CSC 270 – Survey of Programming Languages

C++ Lecture 1 : C++ As A Better C

A First Program

```cpp
#include <iostream>

using namespace std;

int main(void)
{
    cout << "This is my first C++ program."
    << endl;
    return(0);
}
```

- **Header**: makes input and output available to us
- **Open and close braces**: mark the beginning and end
- **Statements**:
#include <iostream>

using namespace std;

int main(void)
{
    int value1, value2, value3;
    float sum, average;

    cout << "What is the first value? ";
    cin >> value1; // for comments

    cout << "What is the second value? ";
    cin >> value2;

    cout << "What is the third value? ";
    cin >> value3;

    sum = value1 + value2 + value3;
    average = sum / 3;

    cout << "Average = " << average << endl;
    return(0);
}
Comments

• Our program is a bit longer than our previous programs and if we did not know how to calculate gross pay, we might not be able to determine this from the program alone.

• It is helpful as programs get much longer to be able to insert text that explains how the program works. These are called comments. Comments are meant for the human reader, not for the computer.

• In C++, anything on a line after a double slash (//) is considered a comment.

A program that uses a character variable

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

// A very polite program that greets you by name
int main(void)
{
    string name;

    cout << "What is your name?\t";
    cin >> name;

    cout << "Pleased to meet you, " << name << endl;
    return (0);
}
```
Formatting **float** output in C++

- `cout` and `cin` are examples of what are called stream input/output.
- Stream I/O uses a set of built-in values called **format flags** to determine how data is printed. We can change these values by using `setf()`. For now, we will use it only to reformat float values.

**setf and the ios flags**

- When we wish to ensure that float is printed with a fixed decimal point and a certain number of decimal places we place:
  
  ```cpp
  cout.setf(ios::showpoint);
  cout.setf(ios::fixed);
  cout.precision(2);
  ```

  - **Guarantees that trailing zeros appear**
  - **Gives us two decimal places**
  - **Float numbers are always written in regular notation** *(not scientific notation)*
the width flag

- Every time we wish a number to be printed and to take up a fixed amount of place (not merely what it needs), we write:
  ```cpp
cout.width(15);
```
  This assures that the item being printed will occupy 15 spaces.
- Unlike the other flags changes that remain in effect once you call them, width must be reset each time you use it.

Changing the width

<table>
<thead>
<tr>
<th>Number</th>
<th>Formatting</th>
<th>Print as:</th>
</tr>
</thead>
<tbody>
<tr>
<td>182</td>
<td><code>cout.width(2)</code></td>
<td>182</td>
</tr>
<tr>
<td>182</td>
<td><code>cout.width(3)</code></td>
<td>182</td>
</tr>
<tr>
<td>182</td>
<td><code>cout.width(5)</code></td>
<td>&quot;`182&quot;</td>
</tr>
<tr>
<td>182</td>
<td><code>cout.width(7)</code></td>
<td>&quot;`182&quot;</td>
</tr>
<tr>
<td>-182</td>
<td><code>cout.width(4)</code></td>
<td>-182</td>
</tr>
<tr>
<td>-182</td>
<td><code>cout.width(5)</code></td>
<td><code>-182</code></td>
</tr>
<tr>
<td>-182</td>
<td><code>cout.width(7)</code></td>
<td>&quot;``-182&quot;</td>
</tr>
</tbody>
</table>
### Changing the width (continued)

<table>
<thead>
<tr>
<th>Number</th>
<th>Formatting</th>
<th>Print as:</th>
</tr>
</thead>
<tbody>
<tr>
<td>23</td>
<td>cout.width(1)</td>
<td>23</td>
</tr>
<tr>
<td>23</td>
<td>cout.width(2)</td>
<td>23</td>
</tr>
<tr>
<td>23</td>
<td>cout.width(6)</td>
<td>....23</td>
</tr>
<tr>
<td>23</td>
<td>cout.width(8)</td>
<td>......23</td>
</tr>
<tr>
<td>11023</td>
<td>cout.width(4)</td>
<td>11023</td>
</tr>
<tr>
<td>11023</td>
<td>cout.width(6)</td>
<td>.11023</td>
</tr>
<tr>
<td>-11023</td>
<td>cout.width(6)</td>
<td>-11023</td>
</tr>
<tr>
<td>-11023</td>
<td>cout.width(10)</td>
<td>.....11023</td>
</tr>
</tbody>
</table>

### Changing the precision

<table>
<thead>
<tr>
<th>Number</th>
<th>Width</th>
<th>Precision</th>
<th>Prints as:</th>
</tr>
</thead>
<tbody>
<tr>
<td>2.718281818</td>
<td>cout.width(8)</td>
<td>cout.precision(5);</td>
<td>`2.71828</td>
</tr>
<tr>
<td>2.718281818</td>
<td>cout.width(8)</td>
<td>cout.precision(3);</td>
<td>```2.718</td>
</tr>
<tr>
<td>2.718281818</td>
<td>cout.width(8)</td>
<td>cout.precision(2);</td>
<td>``````2.72</td>
</tr>
<tr>
<td>2.718281818</td>
<td>cout.width(8)</td>
<td>cout.precision(0);</td>
<td>`````````3</td>
</tr>
<tr>
<td>2.718281818</td>
<td>cout.width(13)</td>
<td>cout.precision(11);</td>
<td>2.71828182800</td>
</tr>
<tr>
<td>2.718281818</td>
<td>cout.width(12)</td>
<td>cout.precision(11);</td>
<td>2.71828182800</td>
</tr>
</tbody>
</table>
The revised *Compound* program

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

// Calculate the interest that the Canarsie
// Indians could have accrued if they had
// deposited the $24 in a bank account at
// 5% interest.
int main(void)
{
    const int present = 2000;
    int year;
    const float rate = 0.05;
    float interest, principle;

    // Set the initial principle at $24
    principle = 24;

    // For every year since 1625, add 5%
    // interest to the principle and print
    // out the principle

    // There has to be two fixed places for
    // the principle
    cout.setf(ios::showpoint);
    cout.setf(ios::fixed);
    cout.precision(2);
```
for (year = 1625; year < present; year++) {
    interest = rate * principle;
    principle = principle + interest;
    cout << "year = " << year ;
    // Use 15 places for printing the principle
    cout << "\tprinciple = ";
    cout.width(15);
    cout << principle << endl;
} return(0);}

