

**LO1. Apply basic language syntax and layout****1.1. Understanding Basic language syntax rules and best practices**

What are the Basic programming languages?

Some of the basic programming languages are:

- C, C++, JAVA, Visual Studio (Visual basic.Net, C-Sharp(C#), and Visual C++), and so on.

Syntax is a rule that specify how valid instructions (constructs) are written.

**Example: C++ Basic Syntax contains:**

- |                     |             |
|---------------------|-------------|
| ▪ Headers (Package) | ▪ Variable  |
| ▪ Return type       | ▪ Data type |
| ▪ Main function     |             |

- **Header:** The C++ language defines several headers, which contain information that is necessary to your program.
- **Function:** function is a portion of code within a larger program, which performs a specific task and is relatively independent of the remaining code.
- **Return type:** Return type is used to terminate main( ) function and causes it to return the value to its type.  
If a function is defined as having a return type of void, it should not return a value.  
If a function is defined as having a return type other than void, it should return a value.
- **Main function:** The main function ( **Main()** ) is a function where program execution begins.
- **Variable:** A variable is the name for a memory location where you store some data. A variable in C++ must be declared (the type of variable) and defined (values assigned to a variable) before it can be used in a program.

**1.2. Understanding Data-Types, Operators and Expressions**

**Data type:** A data type defines which kind of data will store in variables and also defines memory storage of data.

There are two kinds of data types User defined data types and Standard data types.

The C++ language allows you to create and use data types other than the fundamental data types. These types are called user-defined data types.

In C++ language, there are **four main data types**

- int
- float
- double
- char

**1. Character data type**

- A keyword **char** is used for character data type. This data type is used to represents letters or symbols.
- A character variable occupies 1 byte on memory.

**2. Integer data types**

Integers are those values which has no decimal part and they can be positive or negative. Like 12 or -12.

There are 3 data types for integers

- a. int
- b. short int
- c. long int

**int**

- int keyword is used for integers. It takes two bytes in memory.

There are two more types of int data type.

- i. **Signed int** or **short int (2nd type of integer data type)**

ii. **Unsigned int** or unsigned short int

**Signed int:** The range of storing value of signed int variable is -32768 to 32767. It can interact with both positive and negative value.

**Unsigned int:** This type of integers cannot handle negative values. Its range is 0 to 65535.

**Long int**

- As its name implies, it is used to represent larger integers.
- It takes 4 byte in memory and its range is -2147483648 to 2147483648.

**Float:** This type of data type is defined for fractional numbers.

There are two further type of data type for fractional numbers.

Below table is Displaying:

- Number of bytes or memory taken by numbers. Decimal places up to which they can represent value.

Data type	Bytes	Decimal places	Range of values
float	4	6	3.4E -38 to 3.4E+38
double	8	15	1.7E - 308 to 1.7E +308
long double	10	19	3.4E -4932 to 304E +4932

**Constant**

Constant is a data type whose value is never changes and remains the same through the programme execution.

Example: - Area of the circle  $A=\pi r^2$  ; const float  $\pi=3.14$

- Total number of days in a week=7 ;
- Total hours in a day=24
- Body temperature of human= $23^{\circ}\text{C}$

Here the value of above listed can be treated as a constant through the program execution.

The four basic data types in C++, their meaning, and their size are:

Type	Meaning	Size (bytes)	Size (bits)
char	a single byte, capable of holding one character	1 byte	8 bits
int	an integer	2 bytes	16 bits
float	single-precision floating point number	4 bytes	32 bits
double	double-precision floating point number	8 bytes	64 bits

**Byte** is the smallest addressable memory unit. **Bit**, which comes from **BI**nary digi**T**, is a memory unit that can store either a 0 or a 1. A byte has 8 bits.

**Operators in C++**

An operator is a symbol that tells the compiler to perform specific mathematical or logical manipulations.

C++ is rich in built-in operators and provides following type of operators:

- Arithmetic Operators
- Relational Operators
- Increment Operator
- Decrement Operator
- Scope resolution operator
- Logical Operators
- Assignment Operators

**Arithmetic Operators:**

The following arithmetic operators are supported by C++ language:

Assume variable A holds 10 and variable B holds 20 then:

Operator	Description	Example
+	Adds two operands	A + B will give 30
-	Subtracts second operand from the first	A - B will give -10
*	Multiply both operands	A * B will give 200
/	Divide numerator by de-numerator	B / A will give 2
%	Modulus Operator and remainder of after an integer division	B % A will give 0
++	Increment operator, increases integer value by one	A++ will give 11
--	Decrement operator, decreases integer value by one	A-- will give 9

### Relational Operators:

The following relational operators are supported by C++ language:

Assume variable A holds 10 and variable B holds 20 then:

Operator	Description	Example
==	Checks if the value of two operands is equal or not,	(A == B) is not true.
!=	Checks if the value of two operands is equal or not,	(A != B) is true.
>	Checks if the value of left operand is greater than the value of right operand,	(A > B) is not true.
<	Checks if the value of left operand is less than the value of right operand,	(A < B) is true.
>=	Checks if the value of left operand is greater than or equal to the value of right operand,	(A >= B) is not true.
<=	Checks if the value of left operand is less than or equal to the value of right operand,	(A <= B) is true.

### Logical/Boolean operator:

The following logical operators are supported by C++ language:

Assume variable A holds 1 and variable B holds 0 then:

Operator	Description	Example
&&	If both the operands are non-zero then condition becomes true.	(A && B) is false.
	If any of the two operands is non-zero then condition becomes true.	(A    B) is true.
!	Use to reverses the logical state of its operand. If a condition is true then Logical NOT operator will make false.	!(A && B) is true.

### Assignment Operators:

Assignment operator is used for assign/initialize a value to the variable during the program execution.

The following assignment operators are supported by C++ language:

Operator	Description	Example
=	Simple assignment operator, Assigns values from right side operands to left side operand	C = A + B will assign value of A + B into C
+=	Add AND assignment operator, It adds right operand to the left operand and assign the result to left operand	C += A is equivalent to C = C + A

<code>-=</code>	Subtract AND assignment operator, It subtracts right operand from the left operand and assign the result to left operand	<code>C -= A</code> is equivalent to <code>C = C - A</code>
<code>*=</code>	Multiply AND assignment operator, It multiplies right operand with the left operand and assign the result to left operand	<code>C *= A</code> is equivalent to <code>C = C * A</code>
<code>/=</code>	Divide AND assignment operator, It divides left operand with the right operand and assign the result to left operand	<code>C /= A</code> is equivalent to <code>C = C / A</code>
<code>%=</code>	Modulus AND assignment operator, It takes modulus using two operands and assign the result to left operand	<code>C %= A</code> is equivalent to <code>C = C % A</code>

**Increment Operator (++):** This operator is used for increment the value of an operand.

Example: assume that `A=20` and `B=12`, then `++A=21`, `++B=12` and `A++=20`, `B++=12`

**Decrement Operators (--):** This operator is used for decrement the value of an operand.

Example: assume that `B=14` and `C=10`, then `--B=13`, `--C=9` and `B--=14`, `C--=10`

### Scope resolution operator (::)

This operator is used to differentiate a local variable from the global variable. The variable having the same name can have different meaning at different block.

### Expression:

Expressions are formed by combining operators and operands together following the programming language.

### Compile & Execute C++ Program:

Lets look at how to save the file, compile and run the program. Please follow the steps given below:

- Open a text editor and write the code
- Save the file as : `hello.cpp`
- Compile it to check if it has error
- Debug the code.
- You will be able to see 'Hello World' printed on the window.

### Semicolons & Blocks in C++:

In C++, the semicolon is a statement terminator. Each individual statement must be ended with a semicolon.

For example, following are three different statements:

```

x = y;
y = y+1;
add(x, y);

```

A block is a set of logically connected statements that are surrounded by opening and closing braces. For example:

```

{
    cout << "Hello World"; // prints Hello World
    return 0;
}

```

### 1.3. Using the Appropriate Language Syntax for Sequence, Selection and Iteration Constructs

Control Structures in C++ is a Statement that used to control the flow of execution in a program.

There are three types of Control Structures:

1. Sequence structure
2. Selection structure
3. Loops/ Repetition/ Iteration

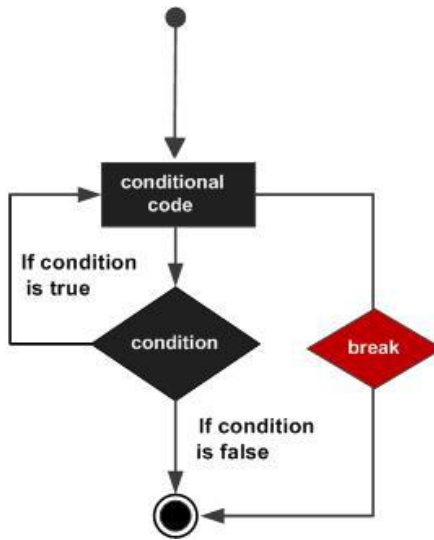
#### Sequence Structure in C++

The sequence structure is built into C++ statements that execute one after the other in the order in which they are written—that is, in sequence. **Break** and **continue** statements are example of sequence control structure.

The '**break**' statement causes an immediate exit from the inner most 'while', 'do...while', 'for loop' or from a 'switch- case' statement.

If you are using nested loops (ie. one loop inside another loop), the break statement will stop the execution of the innermost loop and start executing the next line of code after the block.

Flow Diagram:



Example:

```
#include <iostream.h>
int main ()
{
    int a = 10; // Local variable declaration:
    do{          // do loop execution
        cout << "value of a: " << a << endl;
        a = a + 1;
        if( a > 15)
        {
            break;
        }
    }while( a < 20 );
    return 0;
}
```

When the above code is compiled and executed, it produces following result:

```
value of a: 10
value of a: 11
value of a: 12
value of a: 13
value of a: 14
value of a: 15
```

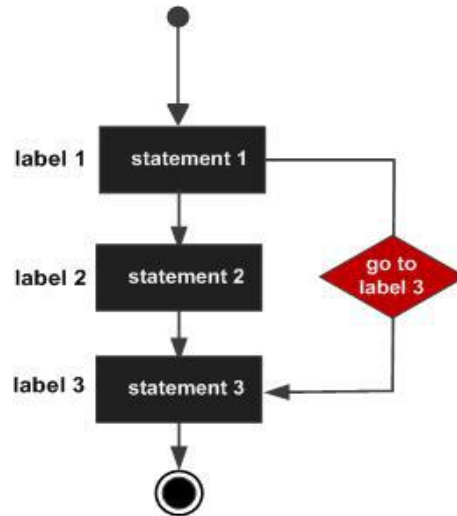
The **goto** statement provides an unconditional jump from the goto to a labeled statement in the same function.

Syntax of a goto statement in C++:

```
goto label;
..
.
label: statement;
```

Where **label** is an identifier that identifies a labeled statement. A labeled statement is any statement that is preceded by an identifier followed by a colon (:).

Flow Diagram:



Example:

```
#include <iostream.h>
int main ()
{
    int a = 10; // Local variable declaration:

    LOOP:do // do loop execution
    {
        if( a == 15)
        {
            a = a + 1; // skip the iteration.
            goto LOOP;
        }
        cout << "value of a: " << a << endl;
        a = a + 1;
    }while( a < 20 );

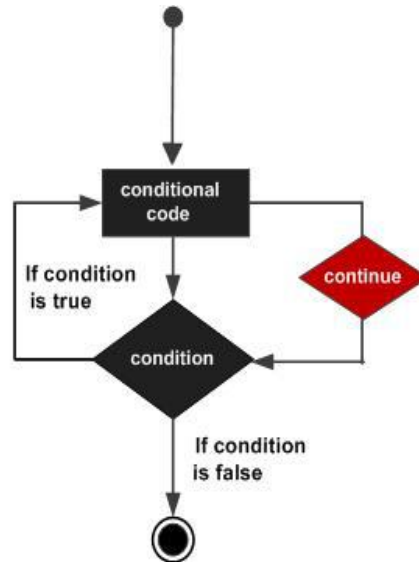
    return 0;
}
```

When the above code is compiled and executed, it produces following result:

```
value of a: 10
value of a: 11
value of a: 12
value of a: 13
value of a: 14
value of a: 16
value of a: 17
value of a: 18
value of a: 19
```

The **continue** statement is similar to 'break' statement but instead of terminating the loop, the continue statement returns the loop execution if the test condition is satisfied.

Flow Diagram:



Example:

```
#include <iostream.h>
int main ()
{
    int a = 10; // Local variable declaration:

    do{ // do loop execution
        if( a == 15)
        {
            a = a + 1; // skip the iteration.
            continue;
        }
        cout << "value of a: " << a << endl;
        a = a + 1;
    }while( a < 20 );

    return 0;
}
```

When the above code is compiled and executed, it produces following result:

```
value of a: 10
value of a: 11
value of a: 12
value of a: 13
value of a: 14
value of a: 16
value of a: 17
value of a: 18
value of a: 19
```

### Selection statements in C++

There are basically two types of control statements in c++ that allows the programmer to modify the regular sequential execution of statements. They are **selection** and **iteration** statements.

The selection statements allow to choose a set of statements for execution depending on a condition.

*If* statement and *switch* statement are the two selection statements.

