# F.Y.B.Sc.IT – SEM II

# Object Oriented Programming With C++ (PUSIT206T)

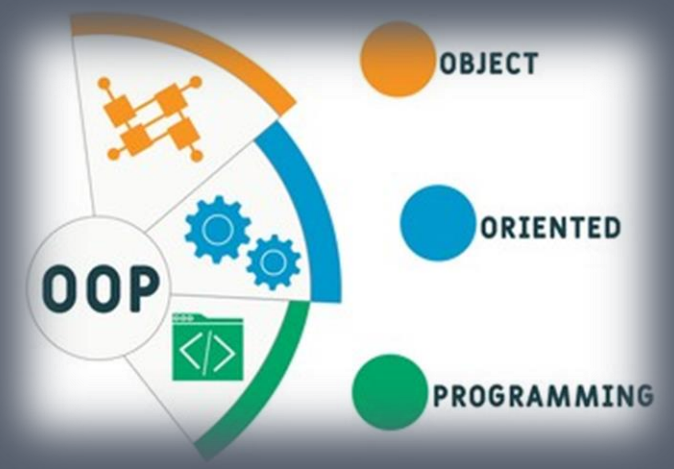
By,
Prof. Nikita Madwal

# UNIT 4

## 12. FILE HANDLING IN C++

## 13. TEMPLATE PROGRAMMING

## 14. EXCEPTION HANDLING IN C++



# 14.
# EXCEPTION HANDLING IN C++

# INTRODUCTION

- We know that it is very rare that program works correctly first time. It might have bugs (errors).
- Errors can be broadly categorized into two types. We will discuss them one by one.
  - Compile Time Errors
  - Run Time Errors
- **Compile Time Errors - Errors caught during compiled time is called Compile time errors**. Compile time errors include library reference, syntax error or incorrect class import.
- **Run Time Errors - They are also known as exceptions**. An exception caught during run time creates serious issues.

# WHAT IS EXCEPTION ?

- An **exception is an event** (unexpected problem) , which **occurs during the execution of a program**, that **disrupts the normal flow** of the program's instructions.
- **Exceptions are run-time anomalies or abnormal conditions** that a program encounters during its execution
- The errors which hinder (makes difficult) normal execution of program or terminates the program.
- **Exception are of two kinds**
  - Synchronous Exception
  - Asynchronous Exception

- **Synchronous Exception**
  - The exceptions (error) which **occur** during the program execution **due to some fault in the input data.**
  - For example: Errors such as out of range, overflow, division by zero
- **Asynchronous Exception**
  - The exceptions (error) caused by events or faults beyond the control of the program.
  - For Example: Keyboard failures, hardware disk failures
- The proposed **exception handling mechanism in C++** is designed to handle **synchronous exception**.

# NEED OF THE EXCEPTION HANDLING

- E.g. Division by zero

runtimeexception.cpp

```cpp
#include<iostream>
using namespace std;

int main()
{
    int a,b,c;
    cout<<"Enter first no:";
    cin>>a;
    cout<<"Enter second no:";
    cin>>b;
    c=a/b;
    cout<<"Answer is: "<<c;
    return 0;
}
```

Compile Log | Debug | Find Results | Close

```
- Errors: 0
- Warnings: 0
- Output Filename: F:\PATKAR_BSCIT\F.Y.B.Sc.IT\OOP's\NIKITA_OOPs\Practical_Relat
- Output Size: 1.83244323730469 MiB
- Compilation Time: 1.39s
```

Compiled Successfully

Output:

Entering correct input values will give proper output

```
F:\PATKAR_BSCIT\F.Y.B.Sc.IT\OOP's\NIKITA_OOPs\Practical_Related_Data\Practicals\runtimeexception.exe
Enter first no:
4
Enter second no:
2
Answer is: 2
--------------------------------
Process exited after 3.493 seconds with return value 0
Press any key to continue . . .
```

This program compiles successfully but the program **fails at runtime**, leading to an **exception**(because of entering incorrect input values)

```
F:\PATKAR_BSCIT\F.Y.B.Sc.IT\OOP's\NIKITA_OOPs\Practical_Related_Data\Practicals\runtimeexception.exe
Enter first no:
4
Enter second no:
0

--------------------------------
Process exited after 10.43 seconds with return value 3221225620
Press any key to continue . . .
```

# EXCEPTION HANDLING ?

- **Exception handling** is the **process of handling errors** and **exceptions** in such a way that they **do not hinder normal execution of the system**
- **Exception Handling** in C++ is a **process** to **handle runtime errors.**
- Exception handling is performed so the **normal flow** of the application **can be maintained** even **after runtime errors.**
- The process of converting **system error messages** into **user friendly error message** is known as **Exception handling**.

- **Advantage**
- It **maintains the normal flow of the application**. In such case**, rest of the code is executed even after exception.**
- This is one of the powerful feature of C++ to handle run time error and maintain normal flow of C++ application

# EXCEPTION HANDLING MECHANISM

- The purpose of the **exception handling mechanism** is to provide means to **detect and report "exceptional circumstances"** so that **appropriate action** can be **taken**.
- The mechanism suggests a separate error handling code that performs the following tasks:

1. Find the problem (**Hit the exception**).
2. Inform that the error has Occurred (**Throw the exception**).
3. Receive the error information (**Catch the exception**).
4. Take corrective actions (**Handle the exception**).

- The error handling code basically consists of two segments, **one to detect errors** and **to throw exceptions**, and the **other to catch the exceptions** and **to take appropriate actions**

