Learning to Work Efficiently and Accurately
Two key concepts in OOP are objects and classes

OOP was developed because limitation was discovered in earlier approaches to programming.

PROCEDURAL LANGUAGES

- Pascal, C, Basic, Fortran and similar languages are procedural languages, that is each statement in the language tells the computer to do something. A program in the procedural language is a list of instructions. The programmer creates a list of instructions and the computer carries them out.
- The procedural language, the emphasis is on doing thing for example read the keyboard, invert the vector, check the errors and so on.
- Data undervalued
- Data is difficult to handle and Data is easily to corrupted
- Memory limited

The Object-Oriented Approach

- The fundamental ideal behind object-oriented languages is to combined into a single unit both data and the functions that operate on that data. Such a unit is called an OBJECT.
An object functions, called member functions in C++, typically provide only way to access its data.

- Can not access data directly. The data is hidden
- Data and its functions are said to be *encapsulated* into a single entity.
- Data *encapsulation* and data hiding are key terms in the description of object-oriented languages.

In C++, program typical consists of number of objects, which communicate with each other by calling one another’s member functions.

- Data items are referred to as *instance* variables.
- Calling an object member’s function is referred to as sending a message to the object.

In OOP, objects are members of the *class*. A class is a collection of similar objects.

- The idea of classes leads to the idea of *inheritance*
- OOP classes can be divided into subclasses. The original class is called the *base class* and the subclasses are called *derived classes*. 
C++ Program - The Development Cycle

Start

Edit Source Code

Compile

Compile Error? YES NO

Link

Link Error? YES NO

Run Program

Run Time Error? YES NO

DONE
Reusability

Every C++ Program has a main( ) function.

A function is a block of code that performs one or more actions. Usually functions are invoked or called by other functions.

main( ) function is special. When your program starts, main( ) is called automatically.

All functions begin with open brace ( { ) and end with closing brace ( } ).

Comments

Comments are simply text that is ignored by the compiler, but that may inform the reader of what you are doing at any particular point in your program.

Types of Comments:

The double slash ( // ) for example   // this is my first c++ program

A start slash ( /*   */ ) for example   /* this is my first c++ program   */
• **Functions**
  - is one of the fundamental of C++ program
  - can be part of a class in which case called member function.
  - Can be independent from the class.
  - A C++ program must have one function that is `main ()`.
  - The parentheses ( ) following the word `main` are the distinguished feature of the function.
  - The first statement is executed will be at the beginning of the `main ()` function every times running a program in C++.

• **Program statements**
  - is the fundamental unit of C++ programming.
  - Each statement must be ended with semicolon ( ; ).

• **White space**
  - is ignored in C++ programming.
• String constant
  ▪ surrounded by quotation marks. For example “ Hello “
  ▪ can not be given new value while program running.

• Preprocessor directives
  ▪ start with a sign # and end without semicolon.
  ▪ Example: \#include<iostream.h>.
  ▪ Instruction to the compiler itself.

• The \#include directive
  ▪ tells the compiler to insert another file into your source file

• Header files
  ▪ iostream.h is one of the header files

• Comment syntax
  ▪ Comment start with a double slash symbol [ // ] in C++
  ▪ A start slash ( /*   */ ) for example /* this is my first c++ program */
• **Declarations and definitions**
  
  ▪ *A declaration* introduces a variable’s name into program and specifies its type.
  
  ▪ If a declaration also sets aside memory for the variable, it is called *definition*.
  
  ▪ Must declare a variable’s type before you can use the variable.

• **Variable names**
  
  ▪ all upper case or lower case letters
  
  ▪ digit from 1 to 9 and the underscore [ _ ]
  
  ▪ the first character must be a letter
  
  ▪ as long as desired ( maximum 32 characters )
  
  ▪ can not use Keyword , keyword is a identifier for a language feature

• **Assignment statements** [ = ]
  
  ▪ example *Var1 = 20*; the equal sign in above example causes value on the right to be assigned to the variable on the left.
- **Integer constant**
  - the number 20 is an integer constant.
  - The is no decimal point in the integer.

- **Output variations**
  - example: `cout << " the value of variable: " << var1 ;`
  - the above example will display the value of `var1`.

- **Character variables**
  - keyword: `char`
  - occupied only 1 byte of memory

- **Character constant**
  - character constant uses single quotation marks a round a character for example ‘a’.
  - Will be stored as decimal in ASCII code.
• **Initialization**
  - variables can be initialized at the same time they are defined.

• **Escape sequences**
  - an escape characters changes the meaning of character that followed it. For example \*n is a letter n but /n is new-line.

• **Escape characters**
  - **CHARACTER** | **WHAT IT MEANS?**
    - \*n          | new line
    - \*t          | tab
    - \*b          | backspace
    - \*"          | double quote
    - \*’           | single quote
    - \*?            | Question mark
    - \*\           | backslash
    - \*xdd         | hexadecimal representation
    - \*f            | form-feed
    - \*r            | return
• **Variables**
  - variable is a location in your computer’s memory in which you can store a value and from which later retrieve that value.

