Data Types 1

C++ Syntax and Semantics

Elements of C++ Programs

- **In C++, subprograms are referred to as functions**
- **EVERY C++ program MUST HAVE a function named main**
- **main is called by the operating system**
- **All other functions are called by main**
- **When main calls (invokes) a function, that function starts executing. When the function completes execution, control is returned to main**

- **Braces '{' and '}' indicate the beginning and end of the statements to be executed by a function. These statements are called the *body* of the function**

- **Functions may or may not return values**

Syntax and Semantics

- **Syntax (grammar) - Formal rules governing how valid instructions are written**
- **Semantics (meaning) - Set of rules determining the meaning of the instructions**
- **Metalanguage - Language used to write the syntax rules for another language. Only shows how to write instructions that the compiler can translate. It does not define what those instructions do (their semantics)**

Syntax Templates

- **In this book we write the syntax rules for C++ using a metalanguage called a *syntax template***
- **A *syntax template* is a generic example of the C++ construct being defined**
- **Graphic conventions show which portions are optional and which can be repeated**
  - Boldface word or symbol - literal word or symbol in C++
  - Non-boldfaced word - replaceable by another template
  - Curly Brace indicates a list of items, from which one item can be chosen
  - Shading indicates an optional part of the definition

Syntax Template Example

SYNTAX TEMPLATE: MainFunction

```
int main( )
{
    Statement
       ⋮
    return 0;
}
```

- **here 'int' and 'main' are reserved words in C++**
- **the ( ) is required in the syntax template for the main function**
- **0 or more statements can be placed in between the braces;**
- **however since main is a value returning function, it requires one line: return 0;**
- **left brace indicates the start of the statements, and the right brace indicates the end of those statements**

Naming Program Elements

- ***Identifiers*** **are used in C++ to name items (i.e. sub-programs and variables)**
  - Consist of Letters, Numbers or the underscore (_)
  - Must start with either a letter or the underscore
  - Examples of valid identifiers:

    `abc`
    `a1a2a3a4`
    `A_b_c_D`

  - Examples of invalid identifiers:

    `2abc` {cannot start with a number}
    `t%h` {can only use the _ character, numbers and letters}
    `u--5` {same as above, and - is a math symbol in C++}
- ***Reserved Word*** **- words that have a specific use in C++. These cannot be used as programmer defined identifiers**

Identifiers

- **Pick identifier names that are meaningful**
  - Use ‘mileage’ instead of ‘m’
  - Uppercase letters are different than lowercase
  - Make identifiers easy to read
    `go_to_store` **or** `GoToStore` **versus**
    `gotostore` **or** `gOtOsTORE`

Data and Data Types

- ***DataType*** **- a specific set of data values, along with a set of operations on those values**
- **Some Standard (built in) types:**
  - int - for integer numbers
  - float, double for numbers with decimals
  - char - for single characters, i.e. ‘a’, or ‘z’
  - string - for sequences of characters, i.e. “Hello”

- **The *char* DataType: contains one alphanumeric character - a letter, digit or special character. Characters are enclosed within single quotes ‘’. A blank is a valid character ‘ ‘**
- **The *string* DataType : A sequence of characters enclosed in double quotes. “This class is great!”, “Hello”**
- **The *null string* contains no characters. Written using “”**

Built in Simple Types

- **DataType - a specific set of data values (the domain) along with a set of operations on those values**
- **Simple Data Types - DataTypes where the domain is made up of indivisible data values (data values that do not have components that can be accessed individually)**
- **Strings are not simple DataTypes because each character in a string can be accessed - strings are a collection of characters**

- **Sizes of a variable are measured in multiples of the size of a char. The size of a char is 1 byte. Can use the integer value returning `sizeof(SomeType)` operator to obtain the number of bytes allocated for a specific DataType - this includes structures**

Working with Character Data

- **Char is defined to be an integral type, and has size of 1 byte. Char values are stored as integer numbers on the computer.**
- **Each computer uses a particular character set. Two common sets are ASCII(128 characters) and EBCDIC(256 characters)**
- **Computer cannot tell the difference between character data and integer data in memory**

```
int someint = 97;
char somechar = 97;
cout << someint << " " << somechar << endl;
```

97 a

- **DataType of a variable determines how it will be printed ←**

Comparing Characters

- **Can use <, <=, >=, > for comparisons. Need to consider character sets. In ASCII, lowercase letters occupy 26 consecutive positions in the character set. In EBCDIC, they don't.**
- **Better to take advantage of the is… functions in the header file cctype. (i.e. islower(char)) #include <cctype>**