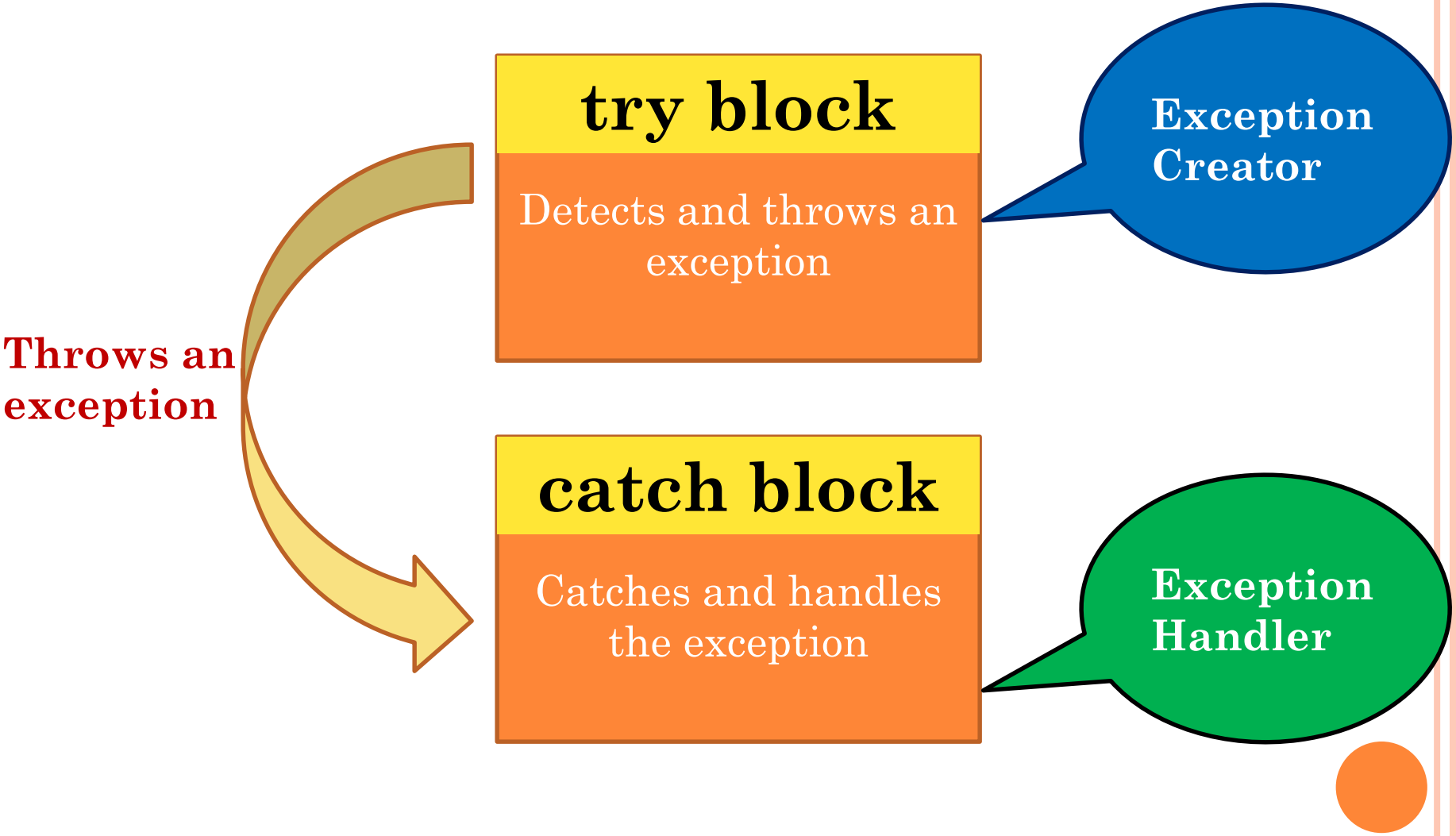
- C++ **exception handling** mechanism is basically **built** upon **three keywords** **try** ,**throw** and **catch**

- **try:**
  - The keyword **try** is to use to preface a block of statements (surrounded by braces { }) which may **generate exceptions**.
  - try block consists of the code that may generate an exception.
  - If an **exception occur** are **thrown from inside** the **try block**.
  - It's followed by one or more catch blocks

- **throw:**
  - **throw** keyword is used to **throw an exception encountered inside try block**.
  - After the **exception** is **thrown**, the **control** is **transferred** to **catch block**.

- **catch:**
  - **catch block catches** the **exception thrown by throw statement from try block**.
  - Then, **exception** are **handled** inside **catch block**.

# Exception Handling Mechanism



- Syntax

```
try
{
.........
throw exception;
.........
}
catch(type arg)
{
...........
..........
}
```

// block of statements that detects and throws an exception

//block of statements that handles the exception

e.g.

withexception.cpp

```cpp
#include<iostream>
using namespace std;
int main()
{
    int a, b,x;
    cout<<"Enter first number :";
    cin >> a;
    cout<<"Enter second number :";
    cin >> b;
        try
        {
            if(b!= 0)
            {
              int c=a/b;
              cout<<"Answer is:  "<<c;
            }
            else // There is an exceptions
            {
            throw(b);
            }
        }
        catch(int x)
        {
            cout<<"\nException : Divide by zero not allowed"<<"\n";
        }
    return 0;
}
```

## After entering correct input values

1st Run

F:\PATKAR_BSCIT\F.Y.B.Sc.IT\OOP's\NIKITA_OOPs\Practical_Related_Data\Practicals\withexception.exe

```
Enter first number :
4
Enter second number :
2
Answer is:  2
--------------------------------
Process exited after 6.186 seconds with return value 0
Press any key to continue . . .
```

## After entering incorrect input values [one input value is 0 (zero)]

2nd Run

F:\PATKAR_BSCIT\F.Y.B.Sc.IT\OOP's\NIKITA_OOPs\Practical_Related_Data\Practicals\withexception.exe

```
Enter first number :
4
Enter second number :
0

Exception : Divide by zero not allowed

--------------------------------
Process exited after 4.162 seconds with return value 0
Press any key to continue . . .
```

# THROWING MECHANISM

- When an **exception** that is desired to be **handled is detected**, it is **thrown** using the **throw statement** in one of the following forms
- **Syntax:**
  - throw (exception);
  - throw exception;
  - throw;  //used to rethrowing an exception
- The operand object exception may be of any type.
- When an **exception** is **thrown**, it will be **caught by** the **catch statement** associated with the try block. That is the control exits the current try block, and is transferred to the catch block after that try block.

# CATCHING MECHANISM

- **Code** for **handling exceptions** is included in **catch blocks**
- A **catch block** **looks like** a **function definition** and is of the form
- **Syntax:**

```
catch (type argument)
{
    //statements for managing exception (action to be taken)
}
```

- The ***type*** indicates the **type of exception** that **catch** block **handles**. The parameter arg(argument) is an optional parameter name.

- The exception handling code is **placed** between <u>two braces { }</u>
- The catch statement catches an exception whose type matches with the type of catch argument. When it is caught, the code in the catch block is executed

# MULTIPLE CATCH STATEMENTS

- It is possible that a program segment has **more than one condition to throw an exception**.
- in **such cases**, we can associate **more than one catch statement** with a **try**
- When an **exception is thrown**, the **exception handler** is **search** in order for an **appropriate match.**
- The first **handler** that **yields a match** is **executed**.

