so the compile will treat the line 3 as comment.

int height; /\* the height of the triangle\*/

int width; /\* the width of the triangle \*/

1. Why does the following program print incorrect answers?

#include <iostream.h>

int array[3][5] = { // Two dimensional array

{ 0, 1, 2, 3, 4 },

{10, 11, 12, 13, 14 },

{20, 21, 22, 23, 24 }

};

main()

{

cout << "Last element is " << array[2,4] << '\n';

return (0);

}

Answer:

The program print incorrect answer because we can’t call array element by using array[2,4] which is wrong statement.

The correct array statement is array[2][4].

cout << "Last element is " << array[2][4] << '\n';

1. Is the following character variable signed or unsigned?

char foo;

Answers:

a. It's signed.

b. It's unsigned.

c. It's compiler dependent.

Answer:

It’s signed, because by default char and int are signed.

We can change it to unsigned explicitly by specifying the keyword unsigned infront of char or int.

1. Explain auto, register, static and extern variables.

Answer:

AUTO:

auto is the default storage class for all local variables.

int count;

auto int count;

REGISTER:

register is used to define local variables that should be stored in a register instead of RAM. This means that the variable has a maximum size equal to the register size (usually one word) and can’t have the unary '&' operator applied to it (as it does not have a memory location).

Register should only be used for variables that require quick access - such as counters. It should also be noted that defining 'register' goes not mean that the variable will be stored in a register. It means that it MIGHT be stored in a register - depending on hardware and implementation restrictions.

register int Miles;

STATIC:

static is the default storage class for global variables. Static variables are automatically initialized to 0.

|  |
| --- |
| static int Count;  int Road;  {  printf("%d\n", Road);  } |

static variables can be 'seen' within all functions in this source file. At link time, the static variables defined here will not be seen by the object modules that are brought in.

static can also be defined within a function. If this is done the variable is initalised at run time but is not reinitalized when the function is called. This inside a function static variable retains its value during vairous calls.

EXTERN:

extern is used to give a reference of a global variable that is visible to all the program files. When you use 'extern' the variable cannot be initialized as all it does is point the variable name at a storage location that has been previously defined.

When you have multiple files and you define a global variable or function which will be used in other files also, then *extern*will be used in another file to give reference of defined variable or function. Just for understanding *extern*is used to declare a global variable or function in another files.

extern int count;

1. Why does the following program fail to print the correct zip code? What does

it print instead?

long int zip; // Zip code

main()

{

zip = 02137L; // Use the zip code for Cambridge MA

cout << "New York's zip code is: " << zip << '\n';

return(0);

}

Answer:

The statement zip=02137L begins with 0. So that the C++ compiler will understand that as octal number instead of decimal number. Any number starting with 0 will be treated as octal by the C++ compiler. The compiler will convert the octal to decimal and print as 1119 for zip=02137L

Output: New York’s zip code is 1119

We can remove zero in 02137 to print as New York’s zip code is 2137.

1. Is it a legal code?

size = 5;

result = ++size;

The first statement assigns size the value of 5. The second statement:

1. Increments size (side effect)

2. Assigns result the value of size (main operation)

But in what order?

1. result is assigned the value of size (5), and then size is incremented.

result is 5 and size is 6.

2. size is incremented, and then result is assigned the value of size (6).

result is 6 and size is 6.

3. The answer is compiler dependent and varies from computer to computer.

4. If you don't write code like this, you don't have to worry about these sorts of questions.

Answer:

It is legal code.

The first statement assigns size the value of 5.

The second statement:

1. Increments size (side effect)

2. Assigns result the value of size (main operation)

Size is incremented, and then result is assigned the value of size (6).

result is 6 and size is 6.

1. How will you change the program to print ch as int?

#include <iostream.h>

signed char ch; // Very short integer

// Range is -128 to 127

int main()

{

ch = 37;

cout << "The number is " << ch << '\n';

return (0);}

Answer:

The above program will print as “The number is %”. To make it print the int value hard code in the program we need to type cast the variable ch to integer.

cout << "The number is " << int(ch) << '\n';

it will print as “The number is 37”.