#### If ... else statement:

**Syntax:** if (expression or condition)

```
{
    Statement 1;
    Statement 2;
```

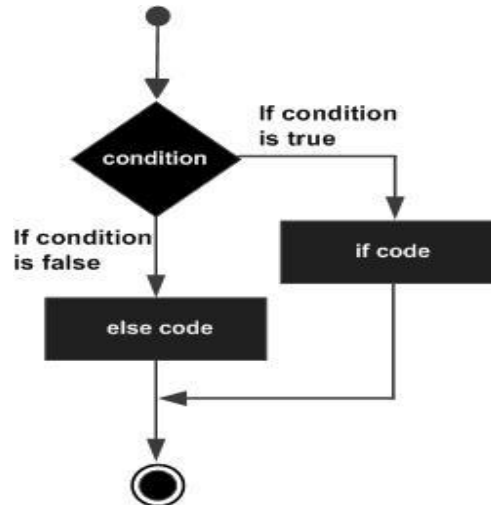
```

    }
Else
{
    Statement 3;
    Statement 4;
}

```

If the condition is true, statement1 and statement2 is executed; otherwise statement 3 and statement 4 is executed. The expression or condition is any expression built using relational operators which either yields true or false condition.

Flow Diagram:



If the condition evaluates to **true**, then the **if block** of code will be executed otherwise **else block** of code will be executed. Following program implements the *if* statement.

```

#include <iostream.h>
void main()
{
    int num;
    cout<<"Please enter a number"<<endl;
    cin>>num;
    if ((num%2) == 0)
        cout<<num <<" is a even number";
    else
        cout<<num <<" is a odd number";
}

```

The above program accepts a number from the user and divides it by 2 and if the remainder (remainder is obtained by modulus operator) is zero, it displays the number is even, otherwise it is odd.

you must use the relational operator '**==**' to compare whether remainder is equal to zero or not.

### Switch statement

One alternative to nested *if* statement is the **switch** statement which allows a variable to be tested for equality against a list of values. Each value is called a case, and the variable being switched on is checked for each case.

#### Syntax:

```

Switch (variablename)
{
    case value1: statement1; break;
    case value2: statement2; break;
    case value3: statement3; break;
}

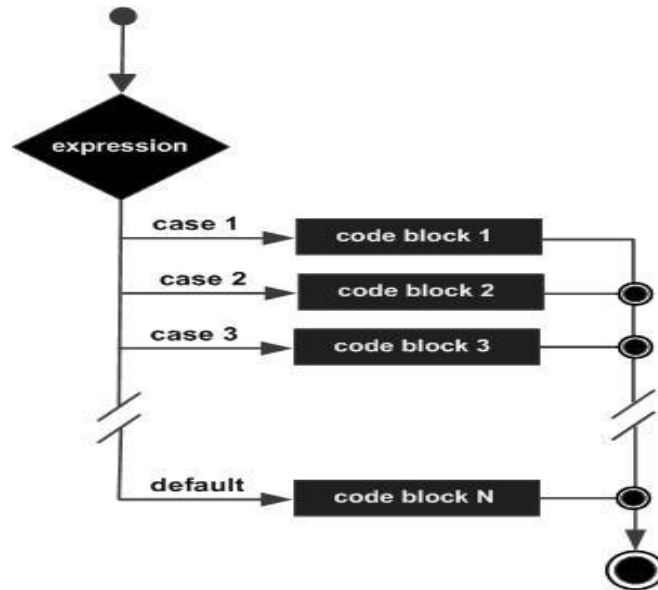
```



```
default: statement4;
```

```
}
```

Flow Diagram:



Example: #include<iostream.h>

```
void main() {
    int choice;
    cout<< "Enter a value for choice \n";
    cin >> choice;
    switch (choice)
    {
        case 1: cout << "First item selected!" << endl;
                break;
        case 2: cout << "Second item selected!" << endl;
                break;
        case 3: cout << "Third item selected!" << endl;
                break;
        default: cout << "Invalid selection!" << endl;
    }
}
```

### Iteration statements in C++

Iteration or loops statements are important statements in c++, which helps to accomplish repetitive execution of programming statements.

There are three loop statements in C++. They are - while loop, do while loop and for loop.

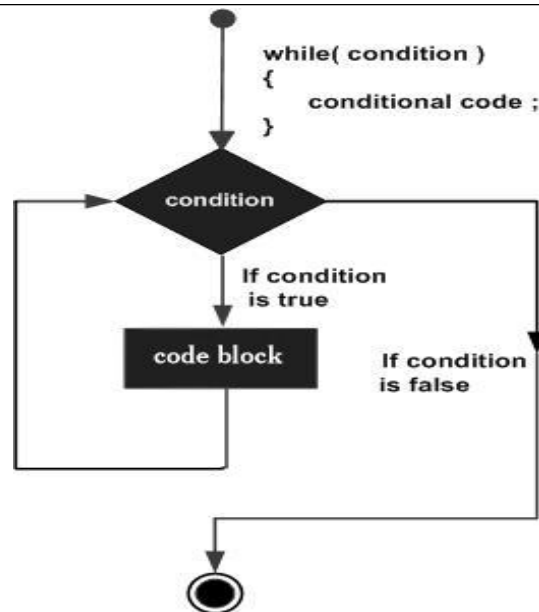
### While Loop

The while loop construct is a way of repeating loop body over and over again while a certain condition remains true. Once the condition becomes false, the control comes out of the loop. Loop body will execute only if condition is true.

Syntax: While (condition or expression)

```
{
    Statement1;
    Statement 2;
}
```

Flow Diagram:



The flow diagram indicates that a condition is first evaluated. If the condition is true, the loop body is executed and the condition is re-evaluated. Hence, the loop body is executed repeatedly as long as the condition remains true. As soon as the condition becomes false, it comes out of the loop and goes to display the output.

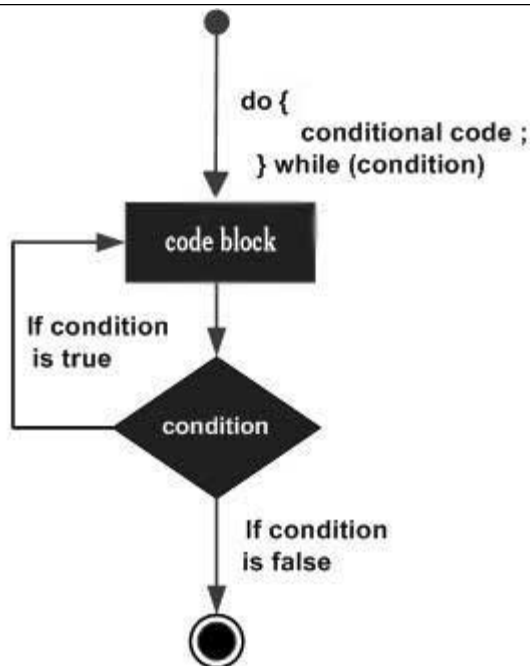
Example: `#include<iostream.h>`  
`void main()`  
`{`  
`int n, i=1, sum=0;`  
`cout<< "Enter a value for n \n";`  
`cin>>n;`  
`while(i<=n)`  
`{`  
`sum=sum+i;`  
`i++;`  
`}`  
`cout<< "sum of the series is "<<sum;`  
`}`

### Do ...While loop

The do... while statement is the same as while loop except that the condition is checked after the execution of statements in the do..while loop.

**Syntax:** `do{`  
`Statement;`  
`While (test condition);`  
`Statement;`  
`}`

Flow Diagram:



Its functionality is exactly the same as the while loop, except that condition in the do-while loop is evaluated after the execution of statement. Hence in do..while loop, Loop body execute once even if the condition is false.

Example: #include<iostream.h>

```

void main()
{
    int n, i=1, sum=0;
    cout<< "Enter a value for n \n";
    cin>>n;
    do{
        sum=sum+i;
        i++;
    } while(i<=n);
    cout<< "sum of the series is "<<sum;
}
  
```

### For loop:

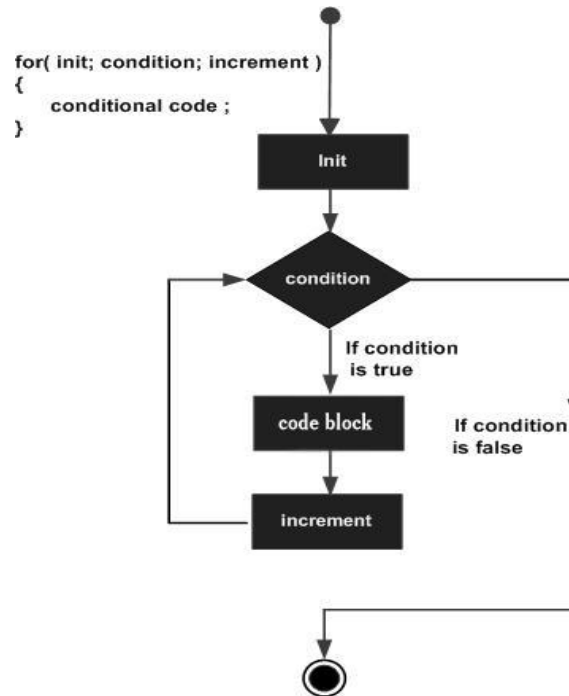
The for loop statements (loops or iteration statement) in C++ allow a program to execute a single statement multiple times (repeatedly) , given the initial value, the condition, and increment/decrement value.

#### syntax:

```

for ( initial value ; test condition ; increment/decrement)
{
    Statement;
}
  
```

## Flow Diagram:



A **for** loop is a repetition control structure that allows you to efficiently write a loop that needs to execute a specific number of times.

Example: #include<iostream.h>

```
void main()
{
    int i, n, sum=0;
    cout<< "Enter the value for n";
    cin>>n;
    for(i=1;i<=n; i++)
    {
        sum=sum+i;
    }
    cout<< "the sum is" <<sum;
}
```

### 1.4. Using modular programming approach

Many programs are too long or complex to write as a single unit. Programming becomes much simpler when the code is divided into small functional units (modules).

**Modular programming** is a programming style that breaks down program functions into modules, each of which accomplishes one function and contains all the source code and variables needed to accomplish that function.

Modular programs are usually easier to code, compile, debug, and change than large and complex programs.

The benefits of modular programming are:

- **Efficient Program Development**

Programs can be developed more quickly with the modular approach since small subprograms are easier to understand, design, and test than large and complex programs.

- **Multiple Use of Subprograms**

Code written for one program is often useful in others.

- **Ease of Debugging and Modifying**

Modular programs is generally easier to compile and debug than monolithic (large and complex) programs.

### 1.5. Using arrays and arrays of objects

An **Array** is a collection of similar data items which shares a common name within the consecutive memory. An Array can be any data type, but it should be the collection of similar items.

Each item in an array is termed as '**element**', but each element in an array can be accessed individually. The number of element in an array must be declared clearly in the definition.

The size or number of elements in the array can be varied according to the user needs.

#### Syntax for array declaration:

DataType ArrayName [number of element in the array]

Example: `int a[5];` - 'int' is the data type.

- 'a' is the array name.

- 5 is number of elements or size.

`int a[5]` means `a[0]`, `a[1]`, `a[2]`, `a[3]`, `a[4]` or

int	int	int	int	int
a[0]	a[1]	a[2]	a[3]	a[4]

- 0, 1, 2, 3, 4 are called **subscript** which is used to define an array element position and is called **Dimension**.
- Array index in C++ starts from 0 to n-1 if the size of array is n.

An individual element of an array is identified by its own unique **index** (or **subscript**).

**NOTE:** The elements field within brackets [], which represents the number of elements the array is going to hold, must be a **constant** value.

Arrays of variables of type "class" are known as "**Array of objects**". The "identifier" used to refer the array of objects is a user defined data type.

#### 1.6. Methods and Namespace

A **method** (member function) is a function that is a member of a class. **Function** is a portion of code within a larger program which performs a specific task and it is relatively independent of the remaining code.

i.e.:- A function is a group of statements that can be executed when it is called from some point of the program.

**Syntax:** Type functionName (parameter1, parameter2, ...)

```
{
    statements
}
```

- Type is the data type specified for the data returned by the function.
- FunctionName is the identifier by which it will be possible to call the function.
- parameters (as many as needed): Each parameter consists of a data type specified for an identifier (for example: `int x`) and which acts within the function as a regular local variable. They allow to pass arguments to the function when it is called. The different parameters are separated by commas.
- Statements are the function's body. It is a block of statements surrounded by braces { }.

Example: `#include <iostream.h>`

```
int addition (int a, int b)
{
    int sum;
    sum=a+b;
    return (sum);
}

int main ()
{
    int z;
    z = addition (5,3);
    cout << "The result is " << z;
```

```
        return 0;
    } //The result is 8
```

## Namespace

Namespaces are used in the visual C++ programming language to create a separate region for a group of variables, functions and classes etc. Namespaces are needed because there can be many functions, variables for classes in one program and they can conflict with the existing names of variables, functions and classes.

Therefore, you may use namespace to avoid the conflicts.

A namespace definition begins with the keyword **namespace** followed by the namespace name as shown bellow.

```
namespace namespace_name
{
    // code declarations
}
```

Let us see how namespace scope the entities including variable and functions:

```
#include <iostream>
using namespace std;
// first name space
namespace first_space{
void func(){
    cout << "Inside first_space" << endl;
}

}

// second name space
namespace second_space{
void func(){
    cout << "Inside second_space" << endl;
}

}

int main ()
{
    first_space::func(); // Calls function from first name space.
    second_space::func(); // Calls function from second name space.
    return 0;
}
```

If we compile and run above code, the code will produce following result:

```
Inside first_space
Inside second_space
```

A **namespace** declaration identifies and assigns a unique name to a user-declared namespace.