- Syntax:

```
try
{
    //try block
}
catch(type1 arg)
{
    //catch block1
}
catch(type2 arg)
{
    //catch block2
}
..........
..........
catch(typeN arg)
{
    //catch blockN
}
```

e.g.

multicath.cpp

```
1   #include<iostream>
2   using namespace std;
3   int main()
4   {   int k,s;
5       char p;
6       cout<<"****Exception Handling Using Multiple Catches****";
7       cout<<"\n\nEnter a value : ";
8       cin>>k;
9           try
10          {
11              if (k==0)
12              throw s;
13
14              else if (k<0)
15              throw p;
16
17              else if (k>0)
18              cout<<"Number is "<<k;
19          }
20          catch(int g)
21          {
22              cout<<"Value cannot be zero \n";
23          }
24          catch (char j)
25          {
26              cout<<"Value cannot be negative \n";
27          }
28      cout<<"\n\n*** end of program ***";
29      return 0;
30  }
```

1st Run

```
F:\PATKAR_BSCIT\F.Y.B.Sc.IT\OOP's\NIKITA_OOPs\Practical_Related_Data\Practicals\multicath.exe
****Exception Handling Using Multiple Catches****

Enter a value : 0
Value cannot be zero


*** end of program ***
--------------------------------
Process exited after 4.066 seconds with return value 0
Press any key to continue . . .
```

2nd Run

```
F:\PATKAR_BSCIT\F.Y.B.Sc.IT\OOP's\NIKITA_OOPs\Practical_Related_Data\Practicals\multicath.exe
****Exception Handling Using Multiple Catches****

Enter a value : -5
Value cannot be negative


*** end of program ***
--------------------------------
Process exited after 3.694 seconds with return value 0
Press any key to continue . . .
```

3rd Run

```
F:\PATKAR_BSCIT\F.Y.B.Sc.IT\OOP's\NIKITA_OOPs\Practical_Related_Data\Practicals\multicath.exe
****Exception Handling Using Multiple Catches****

Enter a value : 2
Number is 2

*** end of program ***
--------------------------------
Process exited after 3.163 seconds with return value 0
Press any key to continue . . .
```

# CATCH ALL EXCEPTION

- There might be situation where all the possible types of exceptions might not be handled.
- Here with this , the code is written in such a way that a **catch statement** may be forced **catch** **all exception** **instead** of **particular type** alone.
- This is useful, If an exception handler to catch all exceptions, instead of just a certain type.
- Syntax:

```
catch(...)
{
//statement for processing all exceptions
}
```

e.g.

catchall.cpp

```cpp
#include<iostream>
using namespace std;
int main()
{   int k,s;
    char p;
    cout<<"****Catches All Exception Example****";
    cout<<"\n\nEnter a value";
    cin>>k;
        try
        {
            if (k==0)
            throw s;

            else if (k<0)
            throw p;

            else if (k>0)
            cout<<"Number is "<<k;
        }
        catch(...)
        {
            cout<<"Exception Ocurred";
        }
    return 0;
}
```

1st Run

F:\PATKAR_BSCIT\F.Y.B.Sc.IT\OOP's\NIKITA_OOPs\Practical_Related_Data\Practicals\catchall.exe

```
****Catches All Exception Example****

Enter a value
0
Exception Ocurred
--------------------------------
Process exited after 2.559 seconds with return value 0
Press any key to continue . . .
```

2nd Run

F:\PATKAR_BSCIT\F.Y.B.Sc.IT\OOP's\NIKITA_OOPs\Practical_Related_Data\Practicals\catchall.exe

```
****Catches All Exception Example****

Enter a value
-4
Exception Ocurred
--------------------------------
Process exited after 2.66 seconds with return value 0
Press any key to continue . . .
```

3rd Run

F:\PATKAR_BSCIT\F.Y.B.Sc.IT\OOP's\NIKITA_OOPs\Practical_Related_Data\Practicals\catchall.exe

```
****Catches All Exception Example****

Enter a value
5
Number is 5
--------------------------------
Process exited after 3.16 seconds with return value 0
Press any key to continue . . .
```

# EXCEPTION HANDLING USING FUNCTION

- We can specify the types of exception for a function to be thrown as a list passed to throw

- E.g.

functionexception.cpp

```cpp
#include<iostream>
using namespace std;
void test(int x)
{
    try
    {
        if (x==0)
        throw x;
        else if (x<0)
        throw 3.4;
        else if (x>100)
        throw 's';
    }
    catch (int i)
    {
        cout<<"Age cannot be zero";
    }
    catch (double s)
    {
        cout<<"Age cannot be in negative";
    }
    catch (char c)
    {
        cout<<"Age cannot be greater than 100";
    }
}
int main()
{
    cout<<"*****Exception Handling Using Function****";
    int a;
    cout<<"\n\nEnte Your Age : ";
    cin>>a;
    test(a);
    cout<<"Your Entered Age is  : "<<a;
    return 0;
}
```

1st Run

F:\PATKAR_BSCIT\F.Y.B.Sc.IT\OOP's\NIKITA_OOPs\Practical_Related_Data\Practicals\functionexception.exe

```
*****Exception Handling Using Function****

Ente Your Age : 7

Your Entered Age is  : 7
--------------------------------
Process exited after 2.134 seconds with return value 0
Press any key to continue . . .
```

2nd Run

Select F:\PATKAR_BSCIT\F.Y.B.Sc.IT\OOP's\NIKITA_OOPs\Practical_Related_Data\Practicals\functionexcepti...

```
*****Exception Handling Using Function****

Ente Your Age : 0

Age cannot be zero
Your Entered Age is  : 0
--------------------------------
Process exited after 2.001 seconds with return value 0
Press any key to continue . . .
```

3rd Run

F:\PATKAR_BSCIT\F.Y.B.Sc.IT\OOP's\NIKITA_OOPs\Practical_Related_Data\Practicals\functionexception.e...

```
*****Exception Handling Using Function****

Ente Your Age : 121

Age cannot be greater than 100
Your Entered Age is  : 121
--------------------------------
Process exited after 3.074 seconds with return value 0

Press any key to continue . . .
```

# EXCEPTION HANDLING WITH CLASS

- In addition to built in data types, exception of user-defined data types such as classes can also be caught and handled.
- In this case, the throw expression throws an object  of the class as exception and the catch block accepts an objects of the class as argument.

- E.g.

classexception.cpp

```
1   #include<iostream>
2   using namespace std;
3   class even
4   {
5       int n;
6       public:
7       void getdata(int x)
8       {
9           n=x;
10          if(n%2!=0)
11          throw even();
12          else
13          cout<<"Divided By 2";
14      }
15  };
16  int main()
17  {
18      int a;
19      even e;
20      cout<<"----- Exception Handling Using CLass -----";
21      cout<<"\n\nEnter a Number :";
22      cin>>a;
23      try
24      {
25          e.getdata(a);
26      }
27      catch(even e)
28      {
29          cout<<"\nException Caught !!"<<endl;
30          cout<<a<<" Not Divided by 2";
31      }
32      return 0; }
```

1st Run

C:\Users\Admin\Desktop\C++Practical\classexception.exe

```
----- Exception Handling Using CLass -----

Enter a Number :12
Divided By 2
--------------------------------
Process exited after 2.864 seconds with return value 0
Press any key to continue . . .
```

2nd Run

C:\Users\Admin\Desktop\C++Practical\classexception.exe

```
----- Exception Handling Using CLass -----

Enter a Number :55

Exception Caught !!
55 Not Divided by 2
--------------------------------
Process exited after 5.007 seconds with return value 0
Press any key to continue . . .
```