1. For some strange reason, the program here thinks that everyone owes a balance of O dollars. Why?

#include <iostream.h>

int balance\_owed; // amount owed

main ()

{

cout << "Enter number of dollars owed:";

cin >> balance\_owed;

if (balance\_owed = 0)

cout << "You owe nothing.\n";

else

cout << "You owe " << balance\_owed << " dollars.\n";

return (0);

}

*Sample output:*

Enter number of dollars owed: 12

You owe 0 dollars.

Answer:

The program think that everyone owes a balance of 0 dollar because in the if statement we have used assignment operation ( balance\_owed=0 ) instead of condition/expression statements ( balance\_owed == 0 ).

So the compiler will make owed balance as 0 dollar for everyone. To solve this we need to replace the assignment operation ( balance\_owed=0 ) with condition/expression statements ( balance\_owed == 0 ) to work correctly.

1. What is the error in this program?

#include <iostream.h>

/\*

\* This program produces a Celsius to Fahrenheit conversion

\* chart for the numbers 0 to 100.

\*

\* Restrictions:

\* This program deals with integers only, so the

\* calculations may not be exact.

\*/

// The current Celsius temperature we are working with

int celsius;

main() {

for (celsius = 0; celsius <= 100; ++celsius);

cout << "Celsius: " << celsius <<

" Fahrenheit: " << ((celsius \* 9) / 5 + 32) << '\n';

return (0);

}

When run, this program prints out:

Celsius: 101 Fahrenheit: 213

and nothing more. Why?

Answer:

The print statement should be within the looping statement. But we have specified the print statement outside the looping statement. At the end of the loop the value of the Celsius will be 101. So the print statement will print Celsius: 101 Fahrenheit: 213.

If we want the program to print every value of Celsius then we should specify print statement within the looping statement.

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

cout << "Celsius: " << celsius <<

" Fahrenheit: " << ((celsius \* 9) / 5 + 32) << '\n';

}

1. This program reads a list of five numbers and counts the number of threes and sevens in the data. Why does it give us the wrong answers?

include <iostream.h>

int seven\_count; // Number of sevens in the data

int data[5]; // The data to count 3 and 7 in

int three\_count; // Number of threes in the data

int index; // Index into the data

main() {

seven\_count = 0;

three\_count = 0;

cout << "Enter 5 numbers\n";

cin >> data[1] >> data[2] >> data[3] >>

data[4] >> data[5];

for (index = 1; index <= 5; ++index)

if (data[index] == 3)

++three\_count;

if (data[index] == 7)

++seven\_count;

}

cout << "Threes " << three\_count << " Sevens " << seven\_count << '\n';

return (0);

}

When we run this program with the data 3 73 0 2, the results are:

Threes 4 Sevens 1

(Your results may vary.)

Answer:

The program gives wrong answer because the array index varies from 0 to n-1 where n is the index size. So memory for data[5] will not be allocated. Garbage value will be stored at memory location of data[5]. We can solve this issue by changing the size of data to 6 or we can change the index value to be starting from 0 instead of 1.

int data[6];

for (index = 0; index < 5; ++index)

1. The following program generates the answer 47 instead of the expected answer 144. Why?

#include <iostream.h>

#define FIRST\_PART 7

#define LAST\_PART 5

#define ALL\_PARTS FIRST\_PART + LAST\_PART

main()

{

cout << "The square of all the parts is " <<

ALL\_PARTS \* ALL\_PARTS << '\n';

return (0);

}

Answer:

The value of ALL\_PARTS will be like FIRST\_PART+LAST\_PART

ALL\_PARTS=7+5;

The statement ALL\_PARTS\*ALL\_PARTS will be evaluated as

7+5\*7+5 in which \* gets higher precedence and execute before +.

7+35+5=42.

We can change this program to print 144 instead of 47 by placing the FIRST\_PART+LAST\_PART with in braces.

# define ALL\_PARTS (FIRST\_PART+LAST\_PART)

This statement will print 144 as result.

1. This program generates a warning that counter is used before it is set.

This is a surprise because the for loop should set it. You also get a very strange warning, "null effect, "for line 11.

1 // Warning, spacing is VERY important

2

3 #include <iostream.h>

4

5 #define MAX 10

6

7 main()

8 {

9 int counter;

10

11 for (counter = MAX; counter > 0;

12 --counter)

13 cout << "Hi there\n";

14

15 return (0);

16 }

Answer:

This program is not generating any warning as mentioned.

