CH 2: Introduction to C++

2.1: Parts of a C++ program (Refer pg 3 of 12 of CH1 notes)

Comments

- Can be in any place of the program
- Starts with _____ or between _____
- Compiler ignores everything from // to the end of the line.
- Comments are not required to run a program.
- Comments help explain what's going on.

Preprocessor directives

- Starts with _____
- Preprocessor reads the program to find for lines beginning with #.
- #include directive includes the content of another file.
- The name of the file is mentioned within angular brackets. E.g. #include <iostream> - iostream allows to display output on the screen and read the input from the keyboard.

Namespaces

- Used to organize the names of program entities.
- A common namespace is std; it has the entities in iostream.

Beginning of a function

- Simply, a function is a group of statements.
- _____ are used to show that it is a function.
- *main* function is the main function of a program; the starting point; the name of the function is main; *int* is the return type. It says that the function returns an integer to the operating system when it is done.

Open and closing brace.

- Use to group statements.
- Must have matching pairs.
- Used in functions and some other places as well (will learn later).

Statements

- Several type of statements are there. output statements are used very often.
    - << is an operator
    - cout is a name of a variable (an object)
- ; marks the end of the statement. Not every line needs a ;

Return statement

- sends the value 0 to the operating system., zero usually indicates that a program executed successfully. The type of the returned data is the data type mentioned in the function definition.

Special characters:

//	.....
#	.....
<>	.....
()	.....
{}	.....
“”	.....
;	.....
<<	.....

2.2 The cout object

- Use the cout object to display information on the computer's screen.
- The simplest type of screen output a program can display is console output.

cout statements that gives the equivalent outputs:

```
cout << "Programming is great fun";
```

Or

```
cout << "Programming is " << "great fun";
```

Or

```
cout << "Programming is "
     << "great fun";
```

Or

```
cout << "Programming is ";
cout << "great fun";
```

- stream insertion operator: _____
- Stream manipulator: endl (stands for end Line)– advances the _____ to the beginning of the next line. (similar to hitting enter key on output)
- Escape sequence: "\n" - advances the OUTPUT to the beginning of the next line. (similar to hitting enter key)
- Difference:
    - endl is typed with no quotations; it stands alone.
    - \n – is typed within quotations. Embed the commands within the string itself.

```
cout << "one " << "two " << "three" << endl;
```

output:

.....

```
cout << “one ” << endl << “two ” << endl << “three” << endl;
```

output:

.....

.....

.....

.....

```
cout << “one \n” << “two \n” << “three” << endl;
```

output:

.....

.....

.....

.....

```
cout << “one” << endl;
cout << “two” << endl;
cout << “three” << endl;
```

output:

.....

.....

.....

.....

```
cout << “one”;
cout << “two”;
cout << “three” <<endl;
```

output:

……

……

……

……

Common escape sequences:

\n	……
\t	……
\a	……
\b	……
\r	……
****	……
\’	……
\”	……

Note: When type escape sequences, no spaces between the backslash and the control character.

An escape sequence is stored in memory as a single character.

2.3 Include directive

e.g., #include <iostream>

- *iostream* is needed to use the cout and cin objects.
- Preprocessor directives are commands to the preprocessors which runs prior to the compiler.

e.g., #include <string>

- *string* is needed to use the string objects.

Note: no semicolon at the end of processor directives.

2.4: Variables and literals

_____: represent storage locations in the computer's memory.

_____: are constant values that are assigned to variables; piece of data that is directly written into a program.

e.g., number = 5;

- The whole line : .....
- ***number*** : .....
- ***5*** : .....

e.g., cout << "Your discount is: " << discount*;*

- The whole line : .....
- discount : .....
- Your discount is:.....

Variable definition:

e.g., int number;

- consists of the **_____** it will hold and the **_____**.
- You must have a variable definition (_____ it is used) for every variable you intend to use in a program.
- Variable definition ends with a semicolon.

Review Programming Style Guide (pg 5 of 8):

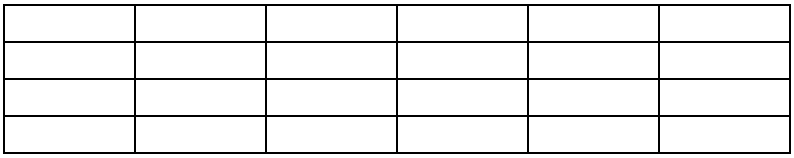
Assignment:

e.g.,

int number;

number = 5;

Memory Layout



number = "5";	// tries to assign a string literal 5 to the variable *number*.
Does not work; compilation error!	