# RETHROWING EXCEPTION

- An **exception** is **thrown** **<u>from</u>** the **catch block** is **known** as the **rethrowing exception**
- It can be simply invoked by **throw without argument i.e. using →** **throw;**
- This causes the current **exception** to be passed on to an **outer try/catch sequence**
- **Rethrown exception will be caught by newly defined catch block**
- When you **rethrow** an **exception**, it **will not** be ***recaught*** by the **same catch** statement. It will **propagate** to the **immediately enclosing try/catch sequence**. (The same catch statement cannot catch the rethrown exception. The next catch statement will catch it)

e.g.

rethrowingexception.cpp

```cpp
#include<iostream>
using namespace std;
void div(int x,int y)
{
    cout <<"Inside function \n";
        try
        {
            if(y == 0)
            throw y;
            else
            cout << "Division =" << x/y << "\n";
        }
        catch(int)
        {
            cout <<"Caught int exception inside function \n";
            throw;
        }
    cout <<"End of function \n\n";
}
int main()
{
    cout <<"Inside main \n";

    int x,y;
    cout<<"Enter 1st Number : ";
    cin>>x;
    cout<<"Enter 2nd Number : ";
    cin>>y;
    try
    {
        div(x,y);
    }
    catch(int)
    {
        cout <<"Caught int exception inside main \n";
    }
cout<<"End of main";
return 0;
}
```

1st Run

F:\PATKAR_BSCIT\F.Y.B.Sc.IT\OOP's\NIKITA_OOPs\Practical_Related_Data\Practicals\rethrowingexception....

```
Inside main
Enter 1st Number : 4
Enter 2nd Number : 2
Inside function
Division =2
End of function

End of main
--------------------------------
Process exited after 3.679 seconds with return value 0
Press any key to continue . . .
```

2nd Run

F:\PATKAR_BSCIT\F.Y.B.Sc.IT\OOP's\NIKITA_OOPs\Practical_Related_Data\Practicals\rethrowingexception.e...

```
Inside main
Enter 1st Number : 4
Enter 2nd Number : 0
Inside function
Caught int exception inside function
Caught int exception inside main
End of main
--------------------------------
Process exited after 5.156 seconds with return value 0
Press any key to continue . . .
```

- **Example : Program to accept password and throw an exception if the password has less than 5 (or 5) characters or contains a digit. Give another chance to enter a correct password and rethrow an exception if the password is incorrect**

passwordexception.cpp

```cpp
#include<iostream>
using namespace std;
void check(char x[20])
{
    char u;
    int i;
    try
    {
        for(i=0; x[i]!='\0'; i++)
        {
            if(x[i]>='0' && x[i]<= '9')
            throw u;
        }
        if(i<=5)
        throw u;
        else
        cout<<"Your Password Is :"<<x;
    }
    catch(char d)
    {
        cout<<"Entered Password Is Not Proper";
        cout<<"\nEnter Password Again : ";
        cin>>x;
            for(i=0;x[i]!='\0';i++)
        {
            if(x[i]>='0' && x[i]<= '9')
            throw;
        }
        if(i<=5)
        throw;
        else
        cout<<"Your Password Is :"<<x;
    }
}
int main()
{
    char x[20];
    int i;
    cout<<"Enter Password :";
    cin>>x;
    try
    {
        check(x);
    }
    catch(char r)
    {
        cout<<"Incorrect Password ...Sorry";
    }
    return 0;
}
```

1st Run

F:\PATKAR_BSCIT\F.Y.B.Sc.IT\OOP's\NIKITA_OOPs\Practical_Related_Data\Practicals\passwordexception.exe

```
Enter Password : nikita
Your Password Is : nikita
--------------------------------
Process exited after 4.116 seconds with return value 0
Press any key to continue . . .
```

2nd Run

F:\PATKAR_BSCIT\F.Y.B.Sc.IT\OOP's\NIKITA_OOPs\Practical_Related_Data\Practicals\passwordexception.e...

```
Enter Password : nik
Entered Password Is Not Proper
Enter Password Again : niki

Incorrect Password ...Sorry
--------------------------------
Process exited after 5.732 seconds with return value 0
Press any key to continue . . .
```

3rd Run

F:\PATKAR_BSCIT\F.Y.B.Sc.IT\OOP's\NIKITA_OOPs\Practical_Related_Data\Practicals\passwordexception.exe

```
Enter Password : nik
Entered Password Is Not Proper
Enter Password Again : nikitamadwal
Your Password Is : nikitamadwal
--------------------------------
Process exited after 38.53 seconds with return value 0
Press any key to continue . . .
```

4th Run

F:\PATKAR_BSCIT\F.Y.B.Sc.IT\OOP's\NIKITA_OOPs\Practical_Related_Data\Practicals\passwordexception.exe

```
Enter Password : niki3
Entered Password Is Not Proper
Enter Password Again : nikit3

Incorrect Password ...Sorry
--------------------------------
Process exited after 15.46 seconds with return value 0
Press any key to continue . . .
```

5th Run

F:\PATKAR_BSCIT\F.Y.B.Sc.IT\OOP's\NIKITA_OOPs\Practical_Related_Data\Practicals\passwordexception.exe

```
Enter Password : niki3
Entered Password Is Not Proper
Enter Password Again : nikitapatkar
Your Password Is : nikitapatkar
--------------------------------
Process exited after 33.19 seconds with return value 0
Press any key to continue . . .
```

# 13.
# TEMPLATE PROGRAMMING

# TEMPLATE

- **Templates** in C++ is an powerful feature that is used for **generic programming**.
- **Generic Programming**(which involves writing code in a way that is independent of any data type) is an **approach** of programming where **generic types** are **used** as **parameters in algorithms** to work for a **variety of data types** (It enables the programmer to write a general algorithm which will work with all data types)
- **Templates** are mostly **implemented** for **crafting** a family of **classes** or **functions** having **similar features**

- **Templates** in C++ is defined as a **blueprint or formula** for **creating** a **generic class** or a **function**. (By using template a **single function** or **single class** can be created to work with **different** data types)

- **Need of template**:
- It is **used** to **pass** the **data type** as a **parameter** so that ***don't need*** to **write same code** for ***different data types***.
- C++ templates are a powerful mechanism for **code reuse**, as they **enable** the **programmer** to **write code** that **behaves** the ***same*** for **data of any type**
- **Syntax :**