Such namespaces are used to solve the problem of name collision in large programs and libraries. Programmers can use namespaces to develop new software components and libraries without causing naming conflicts with existing components.

## **LO2. Apply basic OOP principles in the target language**

### **2.1 Introduction to class**

A class in C++ is an encapsulation of data members and functions that manipulate the data.

In terms of variables, a class would be the type, and an object would be the variable.

#### Classes and Objects

A class is a type, and an object of this class is just a variable.

Before we can use an object, it must be created.

Classes are generally declared using the keyword class.

```
Syntax: class class_name {
        access_specifier_1:
            member1;
        access_specifier_2:
            member2;
        ...
    } object_names;
```

Where class\_name is a valid identifier for the class, object\_names is an optional list of names for objects of this class. The body of the declaration can contain members, that can be either data or function declarations, and optionally access specifiers.

These specifiers modify the access:

- Public: Access to all code in the program.  
public members are accessible from anywhere where the object is visible.
- Private: Access to only members of the same class
- Protected: Access to members of same class and its derived classes

Example: #include <iostream.h>

```
class Rectangle {
    int x, y;
public:
    void set_values (int, int);
    int area () {
        return (x*y);
    }
};

void Rectangle::set_values (int a, int b)
{
    x = a;
    y = b;
}

int main () {
    Rectangle rect;
    rect.set_values (3,4);
    cout << "area: " << rect.area();
    return 0;
}
```

## 2.2. Class inheritance

Modern object-oriented (OO) languages provide 3 capabilities:

- Encapsulation
  - Inheritance
  - Polymorphism
- } which can improve the design, structure and reusability of the code.

**Encapsulation** is the method of combining the data and functions inside a class. This hides the data from being accessed from outside a class directly, only through the functions inside the class is able to access the information.

This is also known as "Data Abstraction", as it gives a clear separation between properties of data type and the associated implementation details. There are two types, they are "function abstraction" and "data abstraction". Functions that can be used without knowing how its implemented is function abstraction. Data abstraction is using data without knowing how the data is stored.

Features and Advantages of the concept of Encapsulation:

- Makes Maintenance of Application Easier:
- Improves the Understandability of the Application
- Enhanced Security

### Example:

```
#include <iostream.h>
class Add
{
    private:
    int x,y,r;
    public:
    int Addition(int x, int y)
    {
        r= x+y;
        return r;
    }
    void show( )
    {
        cout << "The sum is::" << r << "\n";}
    }s;
    void main()
    {
        Add s;
        s.Addition(10, 4);
        s.show();
    }
```

**Result:** The sum is:: 14

**Inheritance** is the process of creating new classes from the existing class or classes with inheriting some properties of the base class. Using Inheritance, some qualities of the base classes are added to the newly derived class.

The advantage of using "Inheritance" is due to the reusability (inheriting) of a class in multiple derived classes.

The ":" operator is used for inheriting a class.

The old class is referred to as the base class and the new classes, which are inherited from the base class, are called **derived classes**.

Example: #include <iostream.h>

```
class Value
{
    protected:
    int val;
    public:
    void set_values (int a)
    {
        val=a;
    }
}
```



```

    };
class Square: public Value
{
public:
    int square()
    {
        return (val*val);
    }
};

int main ()
{
    Square sq;
    sq.set_values (5);
    cout << "The square of "<<5<<" is::" << sq.square() << endl;
    return 0;
}

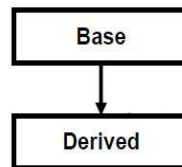
```

### Forms/types of Inheritance:

- Single Inheritance
- Multiple Inheritance
- Multilevel Inheritance
- Hierarchical Inheritance
- Hybrid Inheritance

1. Single inheritance: - If a class is derived from a single base class, it is called as single inheritance.

In Single Inheritance, there is only one Super Class and Only one Sub Class Means they have one to one Communication between them.



### Example: #include <iostream.h>

```

class Value
{
protected:
    int val;
public:
    void set_values (int a)
    {
        val=a;
    }
};

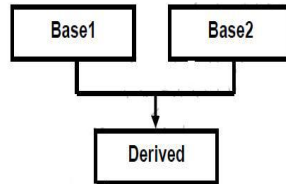
class Cube: public Value
{
public:
    int cube()
    {
        return (val*val*val);
    }
};

int main ()
{
    Cube cub;
    cub.set_values (5);
    cout << "The Cube of 5 is::" << cub.cube() << endl;
    return 0;
}

```

}

2. Multiple Inheritances: - If a class is derived from more than one base class, it is known as multiple inheritances.



Example: #include<iostream.h>

```
class student
{
    protected:
    int rno,m1,m2;
    public:
    void get()
    {
        cout<<"Enter the Roll no :";
        cin>>rno;
        cout<<"Enter the two marks:";
        cin>>m1>>m2;
    }
};

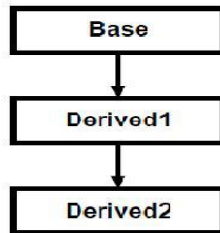
class sports
{
    protected:
    int sm;          // sm = Sports mark
    public:
    void getsm()
    {
        cout<<"\nEnter the sports mark :";
        cin>>sm;
    }
};

class statement: public student,public sports
{
    int tot,avg;
    public:
    void display()
    {
        tot=(m1+m2+sm);
        avg=tot/3;
        cout<<"\n\n\tRoll No : "<<rno<<"\n\tTotal : "<<tot;
        cout<<"\n\tAverage   : "<<avg;
    }
};

void main()
{
    statement obj;
    obj.get();
    obj.getsm();
    obj.display();
}
```

3. Multilevel Inheritance:

When a derived class is created from another derived class, then that inheritance is called as multi level inheritance.



Example: #include <iostream.h>

```

class A
{
protected:
int rollno;
public:
void get_num(int a)
{
    rollno = a;
}
void put_num()
{
    cout << "Roll Number Is:\n" << rollno << "\n";
}
};

class marks : public B
{
    protected:
    int sub1;
    int sub2;
    public:
    void get_marks(int x,int y)
    {
        sub1 = x;
        sub2 = y;
    }
    void put_marks(void)
    {
        cout << "Subject 1:" << sub1 << "\n";
        cout << "Subject 2:" << sub2 << "\n";
    }
};

class C : public marks
{
    protected:
    float tot;
    public:
    void disp(void)
    {
        tot = sub1+sub2;
        put_num();
        put_marks();
        cout << "Total:" << tot;
    }
};

int main()
{
    C std1;
    std1.get_num(5);
    std1.get_marks(10,20);
    std1.disp();
}
  
```

```

return 0;
}

```

### **Result:**

Roll Number Is:5

Subject 1: 10

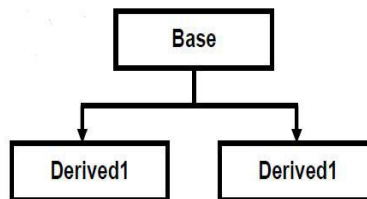
Subject 2: 20

Total: 30

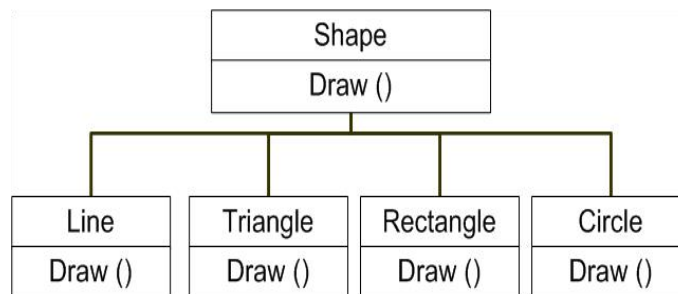
### 4. Hierarchical Inheritance:

If a number of classes are derived from a single base class, it is called as hierarchical inheritance.

This means a Base Class will have Many Sub Classes or a Base Class will be inherited by many Sub Classes.



Example:



**Example:** #include<iostream.h>

Class A

```

{
    int a,b;
public :
    void getdata()
    {
        cout<<"\n Enter the value of a and b";
        cin>>a>>b;
    }
    void putdata()
    {
        cout<<"\n The value of a is :"<<a "and b is "<<b;
    }
};

```

class B : public A

```

{
    int c,d;
public :
    void intdata()
    {
        cout<<"\n Enter the value of c and d ";
        cin>>c>>d;
    }
    void outdata()
    {
        cout<<"\n The value of c"<<c"and d is"<<d;
    }
}

```

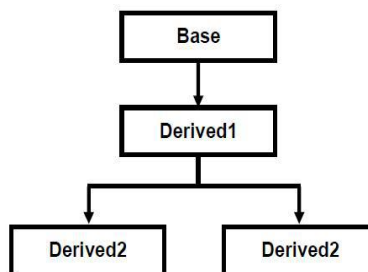
```

};
class C: public A
{
    int e,f;
    public :
    void input()
    {
        cout<<"\n Enter the value of e and f";
        cin>>e;>>f
    }
    void output()
    {
        cout<<"\nthe value  of e is"<<e"and f is" <<f;
    }
    void main()
    {
        B obj1
        C obj2;
        obj1.getdata(); //member function of class A
        obj1.indata(); //member function of class B
        obj2.getdata(); //member function of class A
        obj2.input(); //member function of class C
        obj1.putdata(); //member function of class A
        obj1.outdata(); //member function of class B
        obj2.output(); //member function of class A
        obj2.outdata(); //member function of class C
    }
}

```

5. Hybrid Inheritance: This is a Mixture of two or More Inheritance types.

I.e.: Any combination of single, hierarchical and multi level inheritances is called as hybrid inheritance.



**Example:** #include <iostream.h>

```

class A
{
    protected:
    int rollno;
    public:
    void get_num(int a)
    {
        rollno = a;
    }
    void put_num()
    {

```

```

        cout << "Roll Number Is:"<< rollno << "\n"; }
    };

class B : public A
{
protected:
    int sub1;
    int sub2;
public:
    void get_marks(int x,int y)
    {
        sub1 = x;
        sub2 = y;
    }

    void put_marks(void)
    {
        cout << "Subject 1:" << sub1 << "\n";
        cout << "Subject 2:" << sub2 << "\n";
    }
};

class C
{
protected:
    float e;
public:
    void get_extra(float s)
    {
        e=s;
    }

    void put_extra(void)
    {
        cout << "Extra Score:." << e << "\n";}
};

class D : public B, public C
{
protected:
    float tot;
public:
    void disp(void)
    {
        tot = sub1+sub2+e;
        put_num();
        put_marks();
        put_extra();
        cout << "Total:"<< tot;
    }
};

int main()
{
    D std1;
    std1.get_num(10);
    std1.get_marks(10,20);
    std1.get_extra(33.12);
    std1.disp();
    return 0;
}

```

### **Result:**

Roll Number Is: 10  
 Subject 1: 10  
 Subject 2: 20

Extra score:33.12  
Total: 63.12

### 2.3. Implementing polymorphism

Polymorphism is a generic term that means 'many shapes/forms '.

Polymorphism is a mechanism that allows you to implement a function in different ways.

In C++ the simplest form of Polymorphism is overloading of functions or operators.

Typically, polymorphism occurs when there is a hierarchy of classes and they are related by inheritance.

C++ provides three different types of polymorphism.

- **Virtual functions:** A virtual function is a member function that is declared within a base class and redefined by a derived class.
- **Function name overloading:** It is simply defined as the ability of one function to perform different tasks.
- **Operator overloading:** Operator overloading is the ability to tell the compiler how to perform a certain operation when its corresponding operator is used on one or more variables.

### 2.4. Introduction to Delegates and Events

#### Delegates

A delegate is similar to a function pointer that allows the programmer to encapsulate a reference to a method inside a delegate object. The delegate object can then be passed to code which can call the referenced method, without having to know at compile time which method will be invoked.

#### Events

Events are the actions that are performed by the user during the applications usage.

If a user clicks a mouse button on any object, then the Click event occurs.

If a user moves the mouse, then the mouse move event occurs.

By the same way an application can generate Key down event, Key up event, mouse double click event.

Delegate and Event concepts are completely tied together. Delegates are just function pointers, That is, they hold references to functions.

### 2.5. Introduction to Exception Handling

An exception is a special condition that changes the normal flow of program execution.

Exceptional conditions are things that occur in a system that are not expected or are not a part of normal system operation. When the system handles these exceptional conditions improperly, it can lead to failures and system crashes.

Exception handling is the method of building a system to detect and recover from exceptional conditions.

Exceptional conditions are any unexpected occurrences that are not accounted for in a system's normal operation.

Some examples of exceptional conditions:

- incorrect inputs from the user
- bit level memory or data corruption

If these exceptional conditions are not properly caught and handled, they can cause an error or failure in the system.

An exception is a problem that arises during the execution of a program. A C++ exception is a response to an exceptional circumstance that arises while a program is running, such as an attempt to divide by zero.

Exceptions provide a way to transfer control from one part of a program to another. C++ exception handling is built upon three keywords: **try**, **catch**, and **throw**.