exit()

• exit() allows the user to let a program terminate if the program detects an unrecoverable error.

• The statement
  
  #include <cstdlib>
  
has to be included.

• A non-zero status value should be returned when the program terminates abnormally.
What Are References Parameters?

- Reference parameters do not copy the value of the parameter.
- Instead, they give the function being called a copy of the address at which the data is stored. This way, the function works with the original data.
- We call this **passing by reference** because we are making references to the parameters.

Rewriting **power**

- We can make the **power** function tell the main program about the change in y by placing an ampersand (&) between the data type and variable name:
  ```
  void power (float &y, float x, int n) {
  ... ... ...
  }
  ```
#include <iostream>
using namespace std;

void power(float &y, float x, int n);

// A program to calculate 4-cubed using a
// function called power
int main(void) {
  float x, y;
  int n;

  x = 4.0;
  n = 3;
  y = 1.0;
  power(y, x, n);
  cout << "The answer is " << y << endl;
}

// power() - Calculates y = x to the nth power
void power(float &y, float x, int n) {
  y = 1.0;
  while (n > 0) {
    y = y * x;
    n = n - 1;
  }
  cout << "Our result is " << y << endl;
}
// Find the average of three numbers using a function
int main(void) {
    int value1, value2, value3;
    float mean;

    // Get the inputs
    getvalue(value1);
    getvalue(value2);
    getvalue(value3);

    // Call the function that calculates the average and then print it
    mean = find_average(value1, value2, value3);
    cout << "The average is " << mean << endl;
}

// getvalue() - Input an integer value
void getvalue(int &x) {
    cout << "Enter a value ?";
    cin >> x;
}

// find_average() - Find the average of three numbers
float find_average(int x, int y, int z) {
    float sum, average;
    sum = (float) (x + y + z);
    average = sum / 3;
    return average;
}
Enumeration Constants in C

- Instead of writing
  
  ```c
  #define NO 0
  #define YES 1
  ```
  
  we can write
  ```c
  enum boolean {NO, YES};
  ```
  
- The first value is defined as 0, and each subsequent value is one greater, unless we explicitly state a value.

  ```c
  enum escapes {BELL = '\a', BACKSPACE = '\b',
  TAB = '\t', NEWLINE = '\n', VTAB = '\v'};
  enum months {JAN = 1, FEB, MAR, APR, MAY, JUN,
  JUL,AUG, SEP, OCT, NOV, DEC }
  /* FEB is 2, MAR is 3, etc. */
  ```

enum in C++

- In C, the keyword `enum` must appear before the type when declaring variables:

  ```c
  enum days {sun, mon, … , fri, sat};
  enum days today;
  ```

- This is NOT necessary in C++:

  ```c
  days today;
  ```
struct in C++

- Unlike C, the keyword struct is not required when declaring variables of a struct type:
  ```cpp
  struct myDataType  
  {
  
  }
  
  myDataType myData;
  ```

#define

- In C we have used `#define` to define constants. The preprocessor replaces each occurrence of the name with its value.
- We can also use it to define simple functions
  ```cpp
  #define sqr(x) ((x)*(x))
  ```
- We include parentheses to ensure its correct translation by the preprocessor.
**const**

- The compiler sees the program after the preprocessor is finished, so the error messages reflect the preprocessed code, not the original source code.
- The `const` statement ensures that the compiler produces error messages that reflect what was originally written:
  - `const float pi = 3.14159;`

**inline functions**

- Functions call can take a large amount of time to execute.
- Inline functions avoid this by inserting the code for the function in the object code.

```c
inline float abs(x)
{
    return (x>=0)? x: -x;
}
```

- Because an inline function's code is physically inserted in the program each time its called, we limit its use to small functions.