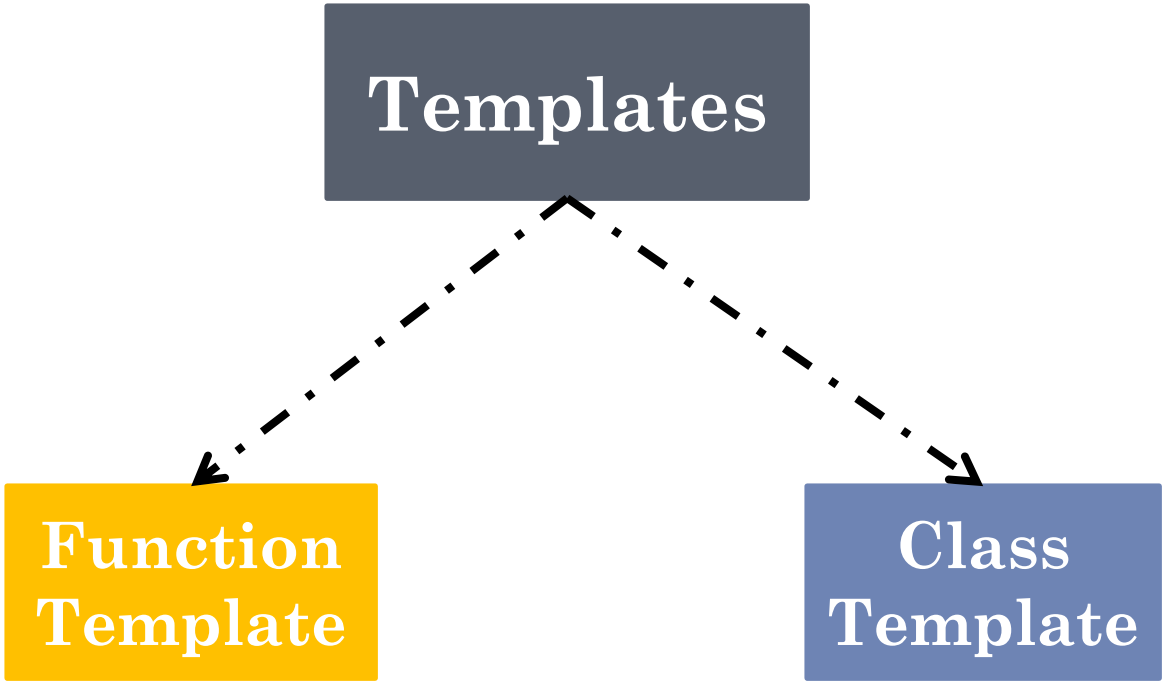
template <class type>

**e.g.**

template<class T>

- The **template** keyword tells the compiler that what follows is a template, **class** specifies generic type in a template and **T** is a template parameter that identifies a type.

# TYPES OF TEMPLATE



# FUNCTION TEMPLATE

- **Function Template** is just **like** a *normal function*, **but** the only **difference is** while *normal function* can *work only* on *one data type* and a **function template** code can **work on multiple data types**. (The function template can work with different data types at once)
- **Function templates** are **special functions** that can **operate** with *generic types*. This **allows** to create a **function template** whose **functionality can** be **adapted** to **more than** ***one type or class*** **without repeating** the **entire code** for **each type**.

- A single **function template** can handle multiple data types at the same time, while a single **normal function** can only handle one set of data types
- When templates used for functions they are called as **function templates**
- **Syntax :**

```
template < class type>
return_type function_name(parameter_ist)
    {
        // body of function.
    }
```

- Where **,**
- **type**: It is a **placeholder name** for a **data type** used by the function. It is used within the function definition. It is only a **placeholder** that the **compiler** will **automatically replace** this **placeholder with** the **actual data type**.
- **class**: A class keyword is used to specify a generic type in a template declaration.

- E.g.

Without Using Function Template

withoutfunctiontempalte.cpp

```cpp
#include <iostream>
using namespace std;
int add(int x,int y)
{
    return x+y;
}
double add(double v,double e)
{
    return v+e;
}

int main()
{
    cout<<"********Without Using Function Template ********";
    int a=20,b=30;
    cout<<"\n\nAddition of integer is : "<<add(a,b)<<endl;

    double l=2.3,m=2.3;
    cout<<"Addition of double is : "<<add(l,m)<<endl;
    return 0;
}
```

- Output:

```
F:\PATKAR_BSCIT\F.Y.B.Sc.IT\OOP's\NIKITA_OOPs\Practical_Related_Data\Practicals\withouttempalte.exe
********Without Using Function Template ********

Addition of integer is : 50
Addition of double is : 4.6

--------------------------------
Process exited after 0.3128 seconds with return value 0
```

# Function Template With Single Parameter

- E.g.

**1. Single Parameter (Same Data Type)**

**Using Function Template**

usingfunctiontemplate.cpp

```cpp
#include <iostream>
using namespace std;
template<class N>  //template : Same data type (single Parameter)
N add(N x,N y)
{
    return x+y;
}

int main()
{
    cout<<"******** Function Template [Single Parameter] ********";
    int a=20,b=30;
    cout<<"\n\nAddition of integer is : "<<add(a,b)<<endl;

    double l=2.3,m=2.3;
    cout<<"Addition of double is : "<<add(l,m)<<endl;
    return 0;
}
```

- Output:

```
F:\PATKAR_BSCIT\F.Y.B.Sc.IT\OOP's\NIKITA_OOPs\Practical_Related_Data\Practicals\usingfunctiontemplate.exe
******** Function Template [Single Parameter] ********

Addition of integer is : 50
Addition of double is : 4.6

--------------------------------
Process exited after 0.2141 seconds with return value 0
Press any key to continue . . .
```

# Function Template With Multiple Parameters

**2. Multiple Parameter (Different Data Type)**

**Using Function Template**

usinfunctiontemplate2.cpp

```cpp
#include <iostream>
using namespace std;
template<class N,class T>  //template : differnt data types(Multiple paameter)
double add(N x,T y)
{
    return x+y;
}

int main()
{
    cout<<"******** Function Template [Multiple Parameter] ********";

    cout<<"\n\nAddition of integer is : "<<add(10,20)<<endl;
    cout<<"\nAddition of double is : "<<add(10.2,10.3)<<endl;
    cout<<"\nAddition of integer and double is : "<<add(10,30.2)<<endl;
    return 0;
}
```

- Output:

```
F:\PATKAR_BSCIT\F.Y.B.Sc.IT\OOP's\NIKITA_OOPs\Practical_Related_Data\Practicals\usinfunctiontemplate2.exe
******** Function Template [Multiple Parameter] ********

Addition of integer is : 30

Addition of double is : 20.5

Addition of integer and double is : 40.2

--------------------------------
Process exited after 0.2259 seconds with return value 0
Press any key to continue . . .
```

# CLASS TEMPLATE

- Normally there is a need to create a different class for each data type OR create different member variable and function within a single class so for that the **class template** can be used
- **Similar** to **function templates**, the **class templates** is **used to create** a **single class** to **work with different data types**
- **Class Templates** Like function templates, class templates are **useful when** a **class defines something** that is **independent of the data type**
- **Using class templates**, the **code can** be **reuse for *all* data types**
- When a **class uses** the **concept** of **Template**, then the **class** is **known** as **generic class**

- **Syntax**
- **Class Template**

```
template <class T>
class class_name
{
    ..........  //class data member and function
};
```

Here, **T** is a **placeholder** for the data type used

- **Create a class template object**

While creating a **class template object** then **have to define** the ***data type*** **<u>inside</u>** **< >** during creation

```
class_name <data_type> class_object;
```