- **throw:** A program throws an exception when a problem shows up. This is done using a **throw** keyword.
- **catch:** A program catches an exception with an exception handler at the place in a program where you want to handle the problem. The **catch** keyword indicates the catching of an exception.
- **try:** A **try** block identifies a block of code for which particular exceptions will be activated. It's followed by one or more catch blocks

**Example:**

```
#include <iostream.h>
double division(int a, int b)
{
```

```

        if( b == 0 )
        {
            throw "Division by zero condition!";
        }
        return (a/b);
    }
}

int main ()
{
    int x = 50;
    int y = 0;
    double z = 0;
    try {
        z = division(x, y);
        cout << z << endl;
    } catch (const char* msg)
    {
        cerr << msg << endl;
    }

    return 0;
}

```

## 2.6. Using Attributes and Overloading Operators

Attributes are a means of decorating your code with various properties at compile time.

**Operator overloading** is the ability to tell the compiler how to perform a certain operation when its corresponding operator is used on one or more variables. Example

- Overloading Unary Operators
- Overloading Increment and Decrement
- Overloading Binary Operators
- Overloading Assignments
- Overloading Function Calls
- Overloading Subscripting
- Overloading Class Member Access

## 2.7. Encapsulation

**Encapsulation** is the method of combining the data and functions inside a class. This hides the data from being accessed from outside a class directly, only through the functions inside the class is able to access the information.

Encapsulation is the term given to the process of hiding all the details of an object that do not necessary to its user.

Encapsulation enables a group of properties, methods and other members to be considered a single unit or object.

Features and Advantages of the concept of Encapsulation:

- Makes Maintenance of Application Easier
- Improves the Understandability of the Application
- Enhanced Security
- Protection of data from accidental corruption
- Flexibility and extensibility of the code and reduction in complexity



### **LO3. Debug code**

#### **3.1. Using an Integrated Development Environment**

An **integrated development environment (IDE)** is a software application that provides comprehensive facilities to computer programmers for software development. An IDE normally consists of a source code editor, build automation tools, compiler or interpreter and a debugger.

Ie: An integrated development environment (IDE) is a programming environment that consisting of a code editor, a compiler, a debugger, and a graphical user interface (GUI) builder.

#### **3.2. The Language Debugging Facilities**

This topic describes methods of debugging routines in Language Environment. Debug tools are designed to help you to detect errors early in your routine.

Debug Tool also provides facilities for setting breakpoints and altering the contents and values of variables.

##### **3.2.1. Visual C++, C#, ASP.Net**

Visual C++ program is an application development tool developed by Microsoft for C++ programmers that supports object-oriented programming with an integrated development environment (IDE).

##### **3.2.2. Visual Studio suite**

###### **What Is Visual Studio?**

The Microsoft Visual Studio development system is a suite of development tools designed to aid software developers. It is used to develop graphical user interface applications along with Windows Forms applications, web sites, web applications, and web services.

###### **System Requirement to install Visual Studio.net 2008:**

<b>System Components</b>	<b>Recommended Configuration</b>
Processor	Minimum 1.6 GHz CPU, recommended 2.2 GHz or higher CPU
RAM	Minimum 384 MB RAM, recommended 1024 MB or more RAM
Hardisk Space	Minimum 5400 RPM hard disk, recommended 7200 RPM or higher hard disk
Supporting Operating System	Microsoft Windows XP , Microsoft Windows Server 2003 , Windows Vista, or Latest version
CD ROM or DVD Drive	Required
VGA	Minimum 1024x768 display, recommended 1280x1024 display

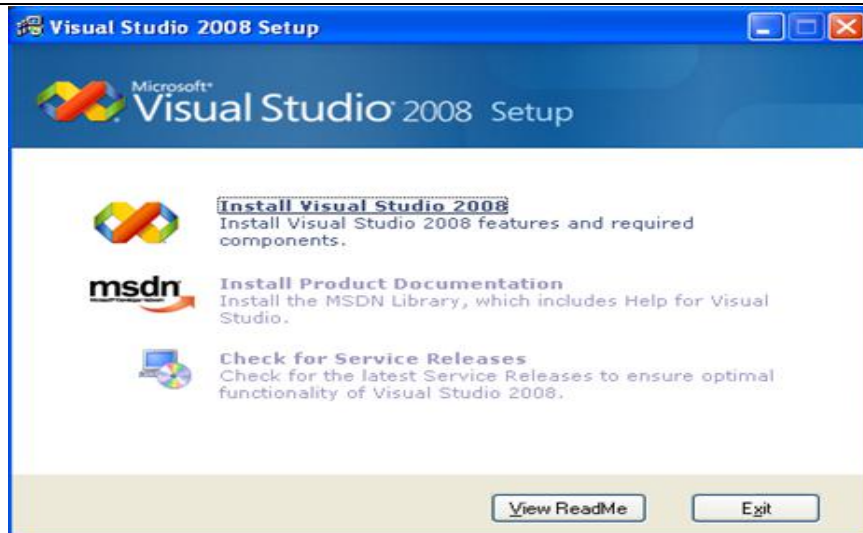
#### **How to Install Visual Studio.net 2008**

Visual Studio can be installed from a CD or by downloading from Microsoft Website at

<http://www.microsoft.com/express/downloads/>.

The downloaded file would be a .exe file, when double clicked on, it will open up an wizard.

By choosing the location, the type of installation like custom or standard user can install only the required applications like Visual Basic.net 2008 from the available choices.



The setup wizard will start copying needed files into a temporary folder. Just wait.

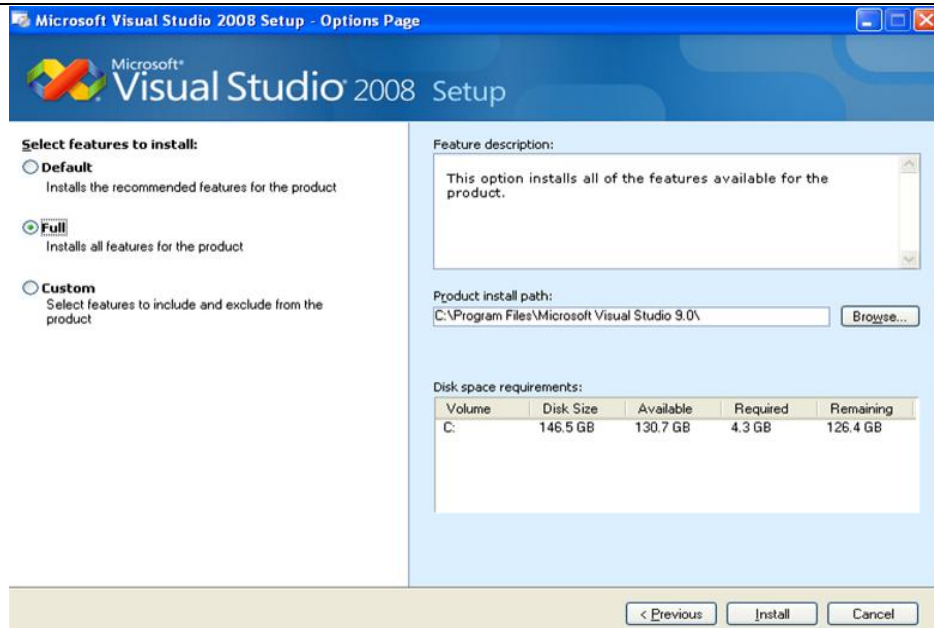


In the welcome setup wizard page you can enable the tick box to send your setup experience to Microsoft if you want .in this case we just leave it unchecked. Just wait for the wizard to load the installation components. Click the next button to go to the next step



The setup wizard will list down all the required components need to be installed. Notice that visual studio 2008 needs .Net Framework version 3.5. Then click the next button.

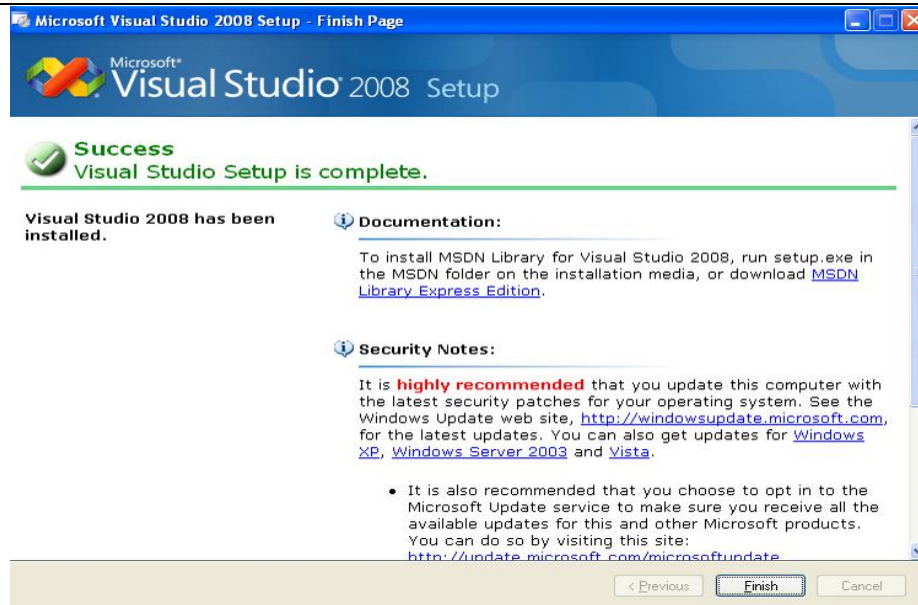
In the installation type, there are three choices: **default**, **full** or **custom**. In our case, select the full and click the install button. Full installation required around 4.3GB of space



The installation starts. Just wait and see the step by step, visual studio 2008 components being installed.



Any component that failed to be installed will be marked with the Red Cross mark instead of the green tick for the successful. In this case we just exit the setup wizard by clicking the Finish button.



Click Restart Now to restart you machine.



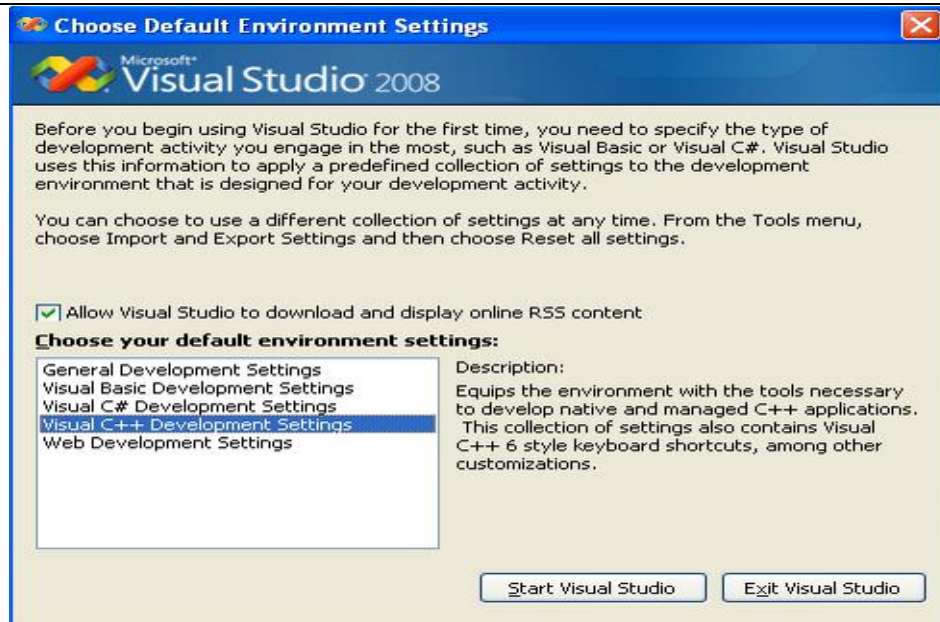
The Windows Start menu for Visual Studio 2008 is shown below.



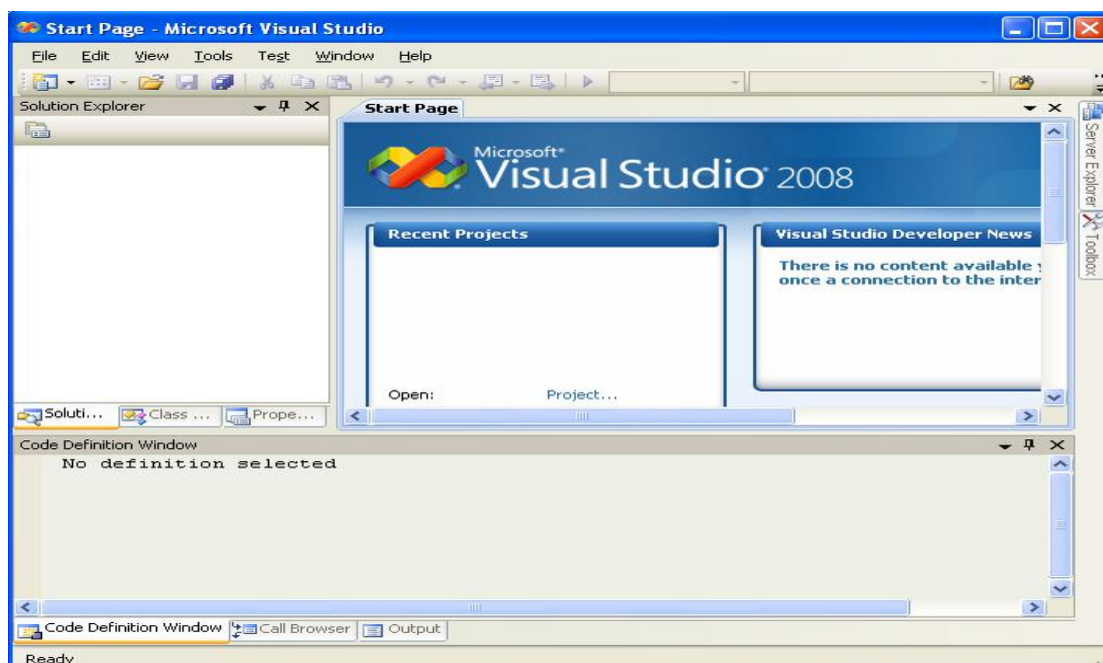
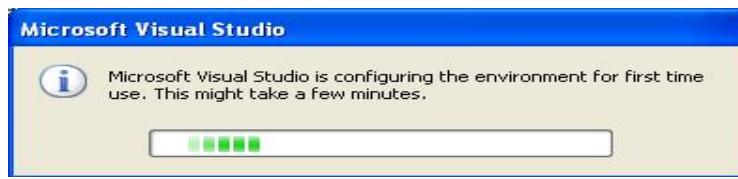
Depending on your programming needs, you will select one of the visual studio component Settings.