- **To convert char numbers to real numbers: '3' - '0' = 3. Can use a char to obtain an input and then make the conversion**
- **Also in cctype are the functions toupper(char) and tolower(char) to change the case of a char accordingly**
- **Can access individual characters of a string by giving its position number in square brackets. The first character is in position 0:**
  - StringObject[Position]

See program DataTypes1_01.cpp for examples involving char variables

Naming Elements: Declarations

- **Identifiers can be used to name both constants and variables**
- **A *declaration* associates an identifier with a data object, function or data type**
- **In C++ every identifier must be declared before it can be used**
- **Identifiers must be unique - different from all others in the program**
- ***Variable* - a location in memory, referenced by an identifier, that contains a data value that can be changed**
- **SYNTAX TEMPLATE: VariableDeclaration (ends with a ";")**

  DataType Identifier, Identifier, …;
- **A literal value is the actual value of a constant in a program**
- ***Named Constant* - another way of representing a literal value**
- **SYNTAX TEMPLATE: ConstantDeclaration**

  const DataType Identifier = LiteralValue;

Declaration Examples

- **Variable Declarations**

```
int       loop_index, index; // looping indices
float     my_average;  // average of some item
char      middle_init; // middle initial of a name
```

- **Constant Declarations (usually indicated by all CAP identifiers)**

```
const float    PI = 3.14159;
const int      LEGAL_AGE = 21;
const char     BLANK = ' ';
const string   ERROR = "Error in program";
```


Matters of Style

- **C++ is case sensitive. Uppercase letters ARE different from lower case letters. Name is different from name**
- **Use a consistent style throughout the program**
- **Typically constant identifiers are all capitals with words separated by the underscore. i.e. MAX_LENGTH, ERROR. However, this style may not always be the case.**

- **Read Matters of style pages 56 -57 - book conventions**
  - functions - All words in the name start with a capital
    FindMyPay(hours, rate)
  - variable identifiers - All words except the first start with a capital
    getTestScore

Executable Statements

- **The value of a variable can be set or changed through an *assignment statement*: my_name = "Ron Bowman";**

- **SYNTAX TEMPLATE: AssignmentStatement (ends with a ";")**

  Variable = Expression;

  - This statement replaces any previous value in the variable with the value of the expression
  - Only one variable can be on the left hand side

- **An *expression* is an arrangement of identifiers, literals and operators that can be evaluated to compute a value of a given type**

Valid and Invalid Char and String Assignments

- **Given the following declarations:**

```
string      class;
string      instructor;
string      location;
char        grade, term;
```

- **Valid assignments**

```
class = "CPE112";
location = class;
grade = 'A';
term = grade;
instructor = "Ron";
```

Invalid assignments

```
class = CPE112;
location = grade;
grade = "B";
term = class;
"Ron" = instructor;
```

String Expressions

- **Cannot perform arithmetic on strings**
- **One string operator is *concatenation* indicated by the plus ("+") sign. Concatenation combines two or more strings**
- **Concatenation works with named string constants, literal strings, char data and string variables provided at least one of the operands(for each "+") is a string variable or a named string constant**
- **Examples**

```
string first_name = "Ron", last_name = "Bowman";
string name;
const char  SPACE = ' ';
name = "Ron" + " Bowman";   // not valid
name = first_name + " Bowman"; // Valid
name = first_name  + SPACE + "Bowman"; // Not Valid
```


Output

- **To display information on standard output, use `cout` and the *insertion operator* "<<"**

```
cout <<  "hello";
```

- **`cout` is predefined in C++ systems to denote an output stream**
- **the insertion operator can be used multiple times on the same line as we have seen from previous examples**

- **SYNTAX TEMPLATE: OutputStatement (ends with a ";")**

```
cout << Expression << Expression  . . . ;
```

- **To include a " (quote mark) in an output, use \"**

```
cout << "al \"the rock\" smith"<<endl;
```

prints out: al "the rock" smith

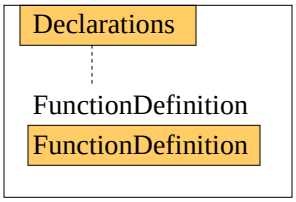
- **Use *endl* to provide a return character to go to the next line**

Program Construction

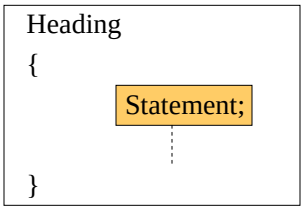
- **C++ programs are made up of functions, one of which must be named main. All functions (except main) must have a declaration**
- **A function definition consists of the function heading and its body. The body is delimited by the left and right braces.**