# Class Template With Single Parameter

1. Single Parameter (Same Data Type)

Using Class Template

classtemplate.cpp

```cpp
#include<iostream>
using namespace std;
template<class T>  //class template : single Parameter
class xyz
{
  T a,b;
  public:
  void getdata()
  {
    cout<<"\n\nEnter Two Numbers :"<<endl;
    cin>>a>>b;
  }
  void sum()
  {
    cout<<"Addition : "<<a+b<<endl;
  }
};
int main()
{
  cout<<"******** Class Template (Single Parameter) ********";
  xyz <int> s1 ;      //xyz is a class and s1 is a object of class xyz
  xyz <float> s2 ;    //xyz is a class and s2 is a object of class xyz
  s1.getdata();
  s1.sum();
  s2.getdata();
  s2.sum();
  return 0;
}
```

- Output:

```
F:\PATKAR_BSCIT\F.Y.B.Sc.IT\OOP's\NIKITA_OOPs\Practical_Related_Data\Practicals\classtemplate.exe

******** Class Template (Single Parameter) ********

Enter Two Numbers :
2
3
Addition : 5


Enter Two Numbers :
3.4
5.4
Addition : 8.8

--------------------------------
Process exited after 16.66 seconds with return value 0
Press any key to continue . . .
```

# Class Template With Multiple Parameters

- E.g.

**2. Multiple Parameters (Different Data Type)**

**Using Class Template**

classtemplatemultipledatatype.cpp

```cpp
#include<iostream>
using namespace std;
template<class T1,class T2>  //Class Template : Multiple Parameter
class xyz
{
  T1 a;
  T2 b;
  public:
  xyz(T1 x,T2 y)
  {
    a=x;
    b=y;
  }
  void display()
  {
    cout<<"\na = "<<a <<"  b = "<<b<<endl;
  }
};
int main()
{
  cout<<"******** Class Template (Multiple Parameter) ********\n";
  xyz <int,float> s1(10,2.3);
  xyz <float,float> s2(3.4,4.5);
  xyz <char,int> s3('N',5);
  s1.display();
  s2.display();
  s3.display();
  return 0;
}
```

- Output:

```
F:\PATKAR_BSCIT\F.Y.B.Sc.IT\OOP's\NIKITA_OOPs\Practical_Related_Data\Practicals\classtemplatemultipledatatype.exe
******** Class Template (Multiple Parameter) ********

a = 10  b = 2.3

a = 3.4  b = 4.5

a = N  b = 5

--------------------------------
Process exited after 0.1389 seconds with return value 0
Press any key to continue . . .
```

# 12.
# FILE HANDLING IN C++

# INTRODUCTION

- In programming, we may require **some specific input data to be generated several numbers** of times.
- The data to be displayed may be very large, and only a limited amount of data can be displayed on the console, and When a program is terminated, the entire data is lost. it is impossible to recover the programmatically generated data again and again.
- It is therefore necessary to have most flexible approach where data can be stored on the disk and read whenever necessary, without destroying the data.

- However, if we need to do so, we may store it onto the local file system which can be accessed every time.
- A file is a collection of related data stored on a particular area on the disk
- File helps in storing the information permanently so the information entered by the user into the file can be retrieve or use for further use.
- Programs can be designed to perform the read and write operations on these files
- So here will discuss about various methods for storing and retrieving the data from files (i.e. how files are handled using C++ program and what are the functions and syntax used to handle files in C++)

- The **I/O system of C++ handles file operations <u>similar to</u>** the **<u>console input and output operation</u>**
- It **uses <u>file streams</u>** as an **interface between** the **<u>programs and the files</u>**
- The stream that **supplies data to the program** is **known** as **input stream** and the one that **receives data from the program** is **known** as **output stream** (*i.e. The **input stream extracts** (or <u>**reads**</u>) **data from** the **file** and the **output stream inserts** (or <u>**writes**</u>) **data to** the **file***)
- The C++ standard library provides **fstream class for** performing **Read** and **Write** operations
- **Operations can be performed on a file**.
  - Naming the file
  - Opening a file
  - Reading data from the file
  - Writing to the file
  - Closing the file