Name of trainer:

Date: \_\_\_\_/\_\_\_\_/04



The Visual Studio 2008 is configuring the development environments to the chosen one for the first time use.



## Create a Project in VB.NET

A Visual Basic **Project** is container for the forms, methods, classes, controls that are used for a Visual Basic

Name of trainer:

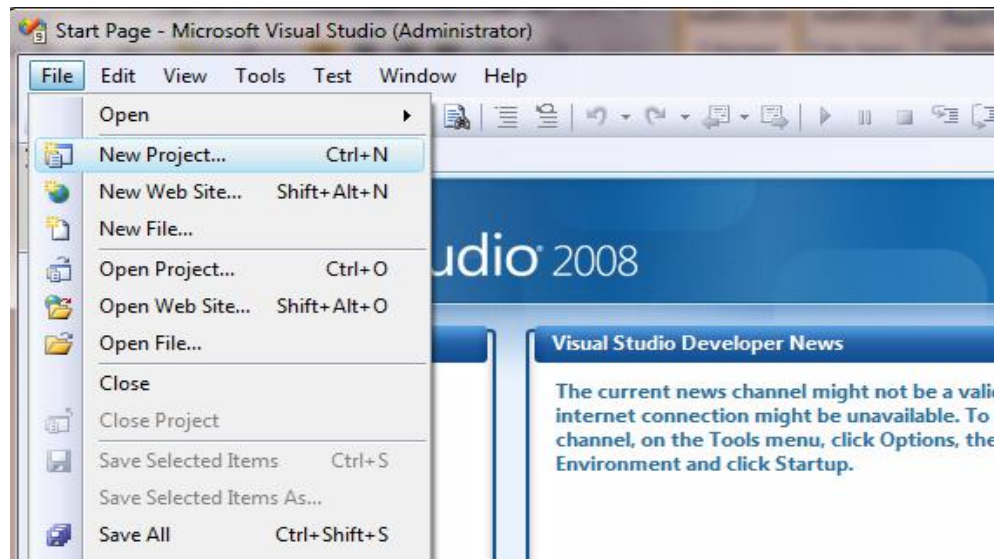
Date: \_\_\_\_/\_\_\_\_/04



Application.

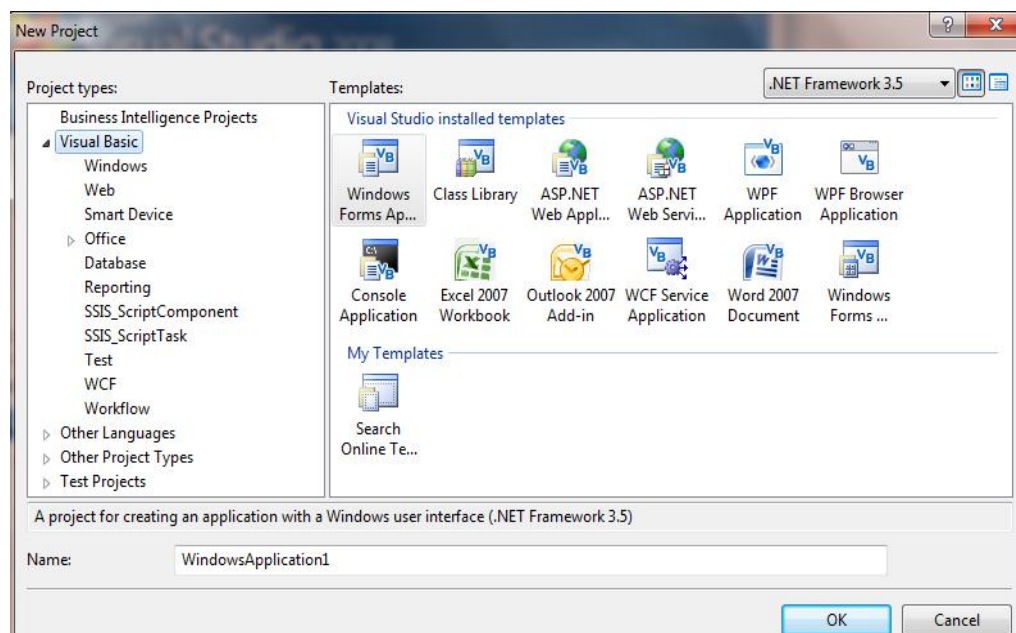
**Steps to Create a VB.net Project:**

1. Click on the Programs->Microsoft Visual Studio.net 2008.
2. Choose **File -->New Project** from the **Menu Bar** to get the **New Project** window.

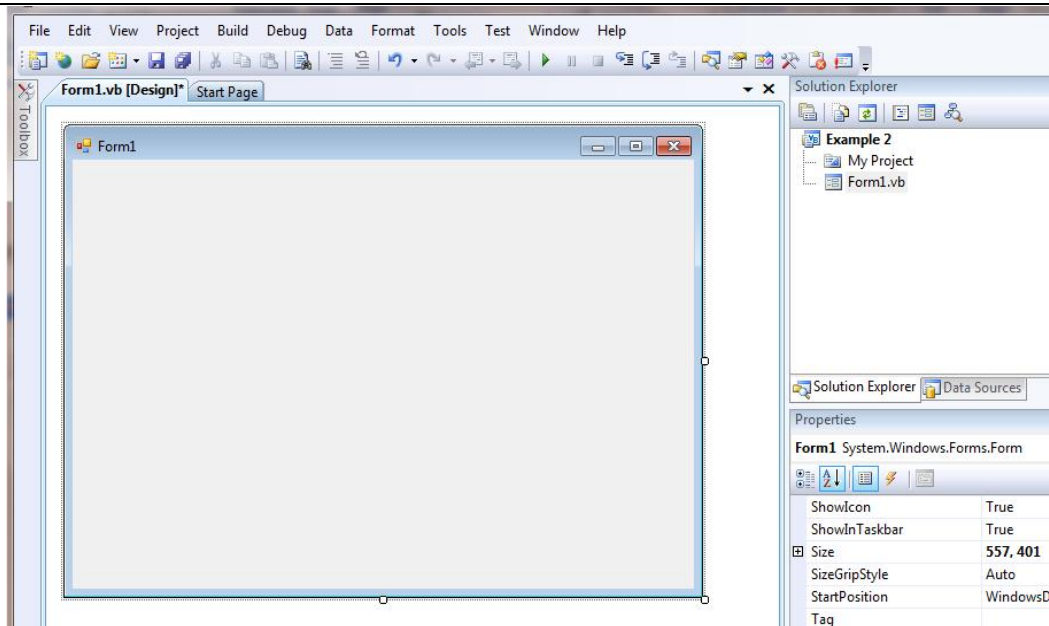


3. Select the type of Project as per the requirement from following choices **Windows Application, Class Library, Console Application, Windows Control Library, Web Control Library, Windows Service, Empty Project, and Crystal Reports.**

To develop a Windows Based Application, Choose **Windows Application**, and fill in a **Name** for the application



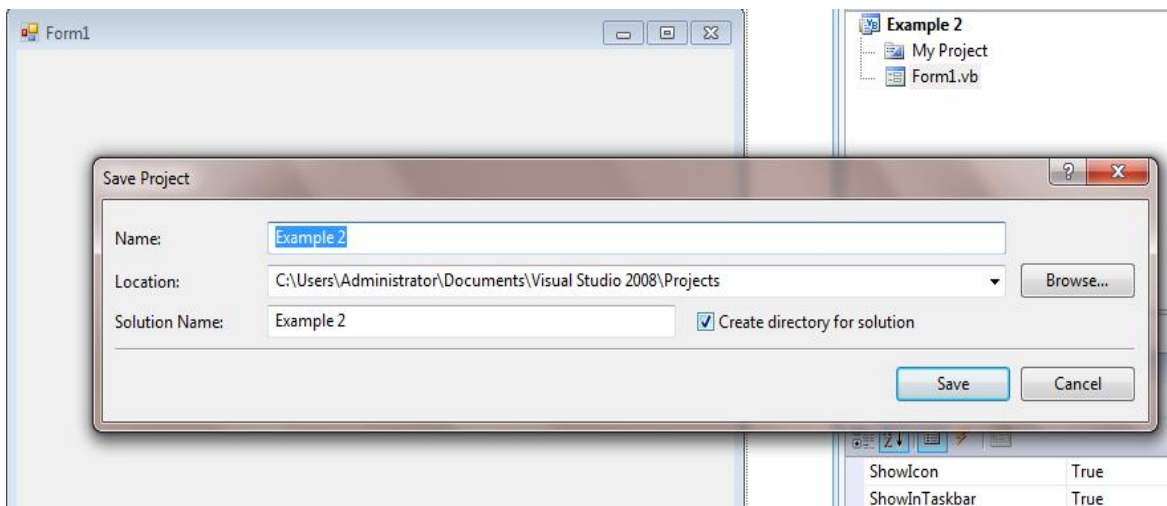
By default, a project named **My Project** and a form **Form1** will be created. Project name and form name can be renamed later.



#### 4. Saving Project in VB.Net

There are many ways for *saving a project* created using VB.Net 2008. But the recommended option is to browse to **File->Save All**. This option saves all the files associated with a project.

5. Provide a **Name** that will be populated as the **Solution Name**, also specify the location to save the project.



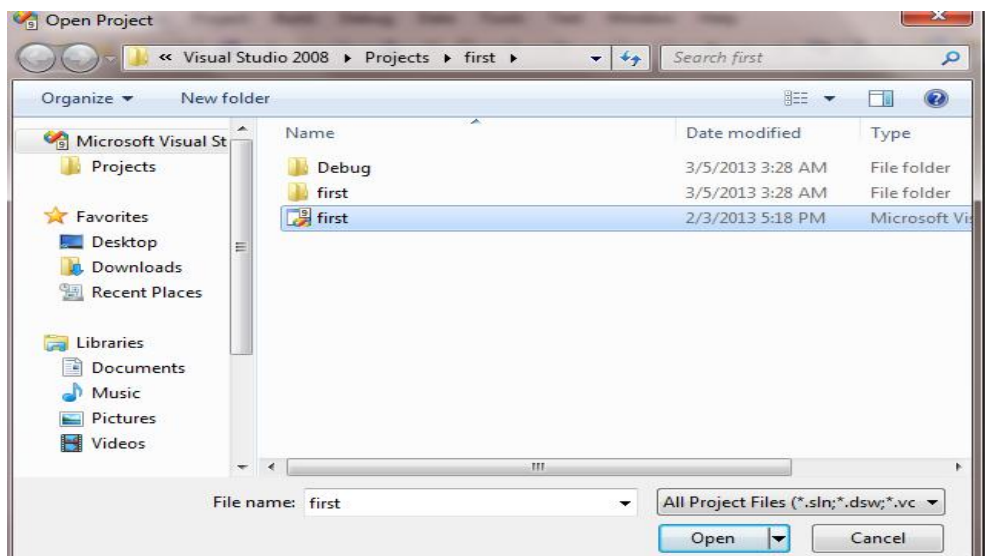
#### 6. Run a Visual Basic .Net 2008 Project

Run a Visual Basic .Net 2008 Project by pressing the **F5** key or by choosing **Debug -> Start Debugging**.

#### 7. Open Existing Project in VB.NET

In Visual Studio.net 2008, an existing project can be opened using the **Recent Projects** option in the **Start Page** or can be opened using the **File -> Open Project** from the Menu Bar. Both these displays a window with project

folder, once the project is selected and opened using the file with the extension **.sln** for windows application, the **Solution Browser** displays all the components of that project.