- **SYNTAX TEMPLATES:**

Program



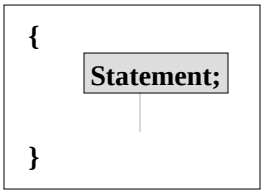
FunctionDefinition



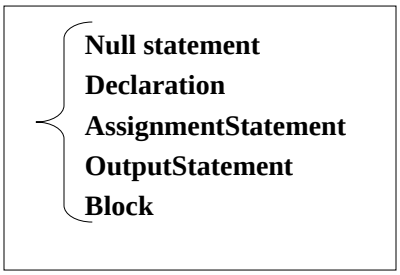
Compound Statements and Their Syntax

- **The body of a function is an example of a *block* or *compound statement***
- **SYNTAX TEMPLATES:**

Block



Statement



FunctionDefinition

Compound Statements (continued)

- **Blocks can be used wherever a single statement is allowed**
- **The null statement is just a semicolon ";"**
- **Terminate each statement except a compound statement (block) with a semicolon. That is the right brace "}" is not followed by a semicolon (however, putting one there does not affect anything)**

- **A declaration can appear anywhere an executable statement can since a declaration is officially considered to be a statement**

Various Information: pages 67 - 68

- **All C++ programs go through a preprocessor**
- **Any line beginning with a "#" sign is not considered to be a C++ statement and is not terminated in a ";"**
- **#include statements are handled by the preprocessor, and the contents of the file specified (a header file) is inserted into the source program in place of the #include statement**
- **Header files have the suffix .h - though standard header files no longer use the .h suffix**

- **using namespace std; - This statement is put at the top of the program before the 'main' function. This statement makes all the identifiers in the std namespace available to the program**

Namespaces

- **In the line: `using namespace std;` using is a directive, and namespace is another way of saying scope. In this line, `std` is the identifier that indicates the scope to use**

- **Namespace definition:**

```
namespace name_of_scope
{
        statements
}
```

- **name_of_scope is an identifier for the namespace being created**
- **pages 369-372**

Namespaces (continued)

- **Identifiers declared in a namespace can be accessed outside the body (scope) of the namespace by one of three methods only**
  - a) Using a qualified name – every time an identifier is used, the namespace must be specified as well (i.e. std::cout…).
  - b) Using a declaration - A declaration is made specifying one identifier to use from a namespace. Identifier is available within the scope of the declaration (i.e. using std::cout;).
  - c) Using a directive - all identifiers are available, but only within the scope in which it appears. We have been placing "using namespace std;" at the top of the program for global coverage

See DataTypes1_02.cpp for Examples

More Output Information

- **Blank lines**
  - In the C++ program, blank lines can be used for making the code readable.
  - To add blank lines to the output of a program, a new line must be output using cout

CODE	OUTPUT
cout << "Hi, " ;	Hi, what is your name?
cout <<"what is your name?" << endl;	
	Another way to end a line.
cout << endl;	This is on the next line.
cout << "Another way to end a line.\n";	
cout << "This is on the next line." << endl;	

- **\n is an escape sequence indicating a new line should be started**

Still More Output

- **Multiple endl manipulators or \n (new_line) escape sequences will insert multiple blank lines into the output**

```
cout << endl << endl << endl;   // results in 3 blank lines
cout << "\n\n\n";               // also results in 3 blank lines
```

- **Other escape sequences (denoted by the \ ). Use these within the "" in a cout statement:**

  \t - horizontal tab, \r - carriage return[1], \" - print out a "

  \' - print out a ' , \\ - print out a \, \n - generates a new line

- **To print out blanks, they must be enclosed in quotes**

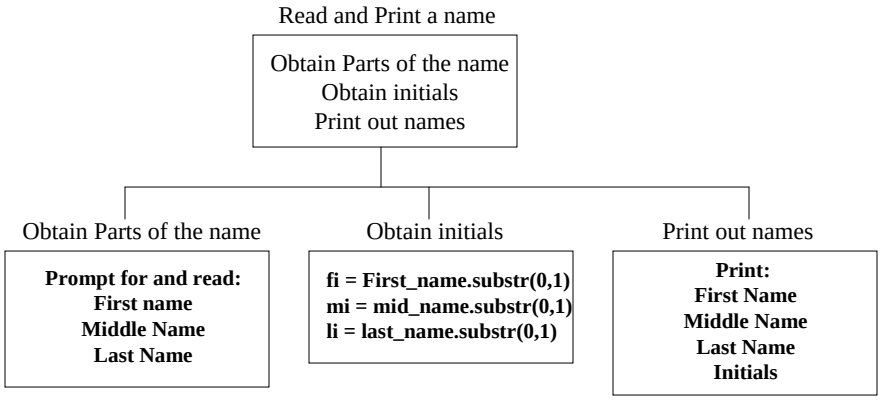
```
cout << "1234     567" << endl;      gives: 1234    567
cout << "1234" <<    "567" << endl;  gives: 1234567
```