1. This Example computes the wrong value for size. Why?

#include <iostream.h>

#define SIZE 10;

#define FUDGE SIZE -2;

main ()

{

int size; // Size to really use

size = FUDGE;

cout << "Size is " << size << '\n';

return (0);

}

Answer:

The semi-colon at the end of #define do not do anything to this program.

#define SIZE 10;

#define FUDGE SIZE -2;

Here size value will be 10; so #define FUDGE 10; -2;. This semi-colon in between size and -2 cause the problem. By removing the semi-colon at the end of both #define statement will solve this issue.

#define SIZE 10

#define FUDGE SIZE -2

1. The following program is supposed to print the message "Fatal Error:

Abort" and exit when it receives bad data. But when it gets good data, it exits. Why?

1 #include <iostream.h>

2 #include <stdlib.h> /\* ANSI Standard only \*/

3

4 #define DIE \

5 cerr << "Fatal Error: Abort\n"; exit(8);

6

7 main() {

8 // A random value for testing

9 int value;

10

11 value = 1;

12 if (value < 0)

13 DIE;

14

15 cerr << "We did not die\n";

16 return (0);

17 }

Answer:

1. What does the above program output? (Try running it on your machine.). Why did it output what it did? (Try checking the output of the preprocessor.)

#include <iostream.h>

#define SQR(x) ((x) \* (x))

main ()

{

int counter; /\* Counter for loop \*/

counter = 0;

while (counter < 5)

cout << "x ” << counter + 1 <<

" x squared" << SQR(++counter) << '\n';

return (0);

}

Why does this not produce the expected output? How much does the counter go up each time.

Answer:

The increment operator in the SQR function increments the counter two times in the counter.

#define SQR(x) ((x)\*(x))

SQR(++counter);

This will work as ((++counter) \* (++counter)). So the counter is incremented two times in the SQR function.

The preprocessor will evaluate from right to left so the SQR function is evaluated first then the counter+1 statement will be evaluated that why the output is like

x 3 x squared 2

x 5 x squared 12

x 7 x squared 30

if we want output to be like this

x 1 x squared 1

x 2 x squared 4

x 3 x squared 9

x 4 x squared 16

x 5 x squared 25

then we need to increment the counter outside the SQR function.

while (counter < 5){

cout << "x ” << counter + 1 <<

" x squared" << SQR(counter+1) << '\n';

++counter;}

1. The following program tells us we have an undefined variable, but our only variable name is counter. Why?

#include <iostream.h>

#define RECIPROCAL (number) (1.0 / (number))

main()

}

float counter;

for (counter = 0.0; counter < 10.0;

counter += 1.0)

cout << "1/" << counter << " = "<<

RECIPROCAL(counter) << "\n";

}

return (0);

}

Answer:

The undefined variable error is caused by (number) in the # define statement.

The space between the RECIPROCAL and (number) tells the compiler that number is not a functional variable.

To make it as functional variable we need to remove the space between RECIPROCAL and (number).

#define RECIPROCAL(number) (1.0 / (number))

Now the compiler will treat number as functional argument and will not tell us that we have undefined variable.

1. In the following program, the HIGH\_SPEED flag works, but the DIRECT\_CONNECT flag does not. Why?

#include <iostream.h>

const int HIGH\_SPEED = (1<<7); /\* modem is running fast \*/

// we are using a hardwired connection

const int DIRECT\_CONNECT = (1<<8);

char flags = 0; // start with nothing

main()

{

flags |= HIGH\_SPEED; // we are running fast

flags |= DIRECT\_CONNECT; // because we are wired together

if ((flags & HIGH\_SPEED) != 0)

cout <<"High speed set\n";

if ((flags & DIRECT\_CONNECT) != 0)

cout <<"Direct connect set\n";

return (0);

}

Answer:

1. In Example the first loop works, but the second fails. Why?

#include <iostream.h>

main ()

{

short int i;

// Works

for (i = 0x80; i !=0; i = (i >> 1)) {

cout << "i is " << hex << i << dec << '\n';

}

signed char ch;

// Fails

for (ch = 0x80; ch!=0; ch = (ch >> 1))

cout << "ch is " << hex << int(ch) << dec << '\n';

}

return (0);

}

Answer: