CENG 112 – Data Structures

C++ Basics

Mustafa Özuysal
mustafaozuysal@iyte.edu.tr

February 24, 2017

İzmir Institute of Technology

Primitive Types and Operators

Variables

In C++, we have to define all variables before their first use.

```
<type> <identifier>; // or
<type> <identifier> = <initial value>;
```

Integral Types	Floating-Point Types
`bool`, `char`, `short`, `int`, `long`	`float`, `double`

Note `char`, `short`, `int`, `long` also have `unsigned` versions.
`bool` can take two values `true` or `false`.

Variables

```
1 int i = -123;
2 long l = 123L;
3 double d = 123.0;
4 float f = 123.0f;
5 double d2 = 1e-5;
6 unsigned int u = 12345;
7 char c = 'A';
8 const char s[] = "a string";
```

Operators

- Arithmetic: `+, -, *, \, %`

Operators

- Arithmetic: `+, -, *, \, %`
- Logical: `<, <=, >, >=, !=, ==, ||, &&, !`

Operators

- Arithmetic: `+, -, *, \, %`
- Logical: `<, <=, >, >=, !=, ==, ||, &&, !`
- Bitwise: `&, |, ^, ~, <<, >>`

Operators

- Arithmetic: `+, -, *, \, %`
- Logical: `<, <=, >, >=, !=, ==, ||, &&, !`
- Bitwise: `&, |, ^, ~, <<, >>`
- Assignment: `=, +=, -=, *=, ...`

Operators

- Arithmetic: `+, -, *, \, %`
- Logical: `<, <=, >, >=, !=, ==, ||, &&, !`
- Bitwise: `&, |, ^, ~, <<, >>`
- Assignment: `=, +=, -=, *=, ...`
- Pre/post increment/decrement: `++,--`

Operators

- Arithmetic: `+, -, *, \, %`
- Logical: `<, <=, >, >=, !=, ==, ||, &&, !`
- Bitwise: `&, |, ^, ~, <<, >>`
- Assignment: `=, +=, -=, *=, ...`
- Pre/post increment/decrement: `++,--`
- Conditional: `<condition> ? <true-part> : <false-part>`

Example

Square-roots with Newton's method:

- Set initial `guess = 1.0` for the square root of x
- Improve guess with $\texttt{guess} \leftarrow \frac{\texttt{guess} + \frac{x}{\texttt{guess}}}{2}$
- Stop when $|\texttt{guess} * \texttt{guess} - x| < \tau$

Example

```cpp
#include <cstdlib>
#include <iostream>
#include <cmath>

using namespace std;

const double THRESHOLD = 1e-4;

double sqrt_newton(double y);

int main(int argc, char** argv)
{
        for (double x = 0.0; x < 10.0; x += 1.0) {
                double c_sqrt = sqrt(x);
                double n_sqrt = sqrt_newton(x);
                cout << x << " " << c_sqrt << " " << n_sqrt << endl;
        }

        return EXIT_SUCCESS;
}
```

Example

```
22 bool sqrt_good_enough(double guess, double y)
23 {
24         return fabs(guess*guess-y) < THRESHOLD;
25 }
26
27 double sqrt_improve(double guess, double y)
28 {
29         return (guess + y/guess)/2.0;
30 }
31
32 double sqrt_newton(double y)
33 {
34         double guess = 1.0;
35         while (!sqrt_good_enough(guess, y))
36                 guess = sqrt_improve(guess, y);
37         return guess;
38 }
```

Control Flow and Functions

`if` Statement

```
if (<cond>)
    <statement>; // Executed only if <cond> is true
```

`if` Statement

```
if (<cond>)
    <statement>; // Executed only if <cond> is true
```

```
if (<cond>)
    <statement>; // Executed only if <cond> is true
else
    <statement>; // Executed only if <cond> is false
```

`if` Statement

```
if (<cond>)
    <statement>; // Executed only if <cond> is true
```

```
if (<cond>)
    <statement>; // Executed only if <cond> is true
else
    <statement>; // Executed only if <cond> is false
```

```
if (<cond1>)
    <statement>; // <cond1> true
else if (<cond2>)
    <statement>; // <cond1> false && <cond2> true
else
    <statement>; // <cond1> && <cond2> both false
```

`while` and `for` Statements

```
while (<cond>)
    <statement>; // Executed as long as <cond> is true
```

`while` and `for` Statements

```
while (<cond>)
    <statement>; // Executed as long as <cond> is true
```

```
for (<expr1>; <expr2>; <expr3>)
    <statement>; // Executed as long as <expr2> is true
```

`while` and `for` Statements

```
while (<cond>)
    <statement>; // Executed as long as <cond> is true
```

```
for (<expr1>; <expr2>; <expr3>)
    <statement>; // Executed as long as <expr2> is true
```

↓

```
<expr1>;
while (<expr2>) {
    <statement>;
    <expr3>;
}
```

Functions

```
<return-type>  <function-name>(<parameter-list>|<void>)
{
    <function-body>;
}
```

Pass-by-Value v.s. Pass-by-Reference

By default all primitive data types are passed by value.

```
1 void add2(int x) { x += 2; }
2
3 ...
4 int x = 1;
5 add2(x);
6 cout << x << endl; // Prints 1
```

Pass-by-Value v.s. Pass-by-Reference

By default all primitive data types are passed by value.

```
1 void add2(int x) { x += 2; }
2
3 ...
4 int x = 1;
5 add2(x);
6 cout << x << endl; // Prints 1
```

We use & to convert a data type into a reference.

```
1 void add2(int &x) { x += 2; }
2
3 ...
4 int x = 1;
5 add2(x);
6 cout << x << endl; // Prints 3
```

Example: grader

```
8  const int MAX_N_STUDENTS = 500;
9  int read_grades(float grades[], int max_n_students);
10
11 int main(int argc, char** argv)
12 {
13         float grades[MAX_N_STUDENTS];
14
15         int n = read_grades(grades, MAX_N_STUDENTS);
16         cout << "There are " << n << " students" << endl;
17
18         float min_grade;
19         float max_grade;
20         min_max_of(n, grades, min_grade, max_grade);
21         cout << "Minimum grade is " << min_grade << endl;
22         cout << "Maximum grade is " << max_grade << endl;
23
24         cout << "Average grade is " << average_of(n, grades) << endl;
25
26         return EXIT_SUCCESS;
27 }
```

Example: grader

```
29 int read_grades(float grades[], int max_n_students)
30 {
31         int n = 0;
32         while (n < max_n_students) {
33                 cin >> grades[n];
34                 if (!cin.good())
35                         break;
36                 ++n;
37         }
38         return n;
39 }
```

Example: grader

```
#ifndef STATS_H
#define STATS_H

void min_max_of(int n, float values[],
                float &min_value, float &max_value);
float average_of(int n, float values[]);

#endif
```

Example: grader

```cpp
#include "stats.h"

#include <limits>

using std::numeric_limits;

void min_max_of(int n, float values[],
                float &min_value, float &max_value)
{
        min_value = numeric_limits<float>::max();
        max_value = numeric_limits<float>::min();
        for (int i = 0; i < n; ++i) {
                if (values[i] < min_value)
                        min_value = values[i];
                if (values[i] > max_value)
                        max_value = values[i];
        }
}
```

Example: grader

```
20 float average_of(int n, float values[])
21 {
22         float avg = 0.0f;
23         for (int i = 0; i < n; ++i)
24                 avg += values[i];
25         avg /= n;
26         return avg;
27 }
```