Introduction to C++ (*and C*)

Chapter 2

function

• basic building block of all C and C++ programs
• a module of code which performs a task
• every program must have a function called *main*
• execution begins and ends in *main*

Parts of a C++ Program

```cpp
// sample C++ program
#include <iostream>
using namespace std;
int main()
{
    cout << "Hello, there!";
    return 0;
}
```

Parts of a C++ Program

```cpp
// sample C++ program
#include <iostream>
using namespace std;
int main()
{
    cout << "Hello, there!";
    return 0;
}
```

int main ( )
{
    return 0;
}

Now look at an example.

```cpp
// sample C++ program
#include <iostream>
using namespace std;
int main()
{
    cout << "Hello, there!";
    return 0;
}
```
### int main ( )

- **return-type** function-name (parameters)
  - ( ) show it is a function
  - empty ( ) say no parameters are received
  - int says one integer value is returned
  - main is the name of this function

```c
int main () {  return 0; }
```

### void main ( )

- void return type says no value is returned
- don’t put a return statement if void is the return type

```c
void main () { }
```

### Special Characters

<table>
<thead>
<tr>
<th>Character</th>
<th>Name</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>//</td>
<td>Double slash</td>
<td>Beginning of a comment</td>
</tr>
<tr>
<td>#</td>
<td>Pound sign</td>
<td>Beginning of preprocessor directive</td>
</tr>
<tr>
<td>&lt;&gt;</td>
<td>Open/close brackets</td>
<td>Enclose filename in #include</td>
</tr>
<tr>
<td>()</td>
<td>Open/close parentheses</td>
<td>Used when naming a function</td>
</tr>
<tr>
<td>{}</td>
<td>Open/close brace</td>
<td>Encloses a group of statements</td>
</tr>
<tr>
<td>&quot; &quot;</td>
<td>Open/close quotation marks</td>
<td>Encloses string of characters</td>
</tr>
<tr>
<td>;</td>
<td>Semicolon</td>
<td>End of a programming statement</td>
</tr>
</tbody>
</table>

### The cout Object

- Displays information on computer screen - console output
- Uses << to send information to cout – called insertion operator
  - cout << "Hello, there!");
- Can be used to send multiple items to cout:
  - cout << "Hello, " << "there!"
  - Or:
  - cout << "Hello, ";
  - cout << "there!";

```c
int main () {  return 0; }
```
The cout Object

- To get multiple lines of output on screen:
  - Use `endl`
    ```cpp
    cout << "Hello, there!" << endl;
    ```
  - Called a manipulator
  - No " " around `endl`
  - L at the end is a lowercase L and not the number 1
  - Use `\n` in output string
    ```cpp
    cout << "Hello, there!\n";
    ```
  - \n is called an escape sequence – new line
  - Notice that it is inside " "

Now look at an example.

```cpp
// The cout Object

int main() {
    cout << "Welcome";
    cout << "Students";
    cout << "Welcome\nStudents";
    cout << "Welcome" << \n << "Students";
    cout << "Welcome" << '\n' << "Students";
    cout << "Welcome" << endl << "Students";
    return 0;
}
```

The #include Directive

- Inserts the contents of another file into the program
- Preprocessor directive, not part of C and C++ languages
- #include lines not seen by compiler
- Do not use ; at end of #include line

```cpp
#include <iostream.h>     // in C++
#include <iostream>       // The method we'll use.
#include <stdio.h>        // in C or C++
#include "a:myheader.h"  // Another method we'll use.

int main() {
    using namespace std;
    cout << "Hello, there!\n";
    return 0;
}
```

Variables and Constants/Literals

- Variable: storage location in memory
  - Has a name and a type of data it can hold
  - Must be defined before it can be used:
    ```cpp
    int item;
    ```
- Constant or Literal: item whose value does not change during program execution – value is written in the program code
  - "hello, there" (string constant)
  - 12 (integer constant)

Identifiers

- Programmer-chosen names to represent parts of the program: variables, functions, etc.
- Name should represent the use of the identifier
- Consist of sequence of letters, digits, and underscore (_)
- Must begin with letter or __
- Upper- and lower-case characters are different (case sensitive)
- Cannot use C/C++ key words as identifiers
C++ Key Words

You cannot use any of the C++ key words as an identifier. These words have reserved meaning.

