C/C++ Primer for CMPT 115/117

Outline

- Assumptions
- C or C++?
  - Hello world
- Data
  - Atomic types
  - Arrays
  - Records
- Program
  - console input/output
  - if-statements
  - loops
  - functions

Assumptions

- You have programming experience comparable to CMPT 111/116
  - loops, if-statement, arrays or lists, functions/ procedures
- Some languages commonly used:
  - C
  - Java, C#
  - JavaScript
  - Python
  - Visual Basic

All computation is based on the same concepts; if you know the concepts, a new language is a minor barrier.

C or C++

- CMPT111/116 teaches “Procedural subset of C++”
  - C++ contains all of C, but adds OOP to it
- If you have learned strict C, almost everything you know is the same
- In CMPT 115/117, we’ll move into OOP
  - We do not assume it.

If you already know C

C

- stdlib
- printf(), scanf()
- structs are awkward
- includes:
  - #include <math.h>

C++

- iostream
- cout, cin
- structs are simplified
- includes
  - #include <cmath>
- namespaces

All programming constructs in C are valid in C++.

If you already know Java

Java	C++
• import	• include
• println(), stream objects	• cout, cin
• Classes, instances	• Classes, instances covered in CMPT 115 later

Most basic programming constructs in Java are valid in C++.

Hello, world

```
/* Your Name Here
   Your Student Number Here
   Assignment or Lab Information Here
   Synopsis: Our first C++ program
*/

#include <iostream>
#include <cstdlib>

using namespace std;

// main program entry point
int main(){

  cout << "Hello, world!" << endl;

  return EXIT_SUCCESS;
}
```

Variables & Assignment

```
int main(){

  // variables declared before use

  int aNumber;     // declaration

  aNumber = 7;     // assignment

  // declare and initialize
  int anotherNumber = -1;

  // declaration pattern:
  // <type> <variable name> ;

  return EXIT_SUCCESS;
}
```

Atomic Types

```
int main(){

  int aNumber = 7;
  int anotherNumber = -1;

  float pi        = 3.14159;
  double morePi   = 3.1415926583;
  float avogadro = 6.02e+23;

  char aLetter = 'D';
  char aDigit  = '7';

  bool aValue        = true;
  bool anotherValue = false;

  return EXIT_SUCCESS;
}
```

Expressions

```
int aNumber;  // declare

aNumber = 7;  // literal

// arithmetic
aNumber = (2 % 3 + 4 * 5 - 6) / 7;

// relational
bool aValue = true;  // literal

aValue = (aNumber > 5);
// more later with if-statements

// boolean
aValue = (aNumber > 5) && (aNumber < 10);
```

Expression Pitfalls

```
int anInteger;
anInteger= 7/2;  // 3

float aFloat;
aFloat = 7/2;    // 3.0
aFloat = 7.0/2;  // 3.5
aFloat = 7/2.0;  // 3.5

// integer division
int aNumber = 1/10;  // zero
```

Console input & output

```
int aNumber;  // declare

aNumber = 7;  // literal

cout << “Here’s a number: “
     << aNumber << endl;

cout << “Enter an integer: “;

cin  >> aNumber;

cout << “Here’s your number: “
     << aNumber << endl;
```

Conditionals

```
int anInteger;
cout << “Enter an integer: “;
cin  >> anInteger;

if (anInteger > 0)
{
  cout << “You seem positive!” << endl;
}
else
{
  cout << “You don’t seem positive.”;
  cout << endl;
}
```

Conditionals (no-else)

```
int anInteger;
cout << “Enter an integer: “;
cin  >> anInteger;

if (anInteger > 0)
{
  cout << “You seem positive!” << endl;
}
```

Chained Conditionals

```
int anInteger;
cout << “Enter an integer: “;
cin  >> anInteger;

if (anInteger > 0)
{
  cout << “You seem positive!” << endl;
}
else if (anInteger == 0)
{
  cout << “That means nothing to me”
       << endl;
}
else
{
  cout << “You don’t seem positive.”;
  cout << endl;
}
```

Chained Conditionals

```
int anInteger;
cout << “Enter an integer: “;
cin  >> anInteger;

if (anInteger > 0 && anInteger <= 9000)
{
  cout << “You seem positive!” << endl;
}
else if (anInteger > 9000)
{
  cout << “It’s over 9000!” << endl;
}
else
{
  cout << “You don’t seem positive.”;
  cout << endl;
}
```

Expressions useful for conditionals

- Relational operators:
  - `==  !=`
  - `<  >  <=  >=`
- Boolean oprerators:
  - **`and &&`**
  - **`or ||`**
  - **`!`**

Expression pitfall:
Assignment operator =
Equality operator ==

```
if (x=3)
{
  cout << "Yes" << endl;
}
```

- Can be used anywhere, but especially useful in loops and if-statements

Nested Conditionals

```
int anInteger;
cout << “Enter an integer: “;
cin  >> anInteger;

if (anInteger > 0)
{
  cout << “You seem positive!” << endl;
}
else
{
  if (anInteger == 0)
  {
    cout << “That means nothing to me”
         << endl;
  }
  else
  {
    cout << “You don’t seem positive.”;
    cout << endl;
  }
}
```

Pitfalls for ~~Conditionals~~ semicolons

```
int anInteger;
cout << “Enter an integer: “;
cin  >> anInteger;

if (anInteger > 0);
{
  cout << “You seem positive!” << endl;
}
```