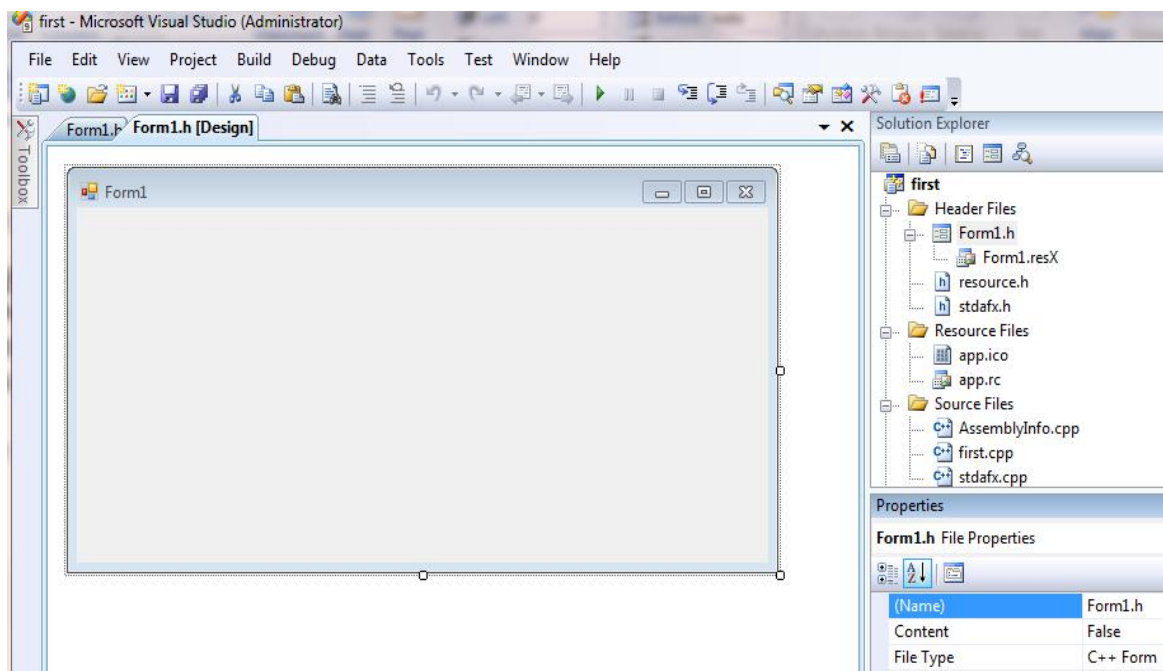


## Visual Basic.net 2008 IDE

VB *Integrated Development Environment (IDE)* consists of inbuilt compiler, debugger, editors, and automation tools for easy development of code.

I.e. it consists of Solution Explorer, Toolbox, Form, Properties Window, and Menu Bar.

The following is the screen shot of the IDE of Visual Studio.net 2008.

























## **Menu Bar**

**Menu bar** in Visual Basic.net 2008 consist of the commands that are used for constructing a software code. These commands are listed as menus and sub menus.



Menu bar also includes a simple Toolbar, which lists the commonly used commands as buttons. This Toolbar can be customized, so that the buttons can be added as required.










Following table lists the Toolbars Buttons and their actions.






Button	Description		
	Adds a new Project.		Uncomment the selected lines.
	Open a New Window.		Undo.
	Open a File.		Redo.
	Saves the Current Form.		Continue Debugging.
	Saves all files related to a project.		Break Debugging.
	Cut the selection.		Stop Debugging.
	Copy the selection.		Displays Solution Explorer.
	Paste the selection.		Displays Properties Window.
	Find the searched text.		Displays Object Browser.
	Comment out selected lines.		Displays ToolBox Window.
			Displays Error List Window.
			Displays Command Window.

### Tool Box

**Toolbox** in Visual Basic.net 2008, consist of Controls, Containers, Menu Options, Data Controls, Dialogs, Components, Printing controls, that are used in a form to design the interfaces of an application.

The following table lists the Common Controls listed in the Toolbox.

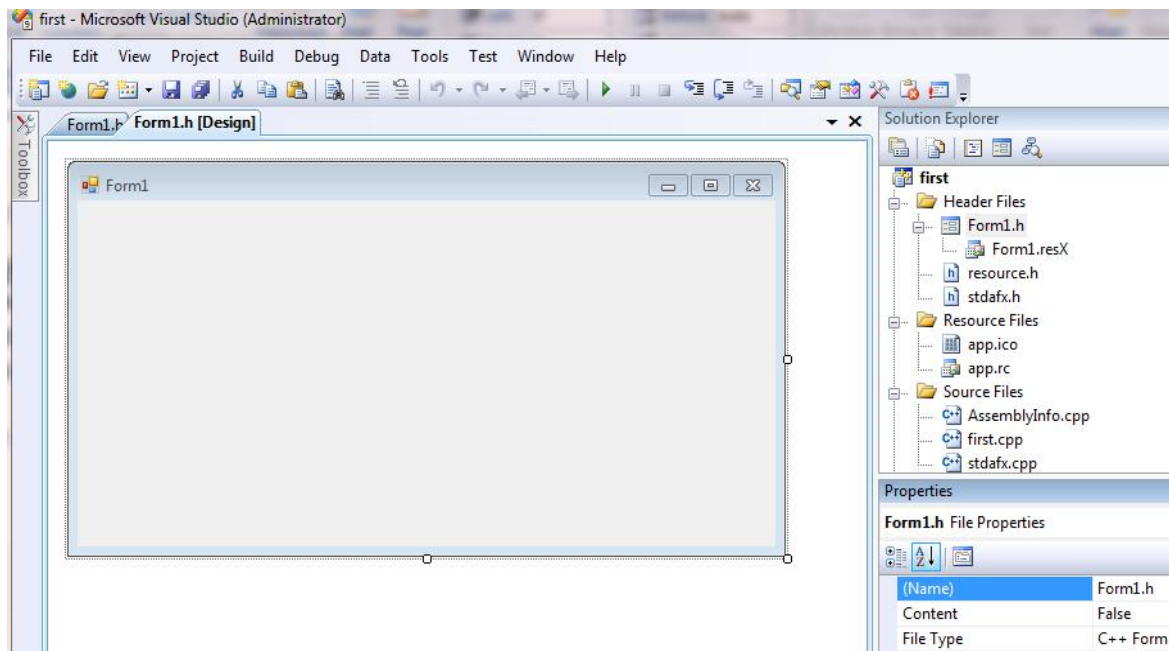
Images	Control Name	Description
	Pointer	Used to move and resize controls and forms.
	Button	This Control triggers an action when accessed.
	Check Box	Control that has values either true or false
	CheckedList Box	Lists check box next to each item
	Combo Box	A combination of list and text box controls that enables to select as well as edit text.
	Label	Displays a label text.
	LinkLabel	Displays a label with a link text.
	List Box	Control that lists number of items.
	List View	Extension of ListBox control with options to add icons,headings.

	Picture Box	Display image files
	Progress Bar	Display the progress of a task.
	Radio Button	Allows to choose a choice from a group of choices.
	Text Box	Control used to input or display text.
	ToolTip	Displays tooltip text.

### Solution Explorer

***Solution Explorer*** in Visual Basic.net 2008 lists of all the files associated with a project. It is docked on the right under the Toolbar in the VB IDE. In VB 6 this was known as 'Project Explorer'

Solution Explorer has options to view the code, form design, and refresh listed files. Projects files are displayed in a drop down tree like structure, widely used in Windows based GUI applications.

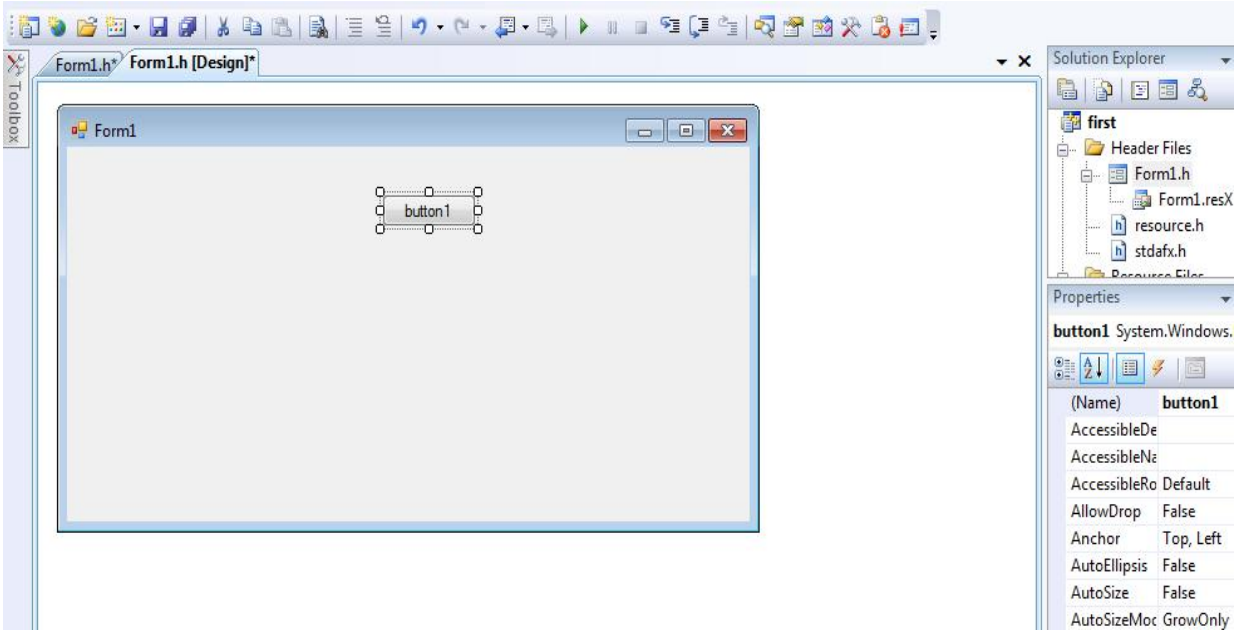


### Properties Window

***Windows form properties*** in Visual Basic.net 2008 lists the properties of a selected object. Every object in VB has its own properties that can be used to change the look and even the functionality of the object.

Properties Window lists the properties of the forms and controls in an alphabetical order by default.

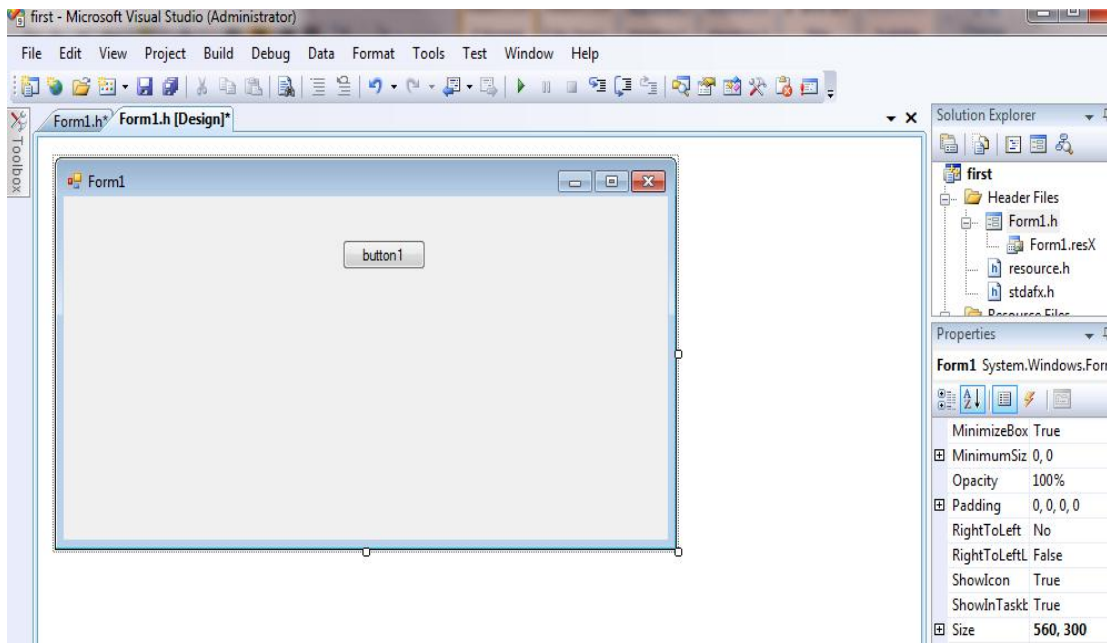
Following is the screen shot of Properties Window.



## Visual Basic Forms

Forms in Visual Basic.net 2008 are the basic object used to develop an application; it also contains the coding as well as the controls embedded in it.

Following is the screenshot of a Form in Visual Basic.net 2008.



A **form** is created by default when a **Project** is created with a default name **Form1**. Every form has its own Properties, Methods and Events. Usually the form properties name, caption are changed as required, since multiple forms will be used in a Project.

## Form Properties

The developers may need to alter the properties of the forms in VB.net.

Following table lists some important Properties of Forms in Visual Basic.net 2008.

<u>Properties</u>	<u>Description</u>
BackColor	Set's the background color for the form
BackgroundImage	Set's the background image for the form
AllowDrop	Specifies whether to accept the data dragged and dropped onto the form.
Font	Get or sets the font used in the form
Locked	Specifies whether the form is locked.
Text	Provide the title for a Form Window
Control Box	Determines whether the ControlBox is available by clicking the icon on the upper left corner of the window.
MaximizaBox	Specifies whether to display the maximize option in the caption bar of the form.
MinimizeBox	Specifies whether to display the minimize option in the caption bar of the form.

### **Show / Hide Forms**

**Form** in Visual Basic.net 2008 are displayed and hidden using two different methods. To display a form the, **Show()** method is used and to hide a form, **Hide()** method is used.

