Part of C++ Program

C++ programs has parts and components that serve specific purposes.

```
//      A simple C++ program

/*      This is Just An Example
         of a Simple C++ Program */

#include <iostream>
using namespace std;

int main ()
{
        cout << " Programming is Fun !!!  ";
        return 0;
}
```

The   output of this program is :

```
Programming is Fun  !!!
```

Parts of a C++ Program

Comment: //...

- ignored by compiler
- notes to human reader

- /*...*/
    - **mark the beginning and end of a comment with multiple lines**

Preprocessor Directive: **#include** …

- Read the program before it is compiled.
- It sets up your code.
- **Preprocessor** inserts contents of file here using namespace std;
- Every name in iostream is part of std namespace.
- (Note: There is **no** semicolon at the end of a preprocessor directive.)

iostream : Name of the file that is going to be included in your program is placed in this area. It is called file header.

Opening/Closing angle brackets < >

- Encloses the filename when used in a preprocessor directive

using namespace std;

- Declares that a program will be accessing entities whose names are part of the namespace called std
- e.g. every name created by the iostream file is part of the namespace std

`int`

- Short for integer – it indicates that the function main will send and integer value back to Operating System (OS) when it is completed execution.

`main()`

- Start of function main . Function is a group of one or more programming statement. named main
- The starting point of the program

{ } - Contains the body of the function

cout << " Programming is Fun !!! ";

- Short for " console output ".
- Statement that displays message on screen

return 0 ; : Sends value of 0 to OS (means success! Or normal program termination)

; Semicolon : Marks the end of a complete programming statement

The cout object

cout : Short for "Console Output" (represents the screen)

- A stream object: works on a sequence of data
- **<<** : the stream insertion operator
- Sends value on right-hand side (rhs) to stream on left hand side (lhs)
- **cout** << " This is an example. " **;**
- **endl** : short for **"end line".** It is a stream manipulator. It Advances output to start of next line

NOTE : C++ is case-sensitive Programming Language.

Special characters used within cout statements

- Newline: **\n** or **endl**
- Horizontal tab : **\t** causes the cursor to skip over to the next tab stop.
- Single quote: **\'** - causes a single quotation mark to be printed
- Double quote: **\"** – causes a double quotation mark to be printed.
- Double backslash: \\ - causes a single backslash mark to be printed
- These can occur in strings:
  - "hello**\n**there"
  - "she said **\"** boo **\"** very quietly"

- See textbook for more (; < > …..etc.)

It's a backslash (\), not a forward slash (/)

Examples with the ***cout*** statement

```
cout << "This is an example. ";

cout << "This is" << " an example. ";

cout << "This is";
cout << " an example";

cout << "The best selling book on Amazon";

cout << " is \"The Help\" ";

cout << "The best selling book on Amazon" << endl;

cout << " is \" The Help\" ";

cout << "The best selling book on Amazon \n is \" The Help \" ";

cout<<"Programming is  \' Fun\'  "<<endl<<endl;

cout<<"Programming\n"<<"          is\n"<<"         Fun "<<endl<<endl;

cout<<"Programming is  \\ Fun \\ "<<endl<<endl;

cout<<" Programming is \t\t\t Fun  "<<endl;
```

Literals

A literal represents a constant value from a given data type. It is used in a program statement.

- **Numbers** : 0, 34, 3.14159, -1.8e12, etc.
- **Characters** : 'A', 'z', '!', '5', etc.
- **Strings** (sequence of characters) : "Hello", "This is a string" "100 years", "100", "Y" etc.

NOTE: These are all different: 5, '5', "5"

Identifiers

- An identifier is a name for some program element (Like a variable)

- Rules:
    a) May not be a keyword (see p. 41 for complete list)
    b) The first character must be a letter or underscore
    c) Following characters must be letters, numbers or underscores **only**.

- **Identifiers are case-sensitive:**

    - myVariable is not the same as MyVariable

Examples : dayOfweek , _legal , May2012

What about This group : _employee_name , 3C, C#

Variables

- Variable : named location in main memory
- Variable definition in a program:
    - <**datatype**> <**identifier name**>; // pay attention to identifier rules.

examples:

- int someNumber;
- char firstLetter;

What about the following : int int , int Int , int _main , int include# , int namespaces.

Note : Variables must be defined before it can be used.