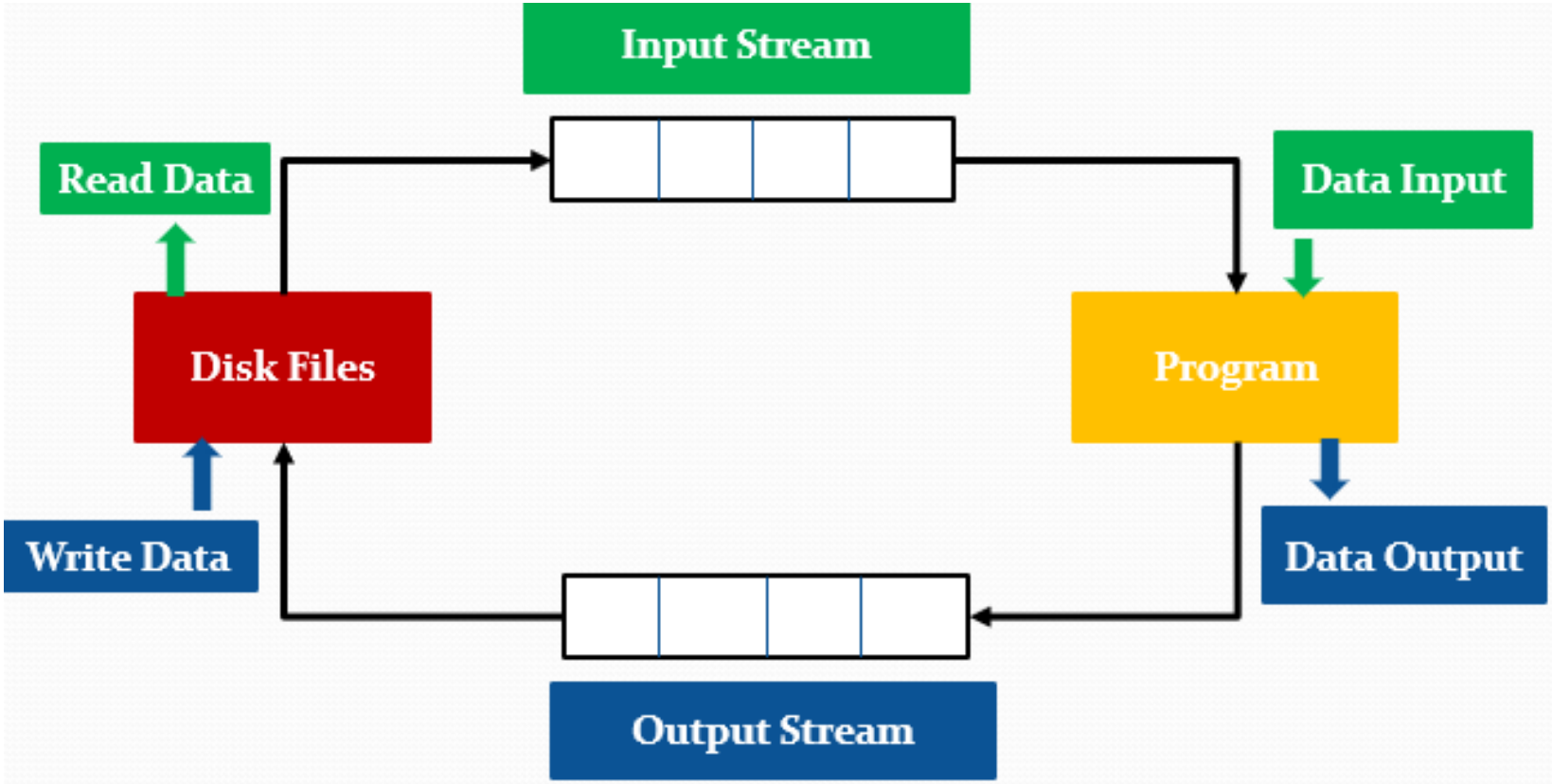



Fig. File Input and Output Stream

# CLASSES FILE STREAM OPERATIONS

- The I/O system of C++ contains a following set of classes that define the file handling methods

| Data Type (Class) | Description |
|---|---|
| **ofstream** | This represents the **output file stream** and is used to create files and to **write information to files (Stream class to write on files** ) |
| **ifstream** | This represents the **input file stream** and is used to **read information from files** (**Stream class to read from files**) |
| **fstream** | This represents the **file stream generally**, and has the capabilities of **both ofstream and ifstream** which means it **can create files**, **write information to files, and read information from files (Stream class to both read and write from/to files.**) |

# OPENING A FILE

- A file must be **opened before** **<u>read from it or write to it</u>**
- Either the **ofstream or fstream object** may be used to **open a file for writing** and ***ifstream object*** is used to ***open a file for reading*** purpose only.
- **A file can be opened in two ways:**
  - Using constructor
  - Using member function open()

# CLOSING A FILE

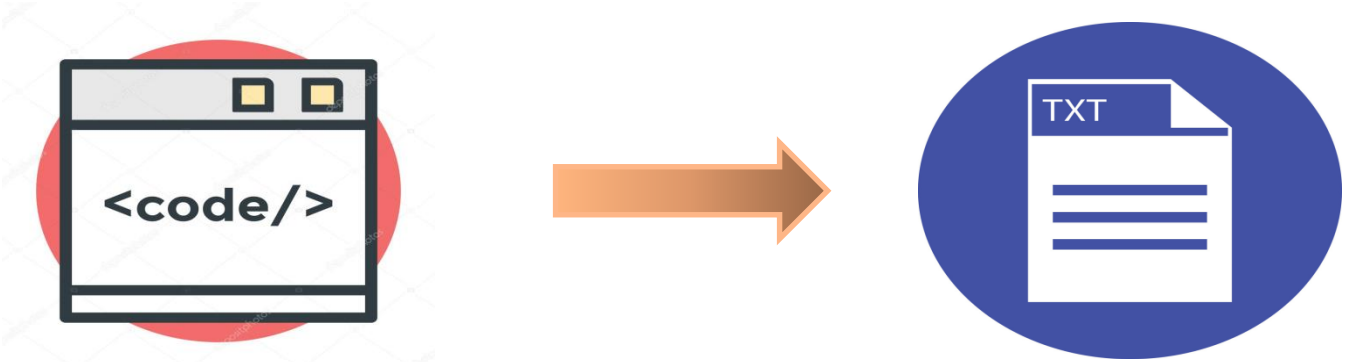
- After **finished** with **input and output operation** on a file ,then **it has to close** so that its resources become available again
- To **close a file** , member **function close()** of stream class is used
- This member function **does not take any parameter**
- e.g.

st.close();

file.close();

# OFSTREAM CLASS

- This **represents** the **output file stream** and is used to **create files and to write information** ***<u>to files</u>*** (*Stream class to* ***write on files*** )
- To **write information to a file <u>from</u>** the **program** using the **stream insertion operator (<<)** just as it is **use** that operator to **output information to the screen**.
- The only difference is that for file handling it uses an **ofstream or fstream object <u>instead</u>** of the **cout** object.



## Use of [ofstream] : open and write into the file (program to file)

- Opening a file using constructor
- It involves **two steps**:

1. Create a file stream object.

- **ofstream** is used for **output stream**

2. Initialize the file object with the desired file name.

fileconstructor.cpp

```
1   #include<iostream>
2   #include<fstream>     //header file for File Handling
3   using namespace std;
4   int main()
5   {
6   ofstream st("one.txt");    //st is the object of ofstream class
7   st<<"File Opening Using Constructor";
8   st.close();
9   return 0;
10  }
```

- Output:

F:\PATKAR_BSCIT\F.Y.B.Sc.IT\OOP's\NIKITA_OOPs\Practical_Related_Data\Practicals\fileconstructor.exe

```
--------------------------------
Process exited after 0.1547 seconds with return value 0
Press any key to continue . . .
```

| | | | |
|---|---|---|---|
| one | 3/7/2023 10:45 PM | Text Document | 1 KB |

one - Notepad

File Edit Format View Help

File Opening Using Constructor

Ln 1, Col 31 | 100% | Windows (CRLF) | UTF-8

**It opens the file "one.txt" , if it exists ; Otherwise it creates a new file named "one.txt" and then writes "File Opening Using Constructor" in the file**

- **Opening a file** using **open() function**
- **Open function** has taken **one argument** i.e. **filename** “one_a.txt”

fileopensuingopen.cpp

```
1   #include <iostream>
2   #include <fstream>
3   using namespace std;
4   int main ()
5   {
6     ofstream file;
7     file.open ("one_a.txt");
8     file << "File Openning Usig open() Function ......";
9     file.close();
10    return 0;
11  }
```

- Output:

F:\PATKAR_BSCIT\F.Y.B.Sc.IT\OOP's\NIKITA_OOPs\Practical_Related_Data\Practicals\fileopensuingopen.exe

```
--------------------------------
Process exited after 0.3279 seconds with return value 0
Press any key to continue . . .
```

| one_a | 3/7/2023 10:51 PM | Text Document | 1 KB |
|---|---|---|---|

one_a - Notepad

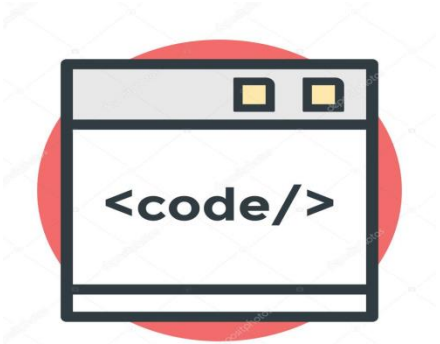
File Edit Format View Help

```
File Openning Usig open() Function ......
```

**It opens the file "one_a.txt" , if it exists ; Otherwise it creates a new file named "one_a.txt" and then writes "File Opening Using open() Function ....." in the file**

# IFSTREAM CLASS

- This **represents the input file stream** and is **used to read information *from files*** (*Stream class to read from files*)
- To **read information from a file <u>into</u> the program** using the **stream extraction operator (>>)** just as it is **use** that operator **to input information from** the **keyboard**.
- The only difference is that for file handling it uses an **ifstream or fstream object <u>instead</u>** of the **cin** object.