Vs

number = 5;	// assign number 5 to the variable *number*.
Works fine	

cout << number;	// display the value of the variable - *number*.
Output:	

Vs

cout << "number";	// display the literal: *number*.
Output:	

2.5 Identifiers

_____: A programmer defined name.

_____: reserved words in C++; not allowed to use for user defined identifiers.

list of keywords: Refer Table 2.4

Name should always give an indication of what the values in variables are used for.

e.g.,

```
int x     // bad naming.
```

Vs

```
int itemsOrdered        // good
naming
```

- No spaces can be in names.
- Naming styles:
  - Snake case
  - Camal case

Review Programming Style Guide (pg 1 of 8):

<u>legal identifier</u>

1. The first letter must be _____ (uppercase or lowercase) or an _____.
2. After the first character you may use the _____(both uppercase and lowercase), _____ (0 through 9), and/or _____.
3. Uppercase and lowercase characters are distinct.

e.g.,

Variable name	Legal?		If Legal, Naming Convention	
	L/ IL	Why?	Ok/not ok	Why?
num1				
Num1				
1stNum				
firstNum				
_1stNum				
dollars$				
99bottles				
R&d				
salesFigureForYear98				
grade_report				

Data types:

1. Integer data Types : store whole numbers; e.g., 3, 100, -39
2. Floating point data types : hold real numbers; e.g., 1.011, 3.000, 100.64
3. The char data type : hold one character; e.g., a, b, z, Z, 4, #, \n, \t
4. C++ string class : hold a string of characters. (provided by string class; so should include string class)
5. Boolean data Type : are set to either true or false.

Considerations for selecting a numeric data type:

- _____ that may be stored in.
- _____ the variable uses
- _____ numbers
- Number of _____ of precision

Data to be stored	features
Distance to a star	Large, positive
Microscopic dimension	Small, high precision
Number of soda bottles	Small, whole numbers only, positives only
Temperature	Small, both negative and positive
Vehicle speed	Small, only positive

2.6 Integer data types

Table 2-6 (pg 43)

data type	Typical size	Typical Range	Abbreviated name
short int			short
unsigned short int			unsigned short
int			int
unsigned int			unsigned
long int			long
unsigned long int			unsigned long
long long int			long long
unsigned long long int			unsigned long long

data type	Typical size	Typical Range
float		
double		
long double		

Types of variables (chart goes here)

Difference between string literals and Character literals

- Strings can be any length. e.g., string name = “Lasanthi Gamage”;
- To represents the end of the string, an additional character (_____) is appended. So, it needs an extra byte.
- A character literal takes only one byte. e.g., char initial = ‘L’;

Variable, name takes 16 bytes, whereas initial takes only one byte.

Assigning floating point literals to integer variables

```
int number = 7.5;

cout << “number = ” << number << endl;
```

output:
number = 7

```
float temperature = 100.1;
int testTemp = temperature;


cout << “temperature = ” << temperature << endl;
cout << “testTemp = ” << testTemp << endl;
```

output:

.....

.....

```
int number1 = 7.5;
float number2 = number1;

cout << “number1 = ” << number1 << endl;
cout << “number2 = ” << number2 << endl;
```

output:

.....

.....

Different possible floating-point literals

- 149590000000.00
- 1.4959E11
- 1.4959e11
- 1.4959E+11
- 1.4959e+11

2.11 determining the size of a data type

The special operator sizeof() will report the number of bytes of memory used by a data type.

```
cout << sizeof(int) << endl;
```
.....

```
cout << sizeof(long) << endl;
```
.....

```
cout << sizeof(short) << endl;
```
.....

```
cout << sizeof(float) << endl;
```
.....

```
cout << sizeof(double) << endl;
```
.....

```
cout << sizeof(char) << endl;
```
.....

```
int number1;
cout << sizeof(number1) << endl;
```
.....

```
long number2;
cout << sizeof(number2) << endl;
```
.....

```
double number3;
cout << sizeof(number3) << endl;
```
.....

2.12 Variable assignment and initialization

Definition: e.g., ...

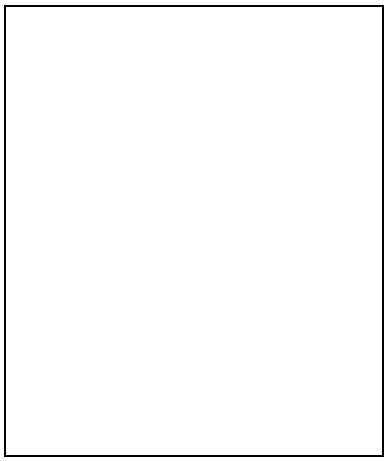
Assignment: e.g., ...