<table>
<thead>
<tr>
<th>Identifier</th>
<th>Valid?</th>
<th>Reason if invalid</th>
</tr>
</thead>
<tbody>
<tr>
<td>totalSales</td>
<td>Yes</td>
<td></td>
</tr>
<tr>
<td>total_Sales</td>
<td>Yes</td>
<td></td>
</tr>
<tr>
<td>totalSales</td>
<td>No</td>
<td>Cannot contain .</td>
</tr>
<tr>
<td>4thQtrSales</td>
<td>No</td>
<td>Cannot begin with digit</td>
</tr>
<tr>
<td>totalSales$</td>
<td>No</td>
<td>Cannot contain $</td>
</tr>
</tbody>
</table>

Valid and Invalid Identifiers

<table>
<thead>
<tr>
<th>Identifier</th>
<th>Valid?</th>
<th>Reason if invalid</th>
</tr>
</thead>
<tbody>
<tr>
<td>totalSales</td>
<td>Yes</td>
<td></td>
</tr>
<tr>
<td>total_Sales</td>
<td>Yes</td>
<td></td>
</tr>
<tr>
<td>totalSales</td>
<td>No</td>
<td>Cannot contain .</td>
</tr>
<tr>
<td>4thQtrSales</td>
<td>No</td>
<td>Cannot begin with digit</td>
</tr>
<tr>
<td>totalSales$</td>
<td>No</td>
<td>Cannot contain $</td>
</tr>
</tbody>
</table>

Are they valid or invalid?

- grossAmt
- netPay
- Dollar
- 3id
- int
- int_value
- am_i_1
- line_count
- hourlyPayRate
- new name
- newname
- newName
- NEWNAME
- new_name
- NewName
- newname$
- print
- accumulator

Naming conventions

- use meaningful names
  - payRate
  - student_name
- use lower-case or mixed upper/lower
  - payrate
  - payRate
- some use prefix to identify type
  - iAge
  - cSex

Data Types

- char - holds one character
- not a string
- int - holds one integer (whole number)
- digits + - (no fractional part)
- float - holds one number which can have a decimal
- digits + - .
- double - holds one number which can have a decimal
- more significant digits than a float

Data types & sizes FOR OUR SYSTEM:

- char - 1 byte - 8 bits
- int - 4 bytes - 32 bits
- float - 4 bytes - 32 bits
- double - 8 bytes - 64 bits
Integer Data Types

- Integer variables can hold whole numbers such as 12, 7, and -99.

<table>
<thead>
<tr>
<th>Data Type</th>
<th>Size</th>
<th>Range</th>
</tr>
</thead>
<tbody>
<tr>
<td>short</td>
<td>2 byte</td>
<td>-32,768 to +32,767</td>
</tr>
<tr>
<td>unsigned short</td>
<td>2 byte</td>
<td>0 to 65,535</td>
</tr>
<tr>
<td>int</td>
<td>4 byte</td>
<td>-2,147,483,648 to +2,147,483,647</td>
</tr>
<tr>
<td>unsigned int</td>
<td>4 byte</td>
<td>0 to 4,294,967,295</td>
</tr>
<tr>
<td>long</td>
<td>4 byte</td>
<td>-2,147,483,648 to +2,147,483,647</td>
</tr>
<tr>
<td>unsigned long</td>
<td>4 byte</td>
<td>0 to 4,294,967,295</td>
</tr>
</tbody>
</table>

Defining Variables

- All variables must be defined
- Variables of the same type can be defined
  - On separate lines:
    ```
    int length;
    int width;
    ```
  - On the same line:
    ```
    int length, width;
    ```
- Variables of different types must be in different definitions

```c
int length; // count of grades
int sum; // sum of grades
```

Variable Assignments and Initialization

- Initialize a variable: assign it a value when it is defined:
  ```
  int length = 12;
  ```
- Can initialize some or all variables:
  ```
  int length = 12, width = 5, area;
  float pi = 3.14;
  char answer = 'Y';
  ```

Rules for definitions

- All variables must be defined before used
- Size is determined by the type
- Memory is reserved
- In C but not in C++ ...
definitions must be placed at the beginning of a block    {    }