[1]Carriage return places the cursor at the beginning of a line, it does not provide for a new line

Functional Decomposition Example

- **Write a functional decomposition for a program that is to read in a person's name in the format First, Middle, Last. The program is to then print out each name on an individual line, and add a fourth line that contains the initials for the name. Page188 – problem #1**
- **Program to read a name – Level 0**
  - Obtain parts of the name – Abstract Step for Level 0
    - Prompt for and read first name – Level 1
    - Prompt for and read middle name – Level 1
    - Prompt for and read last name – Level 1
  - Obtain initials – Abstract Step for Level 0
    - Determine initial for first name using substr function – Level 1
    - Determine initial for middle name using substr function – Level 1
    - Determine initial for last name using substr function – Level 1
  - Print out names – Abstract Step for Level 0
    - Print first name – Level 1
    - Print middle name – Level 1
    - Print last name – Level 1
    - Print out initials – Level 1

Example Continued



Sample Program Analysis

```
//*************************************************************
// PrintName program
// This program prints a name in two different formats
//*************************************************************
#include <iostream>
#include <string>
using namespace std;
const string FIRST = "Herman"; // Person's first name
const string LAST = "Smith"; // Person's last name
const char MIDDLE = 'G'; // Person's middle initial
int main()
{
      string firstLast; // Name in first-last format
      string lastFirst; // Name in last-first format
      firstLast = FIRST + " " + LAST;
      cout << "Name in first-last format is " << firstLast << endl;
      lastFirst = LAST + ", " + FIRST + ", ";
      cout << "Name in last-first-initial format is ";
      cout << lastFirst << MIDDLE << '.' << endl;
      return 0;
}
```


PrintName Example – Part 1

Statements:

```
//*******************************************************
// PrintName program
// This program prints a name in two different formats
//*******************************************************
```

Output: <no visible output>

Comments:

- Program header describes overall function of program PrintName
- All characters following // are ignored by compiler

PrintName Example – Part 2

Statements:
```
#include <iostream>
#include <string>
using namespace std;
```

Output: <no visible output>

Comments:

- #include instructs the preprocessor to add the contents of the files named iostream and string to our source program before compilation. The last line tells the computer to look in the standard namespace for various function declarations.

PrintName Example – Part 3

Statements:
```
const string FIRST = "Herman"; // Person's first name
const string LAST = "Smith"; // Person's last name
const char MIDDLE = 'G'; // Person's middle initial
```

Output: <no visible output>

Comments:

- Declaration statements which define the constants FIRST, LAST, and MIDDLE
- Comments explain what each constant represents

PrintName Example – Part 4

Statements: `string firstLast; // Name in first-last format`
`string lastFirst; // Name is last-first format`

Output: <no visible output>

Comments:

- Declarations of two string variables, firstLast and lastFirst
- Comments describing what these variables represent
- All characters following // up to the end of the line are ignored by compiler

PrintName Example – Part 5

Statements:

```
firstLast = FIRST + " " + LAST;
cout << "Name in first last format is " << firstLast << endl;
```

Output: Name in first last format is Herman Smith

Comments:

- Concatenates three strings, FIRST, " ", and LAST and assigns the resulting value to firstLast
- cout and << operator used to display phrase and value of firstLast
- endl tells computer that any future output will begin on the following line

PrintName Example – Part 6

Statements:
```
lastFirst = LAST + ", " + FIRST + ", ";
cout << "Name in last first format is ";
cout << lastFirst << MIDDLE << '.' << endl;
```

Output: Name in last first format is Smith, Herman, G.

Comments:

- Concatenates four strings, LAST, ", ", FIRST, and ", " and assigns the resulting value to lastFirst
- cout and << operator used to display phrase
- Second cout used to display value of firstLast and MIDDLE, follow by a period
- endl tells computer that any future output will begin on the following line

PrintName Example – Part 7

Statement:
```
return 0;
```

Output: <no visible output>

Comments:

- Returns integer value back to operating system
- This integer value is called the Exit Status
- In this case, 0 is returned to indicate successful completion of this program