**Show Method:** This method is used to display the form on top of all other windows.

**Syntax:** FormName.Show()

**Hide Method:** This method hides a form object from the screen, still with the object being loaded in the memory.

**Syntax:** FormName.Hide()

### **Example:**

```
Public Class Form1
Private Sub Button1_Click(ByVal sender As System.Object,ByVal e As System.EventArgs)Handles Button1.Click
    FormName.Show()
End Sub
Private Sub Button2_Click(ByVal sender As System.Object,ByVal e As System.EventArgs) Handles Button2.Click
    Me.Hide()
End Sub
End Class
```

## Visual Basic.net 2008 Data types

**Data Type** in Visual Basic.net 2008 defines the type of data a programming element should be assigned, how it should be stored and the number of bytes occupied by it.

Following are the common data types used in Visual Basic.Net 2008.

Boolean
Char
Date
Double

Integer
Long
String
Single

## Constants in VB.NET

**Constants in VB.NET 2008** are declared using the keyword **Const**. Once declared, the value of these constants cannot be altered at run time.

### Syntax:

```
[Private | Public | Protected ]  
Const constName As datatype = value
```

In the above syntax, the **Public** or **Private** can be used according to the scope of usage. The **Value** specifies the unchangeable value for the constant specified using the name **constName**.

### Example:

```
Public Class Form1  
    Public Const PI As Double = 3.14  
    Private Sub Button1_Click(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles Button1.Click  
        Dim r As Single  
        r = Val(TextBox1.Text)  
        TextBox2.Text = PI * r * r  
    End Sub  
End Class
```

## Visual Basic.net 2008 Variables

**Variables** in VB.net 2008 are used to store values and also they have a data type and a name.

### Naming Convention:

- Variables in Visual Basic should start with an alphabet or a letter and should not contain any special characters like %, &, !, #, @ or \$.
- The variable should not exceed 255 characters.

### Scope of Variables:

- A variable declared in the general declaration of a form can be used in all the procedures.
- Variables declared inside a procedure will have a scope only inside the procedure, so they are declared using the **Dim** keyword.

### Example

```
Public Class Form1  
    Private Sub Button1_Click(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles Button1.Click  
        Dim c As Integer  
        c = Val(TextBox1.Text) + Val(TextBox2.Text)  
        TextBox3.Text = c  
    End Sub  
End Class
```

## User Defined Data Types

User defined data type in VB.net 2008 is a collection of variables of different primitive data types available in Visual Basic.net 2008 combined under a single user defined data type. This gives the flexibility to include real time objects into an application.

User defined data types are defined using a **Structure** Statement and are declared using the **Dim** Statement.

### Structure Statement:

The **Structure** statement is declared in the general declaration part of form or a Module.

#### Syntax:

```
[ Public | Protected | Private]
Structure varname elementname [(subscripts)] As type
    [elementname [(subscripts)] As type]
    ...
End Structure
```

In the above syntax **varname** is the name of the user defined data type, that follows the naming convention of a variable. **Public** option makes these datatypes available in all projects, modules, classes. **Type** is the primitive datatype available in visual basic.

### Dim Statement:

This is used to declare and allocate a storage space for a variable or an user defined variable.

Syntax: Dim variable [As Type]

#### Example:

```
Structure EmpDetails
    Dim EmpNo As Integer
    Dim EmpName As String
    Dim EmpSalary As Integer
    Dim EmpExp As Integer
End Structure

Public Class Form1
    Private Sub Button1_Click(ByVal sender As System.Object, ByVal e As System.EventArgs)
        Handles Button1.Click
            Dim TotalSal As New EmpDetails()
            TotalSal.EmpNo = TextBox1.Text
            TotalSal.EmpName = TextBox2.Text
            TotalSal.EmpSalary = TextBox3.Text
            TotalSal.EmpExp = TextBox4.Text
            TextBox5.Text = Val(TotalSal.EmpSalary) * Val(TotalSal.EmpExp)
        End Sub
    End Class
```

## If Then Statement

*If Then statement* is a control structure which executes a set of code only when the given condition is true.

#### Syntax:

```
If [Condition] Then
    [Statements]
```

In the above syntax, when the **Condition** is true then the **Statements** after **Then** are executed.

#### Example:

```
Private Sub Button1_Click_1(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles Button1.Click
    If Val(TextBox1.Text) > 25 Then
        TextBox2.Text = "Eligible"
    End If
```

## If Then Else Statement

*If Then Else* statement is a control structure which executes different set of code statements when the given condition is true or false.

**Syntax:** If [Condition] Then  
    [Statements]  
Else  
    [Statements]

In the above syntax when the **Condition** is true, the **Statements** after **Then** are executed.

If the condition is false then the statements after the **Else** part is executed.

**Example:**

```
Private Sub Button1_Click(ByVal sender As System.Object,ByVal e As System.EventArgs) Handles Button1.Click
    If Val(TextBox1.Text) >= 40 Then
        MsgBox("GRADUATED")
    Else
        MsgBox("NOT GRADUATED")
    End If
End Sub
```

**Nested If Then Else Statement**

*Nested If..Then..Else* statement is used to check multiple conditions using if then else statements nested inside one another.

**Syntax:** If [Condition] Then  
    If [Condition] Then  
        [Statements]  
    Else  
        [Statements]  
Else  
    [Statements]

In the above syntax when the **Condition** of the first if then else is true, the second if then else is executed to check another two conditions. If false the statements under the Else part of the first statement is executed.

**Example:**

```
Private Sub Button1_Click(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles Button1.Click
    If Val(TextBox1.Text) >= 40 Then
        If Val(TextBox1.Text) >= 60 Then
            MsgBox("You have FIRST Class")
        Else
            MsgBox("You have SECOND Class")
        End If
    Else
        MsgBox("Check your Average marks entered")
    End If
End Sub
```

**Select Case Statement**

*Select case statement* is used when the expected results for a condition can be known previously so that different set of operations can be done based on each condition.

**Syntax:** Select Case Expression

```
Case Expression1
    Statement1
Case Expression2
    Statement2
Case Expressionn
    Statementn
...
Case Else
    Statement
End Select
```

In the above syntax, the value of the **Expression** is checked with **Expression1..n** to check if the condition is true. If none of the conditions are matched the statements under the **Case Else** is executed.

**Example:**

```
Private Sub Button1_Click(ByVal sender As System.Object,ByVal e As System.EventArgs) Handles Button1.Click
```

```

Dim c As String
c = TextBox1.Text
Select c
Case "Red"
    MsgBox("Color code of Red is::#FF0000")
Case "Green"
    MsgBox("Color code of Green is::#808000")
Case "Blue"
    MsgBox("Color code of Blue is:: #0000FF")
Case Else
    MsgBox("Enter correct choice")
End Select
End Sub

```

In the above example based on the color input in **TextBox1**, the color code for RGB colors are displayed, if the color is different then the statement under **Case Else** is executed.

### **While Statement**

**While Statement** is a looping statement where a condition is checked first, if it is true a set of statements are executed.

#### **Syntax:**

```

While condition
    [statements]
End

```

In the above syntax the **Statements** are executed when the **Condition** is true.

#### **Example:**

```

Private Sub Form1_Load(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles MyBase.Load
    Dim n As Integer
    n = 1
    While n <= 1
        n = n + 1
        MsgBox("First incremented value is:" & n)
    End While
End Sub

```

### **Do Loop While Statement**

**Do Loop While** Statement executes a set of statements and checks the condition; this is repeated until the condition is true.

#### **Syntax:**

```

Do
    [Statements]
Loop While [Condition]

```

In the above syntax the **Statements** are executed first then the **Condition** is checked to find if it is true.

#### **Example 1:**

```

Private Sub Form1_Load(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles MyBase.Load
    Dim cnt As Integer
    Do
        cnt = 10
        MsgBox("Value of cnt is:" & cnt)
    Loop While cnt <= 9
End Sub

```

#### **Example 2:**

```

Private Sub Form1_Load(ByVal sender As System.Object,
    ByVal e As System.EventArgs) Handles MyBase.Load
    Dim X As String
    Do
        X$ = InputBox$("Correct Password Please")
    Loop Until X$ = "Ranger"
End Sub

```



In the above Do Until Loop example, a input box is displayed until the correct password is typed.

### **For Next Loop Statement**

**For Next Loop** Statement executes a set of statements repeatedly in a loop for the given initial, final value range with the specified step by step increment or decrement value.

#### **Syntax:**

```
For counter = start To end [Step]
    [Statement]
Next [counter]
```

In the above syntax the **Counter** is range of values specified using the **Start ,End** parameters. The **Step** specifies step increment or decrement value of the counter for which the statements are executed.

#### **Example:**

```
Private Sub Form1_Load(ByVal sender As System.Object,
    ByVal e As System.EventArgs) Handles MyBase.Load
    Dim i As Integer
    Dim j As Integer
    j = 0
    For i = 1 To 10 Step 1
        j = j + 1
        MsgBox("Value of j is: " & j)
    Next i
End Sub
```

#### **Description:**

In the above For Next Loop example the counter value of i is set to be in the range of 1 to 10 and is incremented by 1. The value of j is increased by 1 for 10 times as the loop is repeated.

#### **Exercise:** For loop code

##### Syntax:

```
Dim variables As datatype
For initializedValue To endValue
    Loop body
MsgBox answer
```

Example: Find the factorial of a number n inputted from the keyboard using **for** loop.

```
Dim fact, i As Integer
fact = 1
For i = 1 To Val (TextBox1.Text)
    fact = fact * i
Next i
TextBox2.Text = fact
```

#### **While loop code**

##### **Syntax:** While condition

```
[statements]
End while
```

Example: Find the factorial of a number n inputted from the keyboard using **while** loop.

```
Dim fact, i As Integer
fact = 1
i = 1
While (i <= Val (TextBox1.Text))
    fact = fact * i
    i = i + 1
End While
TextBox2.Text = fact
```

#### **Do ... while code**

```
Do
    statement-block
Loop While condition
```

Do  
statement-block  
Loop Until condition

Example: find the factorial of a number n inputted from the keyboard using **do...while** or **do ...until** loop.

```
Dim fact, i As Integer
fact = 1
i = 1
Do
    fact = fact * i
    i = i + 1
Loop While (i <= Val(TextBox1.Text))
TextBox2.Text = fact
```

Or

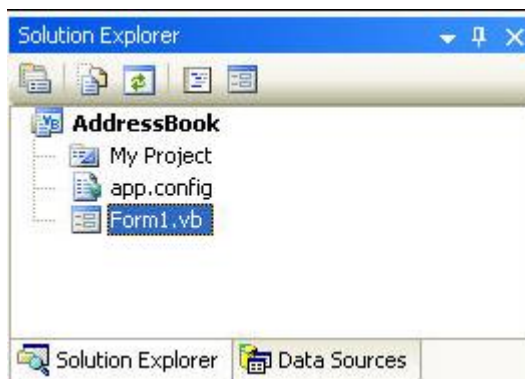
```
Dim fact, i As Integer
fact = 1
i = 1
Do
    fact = fact * i
    i = i + 1
Loop until (i > Val(TextBox1.Text))
TextBox2.Text = fact
```

## **Steps to create a connection to SQL Database VB.Net**

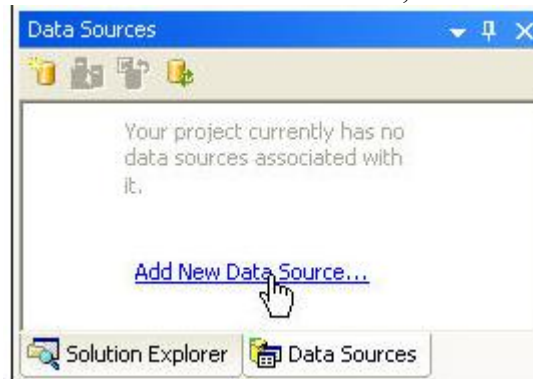
### **1. Using wizard:**

Once you have your VB software open, do the following:

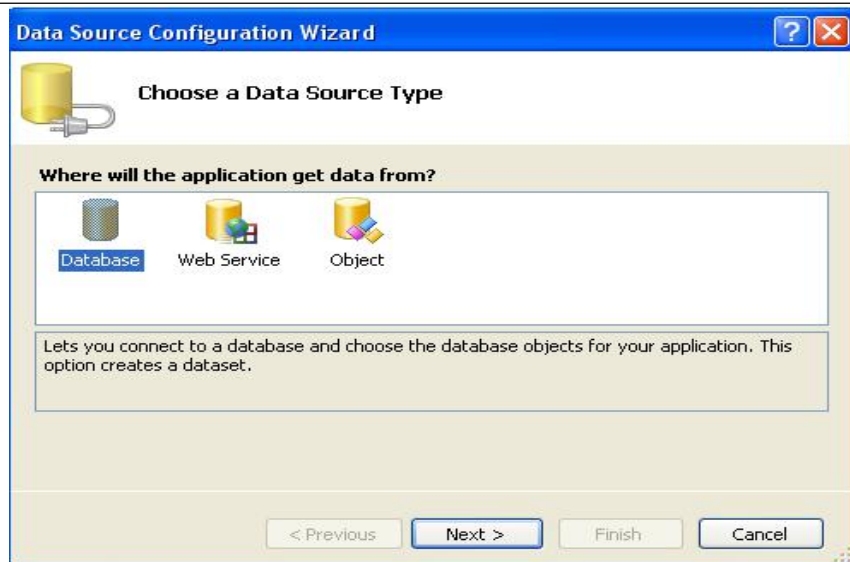
- Click **File > New Project** from the menu bar
- Select **Windows Application**, and then give it the **Name**. Click OK
- Locate the **Solution Explorer** on the right hand side.



