Lecture 3 Notes

Review from Lecture 2

You should know how to use the following new expressions:

- Comparison expressions: For mathematically comparing values
- Logical expressions: For logically comparing values
- Arithmetic assignment expressions: For mathematically modifying variables
- Increment/Decrement expressions: A convenient way for modifying variables used in iteration statements

You should know how to use the following new statements:

- if statement: For selecting a statement to execute based on a condition
- Iteration statements: This includes the while statement, do-while statement, and for statement

Functions

In C++, a function is a unit of statements similar to a compound statement. Like the compound statement, a function contains multiple statements. The difference between a function and a compound statement is that a function can produce a value like an expression; a function may also operate on a provided set of variables when it is executed.

A function follows this form:

data-type function-name `(` *parameter-list* `)` { *statements* }

- *data-type* refers to the data type of the value that this function produces. If the function does not produce a value, you can set the data type to `void`.
- *function-name* is the name of your function.
- *parameter-list* is the list of variables that your function can receive. You can omit this if you do not need them.
- *statements* are the statements that will be executed when your function gets called.

Perhaps functions will make more sense with an example:

Listing 1: Example of a program that uses a function

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

void hr()
{
    cout << "-----------------------------------" <<
    "----------------------------------------" <<
    "-----" << endl;
}

int main()
{
    hr();
    cout << "Fancy" << endl;
    hr();
    return 0;
}
```

This example code shows the function named "hr" which prints a horizontal line in the terminal window. The keyword "void" denotes that this function does not return a value. The empty pair of parenthesis after hr denotes that this function does not have any input variables (known as parameters).

Whenever we want to execute the code inside our hr function, we use `hr()` in our C++ statement.

Functions may also receive input values which are stored as local variables inside the function. Consider this example:

```
void hr(char c)
{
    int i;
    for (i = 0; i < 79; ++i) {
        cout << c;
    }
    cout << endl;
}
```

Now our hr function can use any character for the horizontal line. To execute our function, we must specify a character value as an argument to the function like this:

```
hr('=');
```

C++ supports default values for functions:

```
void hr(char c = '-', int len = 79)
{
    int i;
    for (i = 0; i < len; ++i) {
        cout << c;
    }
    cout << endl;
```

```
}

int main()
{
    hr();
    cout << "Fancy";
    hr();
}
```

Since no arguments were given to the hr function, the values for parameters c and len defaulted to '-' and 79. For this function, we could have also given 1 argument for parameter c and omitted the argument for the len parmaeter.

Functions may also have return values:

Listing 2: gcd.cpp

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

int gcd(int a, int b)
{
    int n;

    if (a < b) {
        n = a;
    } else {
        n = b;
    }
    while (n > 1) {
        if (a % n == 0 and b % n == 0) {
            return n;
        }
        --n;
    }
    return 1;
}

int main()
{
    int n, m;

    cout << "Enter an integer: ";
    cin >> n;
    cout << "Enter another integer: ";
    cin >> m;
    cout << "The greatest common divisor is: " <<
    gcd(n, m) << endl;
}
```

This program finds the greatest common divisor of two integers. The "int" before

the function name "gcd" indicates that the data type of the function's return value is an integer. If our function has no return value, we use "void" as the return type.

Functions are useful because they are often created to perform a particular task. The task that a function performs can be re-used in other programs that you create. With the right number of functions, you can break down any big problem into a number of smaller tasks.

Commenting

Now that you know about functions, you will be creating more sophisticated programs. Very often, you may start a software project, take a long break from it, and find difficulty in continuing the project from where you left off. Your colleagues may also have difficulty understanding your source code when the project becomes very big.

To help prevent this, C++ allows us to insert notes in our source code without modifying our program's execution. These notes are called comments. Here is an example of commented code (next page):

Listing 3: prime.cpp

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


/* Returns true if n is a prime integer.
 * A prime integer is only evenly divisible by
 * 1 and itself.
 */
bool is_prime(int n)
{
    int i;

    /* We stop after n/2 since any integer greater
     * than half of n is not a factor of n.
     */
    for (i = 2; i <= n / 2; ++i) {
        /* n modulo i is zero if i is a factor of n */
        if (n % i == 0) {
            return false;
        }
    }
    return true;
}

/* Calculates the next prime integer after n */
int next_prime(int n)
{
    /* Keep incrementing n until it is a prime integer */
    do {
        ++n;
    } while (not is_prime(n));
    return n;
}

int main()
{
    int n;

    cout << "Enter an integer: ";
    cin >> n;
    cout << "The next prime integer after " << n <<
    " is " << next_prime(n) << endl;
    return 0;
}
```

All multi-line comments in C++ begin with `/*` and end with `*/`. You may also have single line comments which begin with `//`. Single-line comments affect all text from the start of `//` until the end of the line.

– *Mark Swoope*