Initialization: e.g., ... (The value is assigned as part of the definition.)

- Assignment is an .....
- The data that the operator works with are ..... .
- Assignment operator has ..... operands.
- Operator on the ..... must be a variable.

Defining and initializing multiple variables.

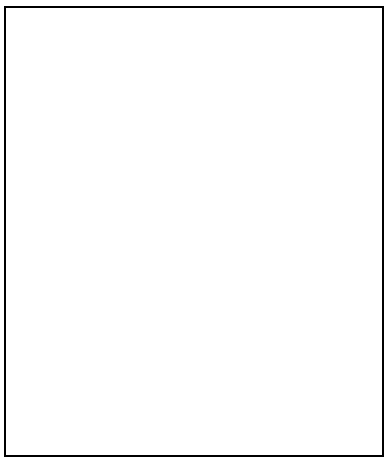
definition



equivalent to

```
int flightNum;
int travelTime;
int departure;
int distance;
```

definition



equivalent to

```
int flightNum = 89;
int travelTime;
int departure = 10;
int distance;
```

Type of the variable	definition	assignment	initialization
integer			
floating			

char			•
bool			•
string			•

Difference between string literals and Character literals

2.13 Scope

- A variable's scope is the part of the program that has access to the variable.
- Variable cannot be used in any part of the program ..... the definition.

2.14 Arithmetic Operators

Types of operators (based on the number of operands required):

.....	.....	.....
1. Unary	2. Binary	3. Ternary
e.g.,	e.g.,	e.g.,
• negation operator • sizeof() operator • increment operator • decrement operator	• addition • subtraction • multiplication • Division • Modulus • assignment	• const int x = (a<b) ? b : a;

Exercise

If `productCost` and `productPrice` are numeric variables, and `productName` is a string variable, and they are the only variables. which of the following statements are valid assignments? If a statement is not valid, explain why not.

		Valid/ Invalid	If invalid, why?
1	`productCost = 100;`	V/ IV	
2	`productPrice = productCost;`	V/ IV	
3	`productPrice = productName;`	V/ IV	
4	`productPrice = "24.95";`	V/ IV	
5	`15.67 = productCost;`	V/ IV	
6	`productCost = $1,35.52;`	V/ IV	
7	`productCost + 20 = productPrice;`	V/ IV	
8	`productName = 43;`	V/ IV	
9	`productName = "44" ;`	V/ IV	
10	`"99" = productName;`	V/ IV	
11	`productName = brush;`	V/ IV	
12	`productPrice = productPrice;`	V/ IV	
13	`productName = productCost;`	V/ IV	

Integer division

```
double number;
number = 5 / 2;

cout << “number = ” << number << endl;
```

output:

.....

```
double number;
number = 5.0 / 2;

cout << “number = ” << number << endl;
```

output:

.....

```
double number;
number = 5 / 2.0;

cout << “number = ” << number << endl;
```

output:

.....

```
double pizzaPerPerson;
int numParticipant = 20,
    numPizzas = 5;
pizzaPerPerson = numPizzas  / numParticipant;

cout << “pizza per Person =  ” << pizzaPerPerson << endl;
```

output:

.....

Comments

- Single Line comments
  - Two slashes
  - Comment only one line
- Multi line comments
  - /* and */
  - Comment a block of lines

Named Constants

Literals can be given names, e.g., **const double KGPERPOUND = 0.453592;**
When a literal is given a name, the definition must have all the 4 parts.

e.g.,

No named constants

```
weightKg = weightLb  *  0.453592 ;
```

vs

Named Constant

```
const double KGPERPOUND = 0.453592;

weightKg = weightLb  *  ……;
```

Statement	Legal?		Naming Convention	
	L/ IL	Why?	Ok/not ok	Why?
const double PI = 3.14159;				
double PI = 3.14159;				
const double PI; PI = 3.14159;				
const double PI; cin >> PI ;				
const double PI = 3.14159; cin >> PI ;				
const double PI = 3.14159; area = PI * r * r ;				

HW 3: (Submit through WorldClassroom)

Write a C++ program segment that takes one user input (radius of a circle) and find the circumference and the area of the circle and output the results. Your code should contain separate statements for variable definition, input, processing, and output.

1. .....
2. .....
3. .....
4. .....
5. .....
6. .....
7. .....
8. .....
9. .....
10. .....
11. .....
12. .....
13. .....
14. .....
15. .....
16. .....
17. .....
18. .....
19. .....
20. .....
21. .....
22. .....
23. .....
24. .....