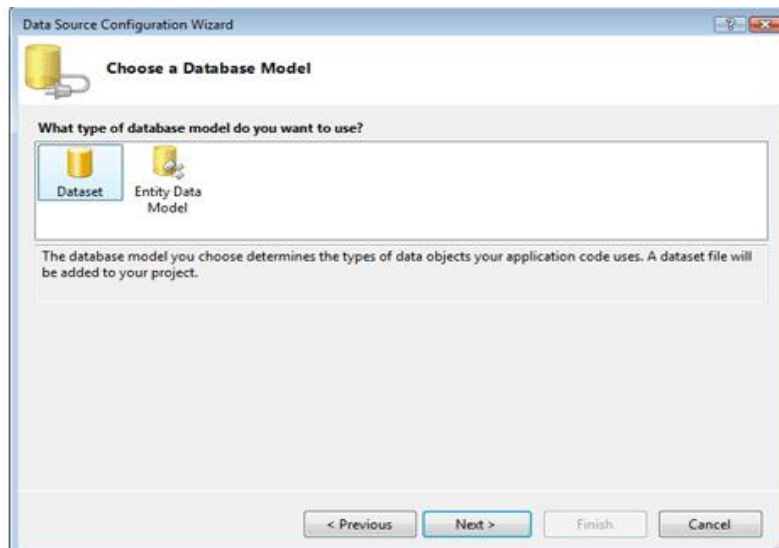
We need to select Show Data Source from data on the menu bar. Then, click on Add New Data Source.



When you click on the link Add a New Data Source, you will see a screen shown bellow. Then select Database and click next.



Select DataSet and click Next.



Click the **New Connection** button.



Write the server name and your Database name. Then click on Test Connection to check the connection.

Data source:  
Microsoft SQL Server (SqlClient) Change...

Server name:  
ICT-PC\SQLEXPRESS Refresh

Log on to the server

☒ Use Windows Authentication

☐ Use SQL Server Authentication

User name:

Password:

☐ Save my password

Connect to a database

☒ Select or enter a database name:  
test1

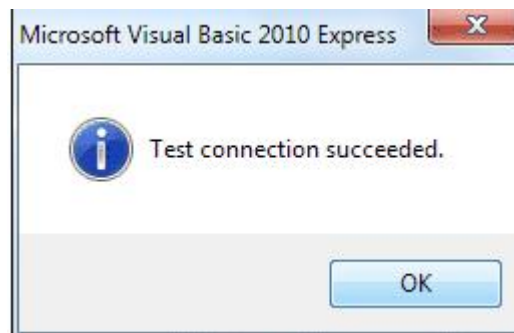
☐ Attach a database file:  
 Browse...

Logical name:

Advanced...

Test Connection OK Cancel

Click **Test Connection** to see if everything is OK.



Click on **Next**.

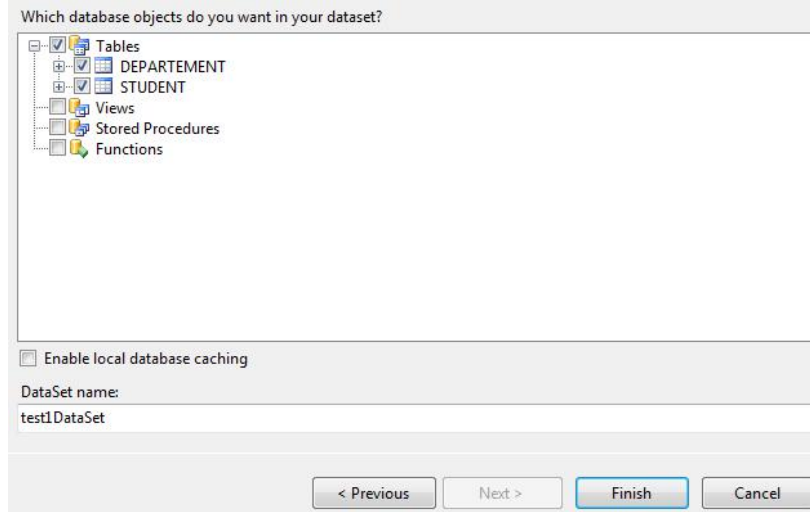
Do you want to save the connection string to the application configuration file?

☒ Yes, save the connection as:

test1ConnectionString

< Previous Next > Finish Cancel

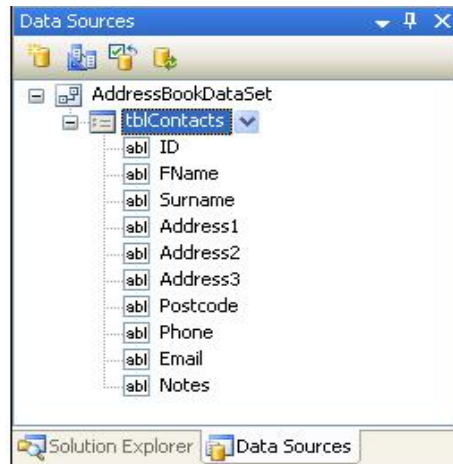
Here, you can select which tables and fields you want. Tick the **Tables** box to include them all. You can give your DataSet a name, if you prefer. Click Finish and you're done.



When you are returned to your form, you should notice your new Data Source has been added:

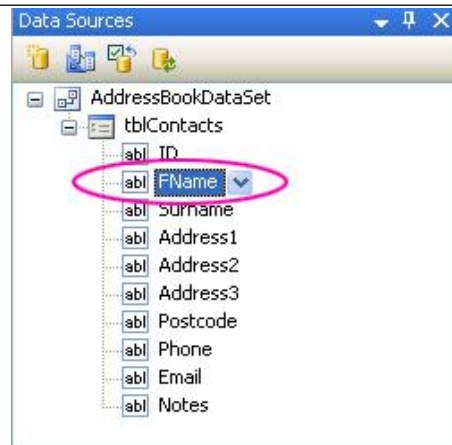


The Data Sources area of the Solution Explorer (or Data Sources tab on the left) now displays information about your database. Click the plus symbol next to **tblContacts**:



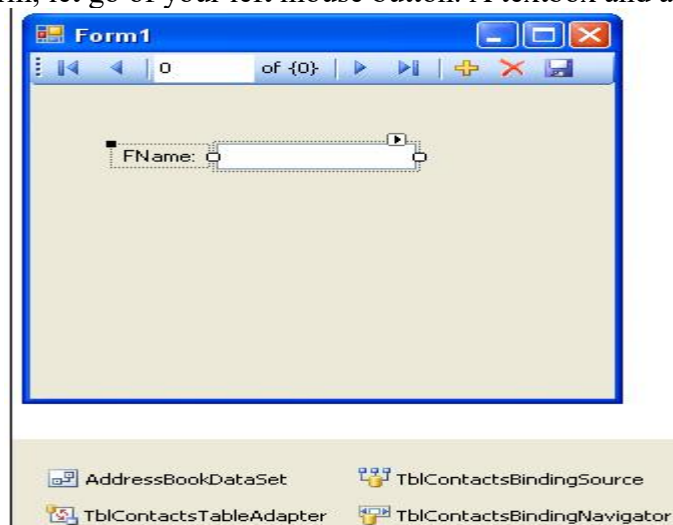
All the Fields in the Address Book database are now showing.

To add a Field to your Form, click on one in the list. Hold down your left mouse button, and drag it over to your form:

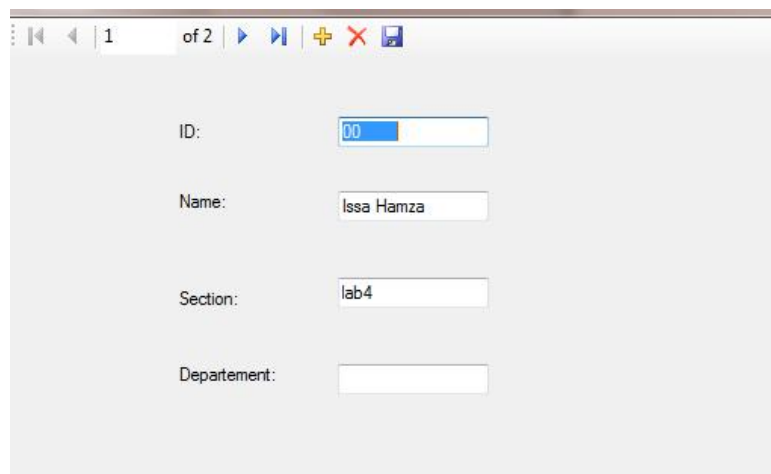


In the image above, the **FName** field is being dragged on the Form.

When your Field is over the Form, let go of your left mouse button. A textbox and a label will be added.



Click the Navigation arrows to scroll through the database.



Click the Navigation icons to move backwards and forwards through your database.

### **3.2.Using Program Debugging Techniques to Detect and Resolve Errors.**

#### **3.2.1. Errors handling**

Error handling refers to the anticipation, detection, and resolution of programming, application, and communications errors. Specialized programs, called error handlers, are available for some applications.

Errors in general come in three situations:

- Compiler errors such as undeclared variables that prevent your code from compiling.
- User data entry error such as a user entering a negative value where only a positive number is acceptable
- Run time errors such as the lack of sufficient memory to run an application or a memory conflict with another program.

A run-time error takes place during the execution of a program, and usually happens because of adverse/undesirable system parameters or invalid input data.

The above types of errors can be resolved or minimized by the use of error handler programs.

**Example: The On Error statement is an example of error handler program.**

```
On Error Goto 0      or
On Error Goto <label>:  or
On Error Resume Next
```

#### **3.2.2. Debugging options**

In computers, debugging means running (executing) the programming code to perform an action and display the result. Debugging is a necessary process in almost any new software development process.

Debugging tools (called debuggers) help identify coding errors at various development stages. Some programming language packages include a facility for checking the code for errors as it is being written.

To debug a programming code, start with a problem, isolate the source of the problem (compile), and then fix it.

#### **3.2.3. Compiling the program**

A compiler is a computer program that transforms source code written in a programming language(the source language) into another computer language (the target language, often having a binary form known as object code). The most common reason for wanting to transform source code is to create an executable program.

The name "compiler" is primarily used for programs that translate source code from a high-level programming language to a lower level language (e.g., assembly language or machine code).

A program that translates from a low level language to a higher level one is a decompiler.

#### **3.2.4. Run the application or program**

When running an application, **Select Run/Debug Configuration** drop-down list on the main toolbar or press F5 from function keys on the keyboard.

### **LO4. Document activities**

#### **4.1. Guidelines for Developing Maintainable Code**

Guidelines for developing maintainable code adhering to a set of coding standard is followed

#### **4.2. The Use of Internal Documentation Standards and Tools**

##### **4.2.1. Documentation techniques**

At the completion of the design process, a comprehensive package of documentation is assembled. Detail documentation of the system should be created during each phase of the design process. The package should contain a clear description of every facet of the new system in sufficient detail to support its development, including these aspects:

- . Examples of output.
- . Descriptions and record layouts of each file.
- . Data-flow diagrams.

- . A data dictionary.
- . Identification of individual programs.

High-level documentation provides a picture of the overall structure of the system, including input, processing, storage, and output. The general flow of information throughout the major processes of the system can be depicted in either a data-flow diagram or a systems flowchart

The purpose of detail documentation is to provide a programmer with sufficient information to write the program. The documentation should include report layouts for input and output, record layouts that define the files, and the relationship among the components of the programs.

#### **4.2.2. Program and documentation standards**

Documentation standard is the structure and presentation of documents on a software development.

#### **4.2.3. Internal documentation techniques**

There are two kinds of documentations 1) System documentation 2) User documentation

**System documentation** is detailed information about a system's design specifications, its internal workings, and its functionality.

System documentation is further divided into internal and external documentation.

- **Internal documentation** is part of the program source code or is generated at compile time.
- **External documentation** includes the outcome of all of the structured diagramming techniques such as DFD and ERD.

**User documentation** is written or visual information about an application system, how it works and how to use it.

- ⌘ The kinds of user documents are reference guide, user's guide, release description, system administrator's guide and acceptance sign-off.

### **LO5. Test code**

#### **5.1. Conducting Simple Tests to Confirm the Coding Process Meets Design Specifications**

Simple tests are developed and conducted to confirm the coding process meets design specification

##### **5.1.1. Testing techniques**

The tests performed are documented.

- The detailed specifications produced during the design phase are translated into hardware, communications, and executable software.
- The design must be translated into a machine-readable form. The code generation step performs this task. If the design is performed in a detailed manner, code generation can be accomplished without much complication. Different high level programming languages are used for coding. With respect to the type of application, the right programming language is chosen.
- Depending on the size and complexity of the system, coding can be an involved, intensive activity. Once coding is begun, the testing process can begin and proceed in parallel. As each program module is produced, it can be tested individually, then as a part of a larger program, and then as part of larger system.
- The deliverables and outcome from the coding are the code and program documentation.

##### **5.1.2. User manual**

A user manual is, also known as a user guide, is a technical communication document intended to give assistance to people using a particular system.

It is a step-by-step describes how the users can use the system. Generally the description is in detail keeping in view the fact that the target users using the system have limited knowledge about it.

##### **5.1.3. Printing documents of the program**

You may print document of the program code.

#### **5.2. Implementation of Required Corrections**

Corrections are made to the code and the documentation as needed