PROGRAMMING FUNDAMENTALS

Lecture # 4

Date: 14-Sep-2019

Credit hours: 1

TODAY'S AGENDA

- Data types practice
  - Float, double, bool, char, string
- Operator precedence
- Unary & Binary operators
- Increment & Decrement Operators
- const
- Type casting

DATA TYPES

- **Float & double**
- Float is a shortened term for "floating point.“
- For example, 321.1234567 cannot be stored in float because it has 10 digits. If greater precision—more digits—is necessary, the double type is used.
- The data type double is also used for handling floating-point numbers. But it is treated as a distinct data type because, it (double data type) occupies twice as much memory as type float, and stores floating-point numbers with much longer range and precision

FLOAT & DOUBLE DATA TYPE

```
#include <iostream>
using namespace std;
int main ()
{
        float myNum = 5.75;
        cout << myNum;
        return 0;
}
```

```
#include <iostream>
using namespace std;
int main ()
{
        double myNum = 19.4565756899;
        cout << myNum;
        return 0;
}
```

DATA TYPES (CONT'D)

- **Booleans**
- A Boolean data type is declared with the **bool** keyword
- Can only take the values true or false
- When the value is returned, true = 1 and false = 0

```
#include <iostream>
using namespace std;
int main() {
    bool isCodingFun = true;
    bool isFishTasty = false;
    cout << isCodingFun << "\n";
    cout << isFishTasty;
    return 0;
}
```

Output:
1
0

DATA TYPES (CONT'D)

- **Characters**
- The char data type is used to store a single character.
- The character must be surrounded by single quotes, like 'A' or 'c'
- Keyword: char

```
#include <iostream>
using namespace std;
int main ()
{
        char myGrade = 'B';
        cout << myGrade;
        return 0;
}
```

CHAR DATA TYPE (CONT'D)

- Alternatively, you can use ASCII values to display certain characters

```
#include <iostream>
using namespace std;
int main ()
{
    char a = 65, b = 66, c = 67;
    cout << a;
    cout << b;
    cout << c;
    return 0;
}
```

Output:
ABC

STRING DATA TYPE

- The string type is used to store a sequence of characters (text)
- This is not a built-in type, but it behaves like one in its most basic usage
- String values must be surrounded by double quotes “text”

```
#include <iostream>
#include <string>
using namespace std;
int main()
{
    string greeting = "Hello";
    cout << greeting;
    return 0;
}
```

Output:
Hello

STRING DATA TYPE (CONT'D)

- **String Concatenation**
- **The + operator can be used between strings to add them together to make a new string**
- **String Length**

```
#include <iostream>
#include <string>
using namespace std;
int main () {
        string firstName = "John ";
        string lastName = "Doe";
        string fullName = firstName + lastName;
        cout << fullName;
        cout << "The length of  first name is: " << firstName .length();
         return 0;
}
```

Output:
John Doe
The length of first name is: 4

STRING DATA TYPE (CONT'D)

- **User input strings**

```
string fullName;
cout << "Type your full name: ";
cin >> fullName;
cout << "Your name is: " << fullName;
```

// Type your full name: John Doe
// Your name is: John

- You would expect the program to print "John Doe", but it only prints "John"
- cin considers a space (whitespace, tabs, etc) as a terminating character, which means that it can only display a single word
- we often use the getline() function to read a line of text

STRING DATA TYPE (CONT'D)

```
#include <iostream>
#include <string>
using namespace std;
int main()
{
        string fullName;
        cout << "Type your full name: ";
        getline (cin, fullName);
        cout << "Your name is: " << fullName;
        return 0;
}
```

Output:

```
Type your full name: John Doe
Your name is: John Doe
```

ANSWER ??

```
#include <iostream>
using namespace std;
int main ()
{
        int x = 10;
        int y = 20;
        int z = x + y;
        cout << z;
        return 0;
}
```

```
#include <iostream>
#include <string>
using namespace std;
int main ()
{
        string x = "10";
        string y = "20";
        string z = x + y;
        cout << z;
        return 0;
}
```

OPERATOR PRECEDENCE

- A single expression may have multiple operators
- X=5+7%2;
- In C++, the above expression always assigns 6 to variable x
- X=5+(7%2); ??
- X=(5+7)%2; ??