Floating-point Constants

- Can be represented in
  - Fixed point (decimal) notation:
    ```
    31.4159
    0.0000625
    ```
  - E notation:
    ```
    3.14159E1
    6.25E-5
    ```
- Are double by default
- Can be forced to be float (3.14159f) or long double (0.0000625L)
The `char` Data Type

- Used to hold characters or very small integer values
- Usually 1 byte of memory
- Numeric value of character from character set is stored in memory:

```
CODE:    MEMORY:
char letter;       letter = 'C'; 67
```

- Character constants - a single character in single quotes
  - ‘a’
  - ‘A’
  - ‘5’
  - ‘%’
  - “A” NO
  - “53” NO

Character Strings

- Can store a series of characters in consecutive memory locations: "Hello"
- Stored with the null terminator, \0, at the end:

```
HELLO \0
```

- Characters between the "    "

Character Strings

- "Southwest Tennessee Community College"
- "John Doe"
- "%"
- "$6.54"
- "int"
- "3.14"

The `bool` Data Type

- Represents values that are `true` or `false`
- `bool` variables are stored as small integers
- `false` is represented by 0, `true` by 1:

```
bool allDone = true;
bool finished = false;
```

Families of data

- `char`
- `int`
- `bool`
- `float`
- `double`
Variable Assignments and Initialization

- Uses the = operator (called the assignment operator)
- = is not a test for equality
- Has a single variable on the left side (referred to as an lvalue)
- Value (constant, variable, or expression) on the right side
- Copies the value on the right into the variable on the left:

```
int numStudents;
float payRate;
char mi;
```

```
numStudents = 20;
payRate = 7.50;
mi = 'E';
```

```
numStudents = twenty;
payRate = 7;
mi = “E”;
```

```
numStudents = “20”;
payRate = 750;
mi = “E”;
```

```
umStudents = 20.6;
payrate = 7.5000;
```

### Binary Arithmetic Operators

<table>
<thead>
<tr>
<th>SYMBOL</th>
<th>OPERATION</th>
<th>EXAMPLE</th>
<th>VALUE OF ans</th>
</tr>
</thead>
<tbody>
<tr>
<td>+</td>
<td>addition</td>
<td>ans = 7 + 3;</td>
<td>10</td>
</tr>
<tr>
<td>-</td>
<td>subtraction</td>
<td>ans = 7 - 3;</td>
<td>4</td>
</tr>
<tr>
<td>*</td>
<td>multiplication</td>
<td>ans = 7 * 3;</td>
<td>21</td>
</tr>
<tr>
<td>/</td>
<td>division</td>
<td>ans = 7 / 3;</td>
<td>2</td>
</tr>
<tr>
<td>%</td>
<td>modulus</td>
<td>ans = 7 % 3;</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>No exponentiation</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

### / Operator

- (division) operator performs integer division if both operands are integers
  - cout << 13 / 5; // displays 2
  - cout << 91 / 7; // displays 13
- If either operand is floating point, the result is floating point
  - cout << 13 / 5.0; // displays 2.6
  - cout << 91.0 / 7; // displays 13.0

### % Operator

- (modulus) operator computes the remainder resulting from integer division
  - cout << 13 % 5; // displays 3
- % requires integers for both operands
  - cout << 13 % 5.0; // error

Comments

- Used to document parts of the program
- Intended for persons reading the source code of the program:
  - Indicate the purpose of the program
  - Describe the use of variables
  - Explain complex sections of code
- Are ignored by the compiler
C++ Style Comments

Begin with // through to the end of line:
int length = 12; // length in inches
int width = 15; // width in inches
int area; // calculated area

// calculate rectangle area
area = length * width;

C-Style Comments

• Begin with /*
• End with */
• Can span multiple lines:
  /* this is a multi-line C-style comment */
• Can be used like C++ style comments:
  int area; /* calculated area */

Stream Input/Output

/* this is a comment */
int main ()
{
  /* another very, very long comment which required multiple lines */
  cout << "Hello world"; /* a comment */
  cout << "Goodbye"; // another one
  return 0;
  // This kind requires
  // a new // for each new line of comment
}