• **Fundamental variable types**
  - | Type                | Size  | Value                                      |
    |---------------------|-------|--------------------------------------------|
    | unsigned short int  | 2 bytes | 0 to 65535                                 |
    | short int           | 2 bytes | -32,768 to 32,767                          |
    | unsigned long int   | 4 bytes | 0 to 4,294,967,295                        |
    | long int            | 4 bytes | -2,147,483,648 to 2,147,483,647           |
    | int (16 bit)        | 2 bytes | -32,768 to 32,767                         |
    | int (32 bit)        | 4 bytes | -2,147,483,648 to 2,147,483,647           |
    | char                | 1 bytes | 256 character values                       |
    | float               | 4 bytes | 1.2e-38 to 3.4e+38                        |
    | double              | 8 bytes | 2.2e-308 to 1.8e+308                      |
• **Case sensitivity**
  - C++ is case sensitive. Uppercase and lowercase letters are considered to be different.

• **Key words**
  - some words are reserved by C++ program and may not be used as variable name
  - Keywords include if, while, for and main().....

• **typedef keyword**
  - `typedef unsigned short int USHORT;`
  - Creates a new name USHORT that can be used anywhere to substituted for `unsigned short int`.

• **The const qualifier**
  - specifies the value of a variable will not change throughout the program. For example `const unsigned short int val=15;`

• **Define constant with #define**
  - `#define PI 3.14159;`
• **Manipulator**
  - manipulators are operators used with the insertion operator [ << ] to modify or manipulate the way data is displayed.
  - The `endl` Manipulator insert a new-line same as “\n”
  - The `setw` Manipulator
    - causes the number or string that follows it in the stream to be printed within a field n characters wide, where n is an argument to `setw(n)`.
    - The value is right justified within the field.

• **Enumerated constants**
  - enables to create a new types and then to define variable of those types whose values are restricted to a set of possible values.
  - Example: `enum color { red, blue, green, white, black };`
    - This statements perform two tasks:
      - It makes `color` the name of enumeration, that is new type
      - It makes `red` a symbolic constant with the value 0, blue with 1, green with 2 and so forth.
// A demonstration of enumerated constants
#include<iostream.h>
int main ( )
{
enum Days { Sunday, Monday, Tuesday, Wednesday, Thursday, Friday, Saturday } ;
Days dayoff ; // dayoff type Days
int x;
cout<"What day would you like off ( 0-6 )?"; 
cin>> x;
dayoff = Days(x);
if ( dayoff == Sunday || dayoff == Saturday )
   cout<"You are already off on weekends!"<endl;
else
   cout<"Okay, I will put you in vacation day!"<endl;
return 0;
} // end main ( )
• **Statements**
  - In C++ statement controls the sequence of execution, evaluates an expression, or does nothing (the null statement).
  - All C++ statements ended with a semicolon (;)

• **Whitespace**
  - Whitespace (tabs, spaces, newlines) is generally ignored in statements.

• **Blocks and compound statements**
  - Any place you can put a single statement, or compound statement, also called a block.
  - A block begins with an opening brace ( { } ) and ends with a closing brace ( } ).
  - Every statement in the block must end with a semicolon ( ; ).

• **Expressions**
  - Anything that evaluates to a value is an expression in C++. An expression is said to return a value.
All expressions are statements.

- **Operators**
  - an operator is a symbol that causes the compiler to take an action.
  - Operators are used to manipulate data, like perform calculations.

- **Assignment operator**
  - the assignment operator ( = ) causes the operand on the left side of the assignment operator to have its value change to the value on the right side of the assignment operator. Example: \( x = a + b; \)

- **Mathematical operator**
  - addition: +
  - subtraction: -
  - multiplication: *
  - division: /
  - modulus: %
  - Increment and decrement
  - increment operator ( ++ ) increases the value of the variable by 1.
  - Example: \( c++ \) is \( c = c + 1 \).
- **Decrement operator (--)**
  - decreases the value of the variable by 1.
  - Example: c-- means c = c - 1.
  - Also c+ = 1 means c = c + 1 also c += b means c = c + b.

- **Prefix and Postfix**
  - Prefix: int a = ++x means increment x then use it.
  - Postfix: int a = x ++ means use it then increment.
  - Example if x=5 then a=6 in prefix and a=5 in postfix.