Variable Assignment

- An assignment statement uses the **= operator** to store a value in an already defined variable.
  - someNumber = 12;

- When this statement is executed, the computer stores the value 12 in memory, in the location named “someNumber”.
- The variable receiving the value must be on the **left side of the =** (the following does NOT work):

  - 12 = someNumber; // This is an **ERROR**

Variable Initialization

- To initialize a variable means to assign it a value when it is defined:

  - `int length = 12;`

- You can define and initialize multiple variables at once (and change them later) :

```
int length = 12, width = 5, area;
area = 35;
length = 10;
area =40;
```

Program with a variable

What is the output of this program ??

```
#include <iostream>
using namespace std;
int main()
{
        int number;
        number = 100;
        cout << "The value of the number is  " << number <<  endl;

        number = 50;
        cout << "The value of the number is  " << number <<  endl;

        cout << "The value of the number is  " << " number " <<  endl;

        return 0;
}
```

Data Types

- Variables are classified according to their data type.
- The data type determines the kind of information that may be stored in the variable.
- A data type is a set of values.
- Generally two main (types of) data types:
    a. Numeric (integers such as 3, 157 , -47 and floating points such 23.7 , 0.94)
    b. Character
- Primary Consideration for selecting a numeric data type are :-
    a. The largest and the smallest numbers that may be stored in the variable.
    b. How much memory the variable uses.
    c. Whether the variable stores signed or unsigned numbers
    d. The number of decimal places of precision the variable has.

Data Types

Data Type	Represents
`int, short, long`	whole numbers (integers)
`float, double`	real numbers (fractional , decimal)
`bool`	logical values : `true, false`
`char`	a single character
`string`	sequence of chars.

Integer Data Types

`int, short int , long int`

- Whole numbers 2 , 1000 , -900
- May be signed or unsigned
- Typical sizes and ranges (may vary depending on the system)
- Literals (are int by default)

Integer Data Types

Data Type	Size	Range
Short int	2 bytes	-32,768 to 32,767
unsigned short int	2 bytes	0 to 65,535
int	4 bytes	-2,147,483,648 to 2,147,483,647
unsigned int	4 bytes	0 to 4,294,967,295
Long int	4 bytes	-2,147,483,648 to 2,147,483,647
unsigned long int	4 bytes	0 to 4,294,967,295
Long long int	8 bytes	-9,223,372,036,854,775,808 to 9,223,372,036,854,775,808
unsigned long long int	8 bytes	0 to 18,446,744,073,709,551,615

Example of Variable Definitions:

```
short  dayOfWeek;
unsigned  long  distance;
int  xCoordinate  =  10;
long  deficit  =  1500;
```

Floating-Point Data Types

- Used to hold real numbers such as 2.5 , -7.8

- Typical sizes and ranges (may vary depending on the system):

Single Precision	**float**	4 bytes	+/- 3.4e +/- 38 (~7 digits)
Double Precision	**double**	8 bytes	+/- 1.7e +/- 308 (~15 digits)
Long Double Precision	**long double**	8 bytes*	+/- 1.7e +/- 308 (~15 digits)

***some compiler use 10 bytes for long double : the range is +/- 3.4E-4932 and**

+/- 1.1E4832

- Floating-point literals can be represented in

  – Fixed point (decimal) notation: `31.4159 0.0000625`
  – E (scientific) notation: `3.14159E1 6.25e-5`

Note : there are no unsigned floating point data types. On all machines, variables of the float , double, and long double data types can store positive or negative numbers.

Literals (default type is double) – can be expressed in a variety of ways :-

```
31.415E5     // equivalent to 3141500.0 - E or e will work - but printed as e.
-31.415e5    // equivalent to -3141500.0
3.1e-4       // equivalent to 0.00031
```

Floating-Point Data Types

```
float      distance, time;
double     mass;

distance = 1.495979E11;  // how far away the sun is (in meters)
mass = 1.989E30;   // how much the sun weighs (in kilograms)
time = 12.816;     // hours of daylight in San Marcos today, 8/31
```

Converting between floating-points and integers:

```
int i;    float f;
f = 8.9;
i = 8.9;          // stores 8 in i (  truncates, does not round  )
i = 8;
f = 8;            // stores 8.0 in f
f = 7.9;
i = f;            // stores 7 in i
```