	Operator Precedence
1	! Logical not (Highest)
2	( ) Parenthesis
3	*, /, %
4	+, -
5	>, >=, <, <=
6	==, !=
7	&& (AND)
8	|| (OR)
9	= (Lowest)

OPERATORS

- **Unary Operator**
- Unary operators only require one operand
- The most common example is the unary minus: -N
  - Changes the sign of the value stored in variable N
- Increment (++) Unary operator
- Decrement (--) Unary operator
- The minus (-) unary
- The logical not (!) operator

- **Binary Operator**
- Binary operators require two operands
- The arithmetic operators are the most familiar examples of binary operators
- A = counter + 5;
- Answer?
- Number of operands for + and =

INCREMENT & DECREMENT OPERATORS

- The increment operator ++ adds 1 to its operand
- x = x+1; is equivalent to x++;
- The decrement operator -- subtracts 1 from its operand
- x = x-1; is equivalent to x--;

- Prefix form
  - ++x; --x;
  - Increment or decrement will be done before rest of the expression
- Postfix form
  - x++; x--;
  - Increment or decrement will be done after the complete expression is evaluated

ANSWER ??

```
#include <iostream>
using namespace std;

main() {
   int a = 21;
   int c ;

   // Value of a will not be increased before assignment.
   c = a++;
   cout << "Line 1 - Value of a++ is :" << c << endl ;

   // After expression value of a is increased
   cout << "Line 2 - Value of a is :" << a << endl ;

   // Value of a will be increased before assignment.
   c = ++a;
   cout << "Line 3 - Value of ++a is  :" << c << endl ;
   return 0;
}
```

```
Line 1 - Value of a++ is :21
Line 2 - Value of a is :22
Line 3 - Value of ++a is  :23
```

CONST

- const keyword is used to define constant values in a source code
- We previously discussed #define
- **#define is pre-processor directive while const is a keyword**
- **#define is not scope controlled whereas const is scope controlled**
  - Value defined by #define can be used in anywhere in the program but the constant can be declared inside the function and thus, it can be accessed only within the function/scope in which constant is declared

```
#include <iostream>
using namespace std;

//macro definition
#define X 30

//global integer constantt
const int Y = 10;

int main()
{
        //local ineteger constant`
        const  int Z = 30;

        cout<<"Value of X:  "<<X<<endl;
        cout<<"Value of Y:  "<<Y<<endl;
        cout<<"Value of Z:  "<<Z<<endl;

        return 0;
}
```

CONST (CONT'D)

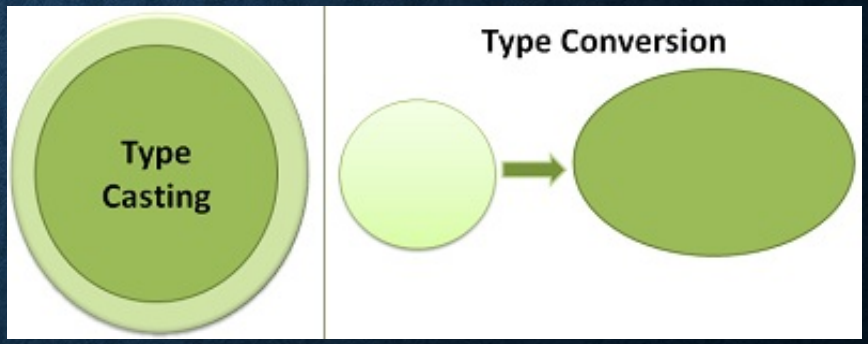
- Defined can be redefined anywhere in the program
  - by un-defining and then defining
- constant cannot be re-declared or re-defined even we cannot re-assign the value in constant

```
#include <iostream>
using namespace std;

//constant int
const int Y=10;

int main()
{
        cout<<"Value of Y: "<<Y<<endl;
        Y=100;  //error, we can not assign value to const
        cout<<"Value of Y: "<<Y<<endl;

        return 0;
}
```

TYPE CASTING

- Type conversion is the automatic conversion of one data type to another whenever required, done implicitly by the compiler.
- Type casting is the conversion of one data type to another whenever required, done by the user

1. int a = 2;
2. float b = a;



- Here, a is promoted to higher data type ‘float’ and is assigned to ‘b’ by the compiler itself so it is Type Conversion and we get the value b = 2

TYPE CASTING (CONT'D)

- If you try the other way round,

```
1. float a = 2.0;
2. int b = a;
```

- Here, on assigning a to b there maybe precision loss so compiler doesn't convert and that calls for explicitly changing the data type of a to assign it to b using ( ) operator

```
1. float a = 2.0;
2. int b = (int)a;
```

```
3.int b = static_cast<int> (a);
```


Why
What
Where
Who
When