Never put a semicolon before a block { … }

While Loops

```
int anInteger;
cout << “Enter a positive integer: “;
cin  >> anInteger;

while (anInteger <= 0)
{
  cout << “That was not positive!”
       << endl;
  cout << “Enter a positive integer: “;
  cin  >> anInteger;
}

cout << “That was positive!” << endl;
```

Condition checked before body

While Loops for counting

```
int i;

i = 0;
while (i <= 10)
{
  cout << i << endl;
  i = i + 1;
}

cout << “That was fun counting!” << endl;
```

For Loops for counting

```
int i;

for (i = 0 ; i <= 10 ; i = i + 1)
{
  cout << i << endl;
}

cout << “That was fun counting!” << endl;
```

Condition checked before body

For Loops for counting

```
for (int i = 0 ; i <= 10 ; i++ )
{
  cout << i << endl;
}

cout << “That was fun counting!” << endl;
```

Do-While Loops

```
int anInteger;

do
{
  cout << “Enter a positive integer: “;
  cin  >> anInteger;
} while (anInteger <= 0);


cout << “That was positive!” << endl;
```

Condition checked *after* **body**

Pitfalls for ~~loops~~ semicolons

```
for (int i = 0 ; i <= 10 ; i++ );
{
  cout << i << endl;
}

cout << “That was fun counting!” << endl;
```

Never put a semicolon before a block { … }

Pitfalls for loops

```
int i = 0;
while (i <= 10)
  cout << i << endl;
  i = i + 1;

cout << “That was fun counting!” << endl;
```

Always use { … } even when its optional

Functions that don't return a value

```
void countDown(float value)
{
  for (int I = value; I > 0; I--)
  {
    cout << "Here's a " << I << endl;
  }
  cout << "BOOOM!!" << endl;

  return; // optional
}


int main() {

  int x = 10;
  coutnDown(x);

  return EXIT_SUCCESS;
}
```

Functions that don't return a value

```
void countDown(float value)
{
  while (value > 0)
  {
    cout << “Here’s a “ << value << endl;
    value = value - 1;
  }
  cout << “BOOOM!!” << endl;
}

int main() {

  int x = 10;
  coutnDown(x);
  cout << x << endl;

  return EXIT_SUCCESS;
}
```

Functions that return a value

```
int askForInteger(int low, int high)
{
  int result;
  do
  {
    cout << “Enter a number between “
         << low << “ and “ << high
         << “: “;
    cin >> result;
    if (result < low || result > high)
    {
      cout << “You missed!” << endl;
    }
  } while (result < low || result > high);

  return result;
}

int main() {

  int x = askForInteger(1,10);

}
```

Arrays for lots of data

```
int numbers[100];  // declare

// initialize with a loop
for (int i = 0; i < 100; i++)
{
  numbers[i] = 0;
}

// only at declaration
int smallArray[5] = {1,2,3,4,5};

// C++ can set the size exactly
int smallish[] = {1,2,3,4,5,6,7};
```

Arrays for lots of data

```
int fibo[100];

fibo[0] = 1;
fibo[1] = 1;

for (int i = 2; i < 100; i++)
{
  fibo [i] = fibo[i-1] + fibo[i-2];
}
```

valid indices 0 to 99

Arrays and functions

```
void initFib(int fibo[], int size)
{
  fibo[0] = 1;
  fibo[1] = 1;

  for (int i = 2; i < size; i++)
  {
    fibo [i] = fibo[i-1] + fibo[i-2];
  }
}

int main()
{
  int array[100];
  initFib(array, 100);

  return EXIT_SUCCESS;
}
```

valid indices 0 to size - 1

2-D Arrays

```
char tictactoe[3][3];

// initialize to all blanks
for (int r = 0; r < 3; r++)
{
  for (int c = 0; c < 3; c++)
  {
    tictactoe[r][c] = ' '; // blank
  }
}

// place 'X' in the center square
tictactoe[1][1] = 'X';
```

3-D Arrays

```
char vulcanChessBoard[8][8][8];

// initialize to all blanks
for (int r = 0; r < 8; r++)
{
  for (int c = 0; c < 8; c++)
  {
    for (int h = 0; h < 8; h++)
    {
      vulcanChessBoard[r][c][h] = ' ';
    }
  }
}

// place 'Q'
vulcanChessBoard[0][3][0] = 'Q';
```

Records for organized data

```
struct Date {
  int day;    // 1-31
  int month;  // 1-12
  int year;
};

int main() {

  Date aDate;  // declare

  aDate.day = 12;
  aDate.month = 1;
  aDate.year = 2015;

  return EXIT_SUCCESS;
}
```

Records for organized data

```
struct Card {
  int value;   // 1-13
  char suit;   // ‘c’ ‘d’ ‘h’ ‘s’
};

int main() {

  Card aCard;  // declare

  aCard.value = 1;  // ace
  aCard.suit  = ‘s’; /spades

  Card deck[52];

  deck[0].value = 1;
  deck[0].suit = ‘c’;

  return EXIT_SUCCESS;
}
```

Development Tools

- Command-line compilation in Lab01
- Eclipse
  - Available in the lab on **all** platforms
  - **Don't** install yourself on Windows!
  - Ideal for Linux (therefore: dual boot!)
  - Mac home installation feasible
- MSVS, Xcode, CodeLite all viable but not supported