26 June 2020

|  |  |  |  |
| --- | --- | --- | --- |
| Date: | 26 June 2020 | Name: | Srinidhi J C |
| Course: | C++ Programming | USN: | 4al16ec078 |
| Topic: | Inheritance & Polymorphism, | Semester & Section: | 8th & b |
| Github Repository: | SrinidhiJC078 |  |  |
| FORENOON SESSION DETAILS | | | | | |
| Image of session  A screenshot of a cell phone  Description automatically generated | | | | | |

Report:

1. Which of the following declarations is illegal?

char str = "hello";

char \*str = "hello";

double a = 1;

const int\* p1;

1. Drag and drop from the options below to declare a class "MyClass", with corresponding constructor and destructor. Constructor initializes class's private integer member named "mem".

class

 MyClass {  
public:  
MyClass() {  
   mem = 12; }

~MyClass()

 {  
   }

private

:  
 int mem;  
};

# Variable Scope in C++

A scope is a region of the program and broadly speaking there are three places, where variables can be declared −

* Inside a function or a block which is called local variables,
* In the definition of function parameters which is called formal parameters.
* Outside of all functions which is called global variables.

We will learn what is a function and it's parameter in subsequent chapters. Here let us explain what are local and global variables.

## Local Variables

Variables that are declared inside a function or block are local variables. They can be used only by statements that are inside that function or block of code. Local variables are not known to functions outside their own. Following is the example using local variables −

#include <iostream>

using namespace std;

int main () {

// Local variable declaration:

int a, b;

int c;

// actual initialization

a = 10;

b = 20;

c = a + b;

cout << c;

return 0;

}

## Global Variables

Global variables are defined outside of all the functions, usually on top of the program. The global variables will hold their value throughout the life-time of your program.

A global variable can be accessed by any function. That is, a global variable is available for use throughout your entire program after its declaration. Following is the example using global and local variables −

#include <iostream>

using namespace std;

// Global variable declaration:

int g;

int main () {

// Local variable declaration:

int a, b;

// actual initialization

a = 10;

b = 20;

g = a + b;

cout << g;

return 0;

}

A program can have same name for local and global variables but value of local variable inside a function will take preference. For example −

#include <iostream>

using namespace std;

// Global variable declaration:

int g = 20;

int main () {

// Local variable declaration:

int g = 10;

cout << g;

return 0;

}

Result −

10

## String Literals

String literals are enclosed in double quotes. A string contains characters that are similar to character literals: plain characters, escape sequences, and universal characters.

You can break a long line into multiple lines using string literals and separate them using whitespaces.

Here are some examples of string literals. All the three forms are identical strings.

"hello, dear"

"hello, \

dear"

"hello, " "d" "ear"

## Defining Constants

There are two simple ways in C++ to define constants −

* Using **#define** preprocessor.
* Using **const** keyword.

## The #define Preprocessor

Following is the form to use #define preprocessor to define a constant −

#define identifier value

Following example explains it in detail −

[Live Demo](http://tpcg.io/N6xxDP)

#include <iostream>

using namespace std;

#define LENGTH 10

#define WIDTH 5

#define NEWLINE '\n'

int main() {

int area;

area = LENGTH \* WIDTH;

cout << area;

cout << NEWLINE;

return 0;

}

When the above code is compiled and executed, it produces the following result −

50

## The const Keyword

const type variable = value;

Following example explains it in detail −

#include <iostream>

using namespace std;

int main() {

const int LENGTH = 10;

const int WIDTH = 5;

const char NEWLINE = '\n';

int area;

area = LENGTH \* WIDTH;

cout << area;

cout << NEWLINE;

return 0;

}

Result −

50

# C++ Modifier Types

C++ allows the **char, int,** and **double** data types to have modifiers preceding them. A modifier is used to alter the meaning of the base type so that it more precisely fits the needs of various situations.

The data type modifiers are listed here −

* signed
* unsigned
* long
* short

The modifiers **signed, unsigned, long,** and **short** can be applied to integer base types. In addition, **signed** and **unsigned** can be applied to char, and **long** can be applied to double.

The modifiers **signed** and **unsigned** can also be used as prefix to **long** or **short** modifiers. For example, **unsigned long int**.

C++ allows a shorthand notation for declaring **unsigned, short,** or **long** integers. You can simply use the word **unsigned, short,** or **long,** without **int**. It automatically implies **int**. For example, the following two statements both declare unsigned integer variables.

unsigned x;

unsigned int y;

#include <iostream>

using namespace std;

/\* This program shows the difference between

\* signed and unsigned integers.

\*/

int main() {

short int i; // a signed short integer

short unsigned int j; // an unsigned short integer

j = 50000;

i = j;

cout << i << " " << j;

return 0;

}

Output −

-15536 50000