The bool Data Type

- Defined as **bool**
- Literals: the values are true or false

```
bool   boolValue;
boolValue     =       true;
cout << boolValue << endl;
boolValue     =        false;
cout << boolValue << endl;
```

Output:

```
1
0
```

- bool is a numeric type:
  - true is 1 and false is 0

The char Data Type

- **char**
- Literals: All the keyboard and printable symbols such as 'A' '3' '!' '\n' 'n'.
- Numeric value of character from the ASCII character set is stored in memory:

```
char   letter;
letter  =        'A';      //    65  is stored in memory
cout << letter << endl;
letter = '!';
cout << letter << endl;
```

Output:

```
A
!
```

- char is really a numeric type also!
- Note: 65 is the ASCII code for 'A'

```
char letter;
letter = 65;
cout << letter << endl;
letter = 66;
cout << letter << endl;
```

Output:

```
A
B
```

The string Data Type

- A string is a sequence of characters.
- Requires the string header file: #include <string>
- Literals: “Hello again” “Over\nThere” “Y”
- A string is stored sequentially in memory, with the null character ('\0') at the end.
- The null character is not displayed.
- To define string variables in programs:

```
string   firstName  ,  lastName;
```

- To assign literals to variables :

```
firstName  = "George";
lastName = "Washington";
```

- To display via cout :

```
cout << firstName << " " << lastName;
```

Named Constants

- **Variable whose value cannot be changed during program execution**

Literals do not have "meaningful names"

```
cost = price + (price * .0825);
```

- what is the meaning of .0825?

Same literal may be used throughout a program, but may want to change it later.

- Maybe `.0825` occurs in dozens of places in the code.
- Search and replace problem.

Literals may be given names to be used in their place.

General Form:

- **const data_type VARIABLE = value;**

For Example

```
o const double SALES_TAX_RATE = .0825;
```

Then the equation will be

```
cost = price + (price * SALES_TAX_RATE);
```

- const makes the variable read-only
- Initialization required
- All-caps for the name of the constant is just a convention

Scopes of a Variable

A variable's scope is the part of the program in which a variable can be accessed.

Rule : A variable cannot be used before it is defined.

Example :-

```
#include <iostream>
using namespace std;
int main () {
value = 150; //error, use of value before it is
defined
int value;
cout << value;       }
```

sizeof

- sizeof function returns size of a data type in bytes in any system.
- The result is system-dependent.
- The argument may be a data type:

```
sizeof(int)                    // result is 4 on most systems
```

- The argument may be a variable:

```
double salary;
cout << sizeof(salary);        // result is 8 on most systems
```

What is the output of the following ??

```
cout << "The size of a short is " << sizeof(short)       << " bytes.\n";

cout << "The size of an integer is " << sizeof(int)       << " bytes.\n";
```

Declaring Variables with the **auto** Key Word

The auto word key tells the compiler to determine the variable's data type from the initialization value.

```
auto  amount = 100;
auto interestRate =  12.5;
auto  stockCode = 'X';
```

The above statements uses *auto* instead of a data type

Find any errors in the following C++ program:

```
#include<iostream>
using namespaces std;

int mian(){

  integer a;
  a,b,s int;
  d float;
  cin <<"The end of the program";
  retun 0;
}
```

Example of C++ Reserved words Reference List and Keyword Description

----

bool : declare a Boolean variable
break : break out of a loop
case : a block of code in a switch statement
catch : handles exceptions from throw
char : declare a character variable
class : declare a class
const : declare immutable data or functions that do not change data
const_cast : cast from const variables
continue : bypass iterations of a loop
default : default handler in a case statement
do : looping construct
double : declare a double precision floating-point variable
else : alternate case for an if statement
float : declare a floating-point variable
for : looping construct
if : execute code based off of the result of a test
int : declare a integer variable
long : declare a long integer variable
namespace : partition the global namespace by defining a scope
return : return from a function
short : declare a short integer variable
signed : modify variable type declarations
sizeof : return the size of a variable or type
static : create permanent storage for a variable
static_cast : perform a nonpolymorphic cast
struct : define a new structure
switch : execute code based off of different possible values for a variable
true : the Boolean value of true
unsigned : declare an unsigned integer variable
using : import complete or partial namespaces into the current scope
void : declare functions or data with no associated data type
while : looping construct