19CSE201 :Advanced Programming

Lecture 4
Selection, Repetition & Functions

By
Ritwik M
Assistant Professor(SrGr)
Dept. Of Computer Science & Engg

Based on material from The Professor's C++Course by Linda W. Friedman and Stroustrup B. Programming: principles and practice using C++. Second edition,Addison Wesley; 2014.

A Quick Recap

- Basic C++ constructs
- Input/output
- Variables
- Datatypes
- Declaration
- Type casting
- Overflow
- Operators
- Errors
- Programming Paradigms

Selection Constructs

- If
  - ```
    if(mark>=50)
    cout<< "passed";
    ```
- If-else
  - ```
    if(mark>=50)
    cout<< "passed";
    else
    cout<<"Failed";
    ```
- Nested If
  - ```
    if(mark>=50)
    cout<< "passed";
    if(mark>=80)
    cout<<"Distinction";
    ```
- Switch-Case
  - ```
    int main(){
    int score;
    cout << "Enter the test score:  ";
    cin >> score;
    switch (score/10) {
      case  6: cout << 'C' << endl; break;
      case  5: cout << 'P' << endl; break;
      case  4:
      case  3:
      case  2:
      case  1:
      case  0: cout << 'F' << endl; break;
      default: cout << "Error: \n";
     }
    return 0;
    }
    ```

Ternary Operator

- AKA Conditional Operator in C++
- ? :
- Syntax
  - <condition> ? <Action if TRUE> : <Action if FALSE>
- Example
  - `grade>=60? cout << "Passed" : cout << "Failed";`
  - `cout << (grade >=60? "Passed" : "Failed");`

Iteration

- Iteration ≡ Repetition ≡ Looping
- In order to control iteration, we use one of three structured control statements.
- For
    - Usually a counting loop.
- While
    - A "test before" loop.
- do/while
    - A "test after" loop.

A Short Note on Branching

- Conditional
  - Break
    - used to break out of a loop.
    - Control is transferred to the first statement after the end of the loop.
  - Continue
    - skips the remainder of the body of the loop and continues the loop.
    - Transfers control to "loop again."
- Unconditional
  - GOTO
    - highly discouraged in any programming language!
    - it makes difficult to trace the control flow of a program, making the program hard to understand and hard to modify.

A Short Exercise – What is the output ?

```
#include <iostream>
using namespace std;
int main(){
    int x;
   for (x=1; x<=10; x++) {
       if (x==5)
          break;
      cout << x << " ";
   }  //end for
   cout << "\n What happened at x = " << x << endl;
   cout<< endl << endl << endl;
   return 0;
}
```

Exercise 11 – What is the output ?

```
#include <iostream>
using namespace std;
int main(){
    int x;
   for (x=1; x<=10; x++) {
       if (x==5)
                continue;
       cout << x << " ";
   }
   cout << "\n What happened here? " << endl;
   return 0;
}
```

Test your C++ Skills – Write Programs

1. To check whether a number is palindrome or not.
2. To find frequency of each digit in a given integer.
3. To enter a number and print it in words.
4. To print all ASCII character with their values.
5. To find power of a number using loop.
6. To find all factors of a number.
7. To calculate factorial of a number.
8. To find HCF (GCD) of two numbers.
9. To find LCM of two numbers

Try this on the HPOJ IDE.

Submit your code as a single word file on AUMS

Functions & Data

- Functions – Modules in a program
- Many ways to deal with data in functions
  - Data may be required by the function, an input value much like f(x) in math.
  - The function may compute a value that must be returned to the point of call, again, much like a mathematical function, y=f(x).
  - The function may have zero or more data value(s) to return to the point of call
  - The function may only use certain data items locally.
  - The function may be required to change the values of data used elsewhere in the program.

Global and Local variables

- Global variables
  - a.k.a. external variables
  - are accessible throughout all the functions in the program.
- Local variables
  - a.k.a. automatic variables
  - are accessible only within the function (or other program module) in which they are declared.

Scope

- Every name (identifier) in a C++ program must refer to a unique entity.
- Scope refers to the area of the program code where a declared variable (or object or function or ...) is accessible,
  - i.e., where it can be referenced.
  - Depending on its scope, a variable may exist (lifetime) but still not be visible.
  - The lifetime of a variable is the duration of time during which it takes up space in memory, i.e., it exists.
- Local scope
  - local to function, block of code (compound statement)
- Namespace scope; class scope -- Discussed Later

Scope (Example)

```
#include <iostream>
using namespace std;
int main(){
    int x=10;
    cout << x << endl << endl;
    {
        int x=25;
        cout << x << endl << endl;
    }
    cout << x << endl << endl;
    return 0;
}
```

Parameter Passing

- We know that when we call a function, we can pass data to it in the parameter list.
  - An actual parameter, or argument, is the data item that is passed from the calling function to the called function.
  - A formal parameter, or parameter, is the variable in the parameter list in the function implementation.
- All are familiar and have used the default parameter passing mechanism, pass by value.
  - The value of the argument is copied into the corresponding parameter
  - it is similar to a local variable in the executing function.

Example

Suppose we wish to write a function that will swap two numeric values.

```
void swap (float x, float y){
     float temp = x;
     x = y;
     y = temp;
}
```

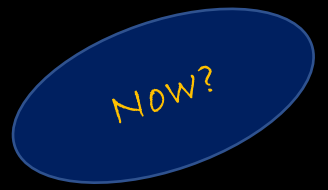
Example Cont.

Here's a program to test the swap function:


```
#include <iostream>
using namespace std;
void swap (float,
  float);
int main(){
float a = 10;
float b = 27;
cout << "A=   " << a <<
  endl
     <<"B=   " << b <<
  endl;
swap(a,b);
cout << "After
  swapping..." << endl
     << "A=   " << a <<
  endl
     <<"B=   " << b <<
  endl;
return 0;
}
```

Example Cont.

Trying again with reference parameters:




```
#include <iostream>
using namespace std;
void swap (float&, float&);
int main(){
    float a = 10;
    float b = 27.3;
    cout << "A=  " << a <<
 endl
        <<"B=  " << b <<
 endl;
    swap(a,b);
    cout << "After swapping..." <<
  endl
        << "A=  " << a << endl
        <<"B=  " << b << endl;
    return 0;
}
void swap (float &x, float
  &y){
    float temp = x;
    x = y;
    y = temp;
}
```

Overview of Functions

Passing By Value	Passing By Reference
int x;	int &x;
Formal parameter is a local variable	Formal parameter is a local reference
Formal parameter is a duplicate of the actual parameter	Formal parameter is a synonym for the actual parameter
Formal parameter cannot change the actual parameter	Formal parameter can change the actual parameter
Actual parameter can be a constant, variable or expression	Actual parameter must be a variable
Actual parameter is read only	Actual parameter is read-write

Quick Summary

- Selection Operations
- Iteration
- Branching
- Functions and Data
- Global Variable
- Local Variable
- Scope
- Formal and Actual Parameters

Up Next

Objects and Classes