- **Precedence**
  - is the order in which a program performs the operations in formula.
  - If one operator has precedence over another operator; it is evaluated first.
### Operator Precedence

<table>
<thead>
<tr>
<th>Rank</th>
<th>Name</th>
<th>Operator</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>scope resolution</td>
<td>::</td>
</tr>
<tr>
<td>2</td>
<td>indirect membership</td>
<td>-&gt;</td>
</tr>
<tr>
<td></td>
<td>Direct membership</td>
<td>.</td>
</tr>
<tr>
<td></td>
<td>function call,</td>
<td>( )</td>
</tr>
<tr>
<td></td>
<td>postfix</td>
<td>++</td>
</tr>
<tr>
<td></td>
<td>and prefix</td>
<td>--</td>
</tr>
<tr>
<td>3</td>
<td>multiply/divide/modulo</td>
<td>*</td>
</tr>
<tr>
<td></td>
<td></td>
<td>/</td>
</tr>
<tr>
<td></td>
<td></td>
<td>%</td>
</tr>
<tr>
<td>4</td>
<td>add/ subtraction</td>
<td>+</td>
</tr>
<tr>
<td></td>
<td></td>
<td>-</td>
</tr>
<tr>
<td>5</td>
<td>logical AND/OR/NOT</td>
<td>&amp;&amp;</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>!</td>
</tr>
</tbody>
</table>
### Relational operators

<table>
<thead>
<tr>
<th>NAME</th>
<th>OPERATOR</th>
<th>EXAMPLE</th>
</tr>
</thead>
<tbody>
<tr>
<td>equals</td>
<td>==</td>
<td>if ( x==10)</td>
</tr>
<tr>
<td>not equals</td>
<td>!=</td>
<td>if ( x!=10)</td>
</tr>
<tr>
<td>greater than</td>
<td>&gt;</td>
<td>if ( x&gt;10)</td>
</tr>
<tr>
<td>greater than</td>
<td>&gt;=</td>
<td>if ( x&gt;=10)</td>
</tr>
<tr>
<td>less than</td>
<td>&lt;</td>
<td>if ( x&lt;10)</td>
</tr>
<tr>
<td>or equals</td>
<td>&lt;=</td>
<td>if ( x&lt;=10)</td>
</tr>
<tr>
<td>indirection operator</td>
<td>*</td>
<td>int x =*y</td>
</tr>
<tr>
<td>address of operator</td>
<td>&amp;</td>
<td>int* x = &amp;y</td>
</tr>
</tbody>
</table>
• Logical operators

<table>
<thead>
<tr>
<th>operators</th>
<th>symbol</th>
<th>example</th>
</tr>
</thead>
<tbody>
<tr>
<td>AND</td>
<td>&amp;&amp;</td>
<td>express.1 &amp;&amp; express.2</td>
</tr>
<tr>
<td>OR</td>
<td></td>
<td></td>
</tr>
<tr>
<td>NOT</td>
<td>!</td>
<td>! Expression</td>
</tr>
</tbody>
</table>

Example:

if ( ( x= =5) && (y= =5))

if ( ! ( x= =5)) is true only if x is not equal to 5

or use the same as  if ( x != 5).

• Library functions

- many activities in C++ carried by library functions.
- These functions perform file access, mathematical computations, graphics, memory management and data conversion...
- for example to calculate the square of a number then first 
  #include(math.h) then using function sqrt(number).
• **The if statement**
  - The if statement is used to test for a condition and then execute section of code based on whether that condition is true or false.

• **The if statement (first form)**
  
  ```
  if (cond_expression)
  {
    dosomethings;
  }
  ```

• **The if statement (second form)**
  
  ```
  if (cond_expression)  
  {
    dosomethings;
  }
  else
  {
    dodifferentthing;
  }
  ```

  ```
  if (direction == EAST)  
  {
    lost = true;
  }
  else
  {
    lost = False;
  }
  ```
• The if statement (third form)

```c++
if ( cond_expression1 )
    {   true statement1 ;   }
else if ( conf_expression2)
    { true statement2;   }
else if ( conf_expression3 )
    { true statement3;   }
else
    {   false statement;   }
```

• The while loop

- a while loop causes your program to repeat a sequence of statements as long as the starting condition remains true.

**Syntax:**

```c++
while ( condition )
    statement ;
```

**example**

```c++
while ( x<10 )
    cout<<"x is "<< x++;
```
• The **continue** statement
  - Causes a **while** or **for** loop to begin again at the top of the loop.

• The **break** statement
  - Causes the immediate end of a **while** or **for** loop.
  - Execution jumps to the closing brace.
  - Example of break statement

```c++
while ( condition )
{
    if ( condition1 )
        break;
        // statements ;
}
// break.cpp
// demonstrates break and continue statement
#include<iostream.h>
```
• **The `do...while` loops**
  - it is possible that the body of the `while` loop will never executed.
  - The while statement checks its condition before executing any of its statements.

• **`do...while`**
  - the `do...while` loop executes the body of the loop before its condition is tested and ensures that the body always executes at least one time.
  - Syntax:
    ```c++
    int x=0;
    do
    statements;
    while ( condition )
    cout<<" x: "<<x++;
    ```
  - example:
    ```c++
    for ( int l=0; l<10; l++ )
    cout<<" hello..”;
    ```

• **for loops**
  ```c++
  for ( initialization; test; action )
  statements;
  ```
  ```c++
  for ( int l=0; l<10; l++ )
  cout<<" hello..”;
  ```
• The `switch` statements
  - Allow to branch to any of a number of different values.
  - **Syntax:**

    ```
    switch (expression )
    {
    case valueone: statement;
      break;
    case valuetwo: statement;
      break;
    case valuen: statement;
      break;
    default:  statement;      }
    ```