## Use of [ifstream] : open and read from the file (file to program)

ifstreamfile.cpp

```cpp
#include <iostream>
#include <fstream>
using namespace std;
int main ()
{
  ifstream rfile;  // rfile is the object of ifstream class
  char name[50];
  rfile.open ("two.txt");
  rfile.getline(name,50);    //it will take the input from file with space
  cout<<"Name = "<<name;
  rfile.close();
  return 0;
}
```

**NOTE :** if **rfile>>name** is used instead of **rfile.getline()** it will not take the value after space (i.e. The data without space )

- Output:

| | | | |
|---|---|---|---|
| two | 3/7/2023 11:06 PM | Text Document | 1 KB |

two - Notepad

File Edit Format View Help

```
Nikita Madwal
```

Select F:\PATKAR_BSCIT\F.Y.B.Sc.IT\OOP's\NIKITA_OOPs\Practical_Related_Data\Practicals\ifstreamfile.exe

```
Name = Nikita Madwal
--------------------------------
Process exited after 0.1766 seconds with return value 0
Press any key to continue . . .
```

1. First text file must have some data
2. Then After writing the code in editor (Dev C++) Compile and Run.
3. The data which has been added to the text file will print to the C++ console.

# FILE OPENING MODES

- Opening file also involves several modes depending upon operation to be carried out with the file.
- Syntax of open() function:

  object. open("file_name",mode);
- It is possible to use **more than one modes** at the **same time**.
- **This modes will combine using the ( | ) [OR] operator**

| Mode | Description |
| --- | --- |
| ios::in | It opens the file for a read |
| ios::out | It opens the file for a write |
| ios:: app | The Append mode. The output sent to the file is appended to it (**Append data at the end of the file**) |
| ios::ate | It opens the file for the output then moves the read and write control to file's end (**Allow data to be added or modified anywhere within the file**) |
| ios::nocreate | Open fails if file does not exists **(If file exists then it will open that file but if file doesn't exit it will not create new file**) |
| ios::noreplace | **Open fails if tried to open already existing file in writing mode** |
| ios::trunc | If a file exists, the file elements should be truncated prior to its opening (**Deleted or truncated data and used to write new data i.e. removes the data in the existing file** ) |

# e.g. ios::out

iosout.cpp

```cpp
#include<iostream>
#include<fstream>
using namespace std;
int main ()
{
  fstream file;
  file.open ("abc.txt",ios::out);  //writing to the file
  file << "File Openning Using ios::out ......";
  file.close();
  return 0;
}
```

- Output:

F:\PATKAR_BSCIT\F.Y.B.Sc.IT\OOP's\NIKITA_OOPs\Practical_Related_Data\Practicals\iosout.exe

```
--------------------------------
Process exited after 0.2212 seconds with return value 0
Press any key to continue . . .
```

| | | | |
|---|---|---|---|
| abc | 3/7/2023 11:29 PM | Text Document | 1 KB |

abc - Notepad

File Edit Format View Help

```
File Openning Using ios::out ......
```

Ln 1, Col 36 | 100% | Windows (CRLF) | UTF-8

# e.g. ios::in

iosin.cpp

```cpp
#include <iostream>
#include <fstream>
using namespace std;
int main ()
{
  fstream file3;  //object of fstream class
  char name[50];
  file3.open ("two_t.txt",ios::in);
  file3.getline(name,50);    //it will take the input from file with space
  cout<<"College Name = "<<name;
  file3.close();
  return 0;
}
```

- Output:

| | | | |
|---|---|---|---|
| two_t | 3/7/2023 11:20 PM | Text Document | 1 KB |

two_t - Notepad

File Edit Format View Help

```
patkar-varde college
```

Ln 1, Col 1 | 100% | Windows (CRLF) | UTF-8

C:\Users\Admin\Desktop\backup 29.7.22\f\PATKAR_BSCIT\Education_2022-23\F.Y.B.Sc.IT\2021-2022\...

```
College Name = patkar-varde college
--------------------------------
Process exited after 0.02196 seconds with return value 0
Press any key to continue . . .
```

# e.g. ios::app

iosapp.cpp

```cpp
#include <iostream>
#include <fstream>
using namespace std;
int main ()
{
  char arr[100];
  cout<<"Enter Your Name "<<endl;
  cin.getline(arr,100);

  fstream file3;  //object of fstream class

  file3.open ("abc.txt",ios::out|ios::app);
  file3<<arr;
  file3.close();
  return 0;
}
```

- Output:

C:\Users\Admin\Desktop\backup 29.7.22\f\PATKAR_BSCIT\Education_2022-23\F.Y.B.Sc.I...

```
Enter Your Name
Nikita


--------------------------------
Process exited after 10.17 seconds with return value 0
Press any key to continue . . .
```

| | | | |
|---|---|---|---|
| abc | 3/7/2023 11:32 PM | Text Document | 1 KB |

abc - Notepad

File Edit Format View Help

```
File Openning Using ios::out ......Nikita
```

Ln 1, Col 1 | 100% | Windows (CRLF) | UTF-8

# CHECKING END OF FILE

- **eof()** function is uses with the stream object to **check for the file's end**
- Detection of the end-of-file condition is necessary for preventing any further attempt to read data from the file
- The member function eof(End-of-file) returns a boolean true if the file reaches the end of it and false if not. (It returns non-zero when the end of file has been reached, otherwise it returns zero)

- E.g.

eoffile.cpp

```cpp
#include<iostream>
#include<fstream>
using namespace std;
int main ()
{
  ifstream rfile;  //object of ifstream class
  char data[50];
  rfile.open ("two.txt");

  while(!rfile.eof())
  {
    rfile.getline(data,50);
    cout<<"\n"<<data;
  }
  rfile.close();
  return 0;
}
```

- Output:

two - Notepad

File Edit Format View Help

```
Patkar - Varde College
It is in Goregaon(West)
```

Ln 2, Col 24 | 100% | Windows (CRLF) | UTF-8

F:\PATKAR_BSCIT\F.Y.B.Sc.IT\OOP's\NIKITA_OOPs\Practical_Related_Data\Practicals\eoffile.exe

```
Patkar - Varde College
It is in Goregaon(West)


--------------------------------
Process exited after 0.128 seconds with return value 0
Press any key to continue . . .
```

# RANDOM ACCESS IN FILE

- Random access means **reading data randomly** from any where in the file.
- For this purpose we need to set the position of the file pointer first in the file and then read the data.
- A **file pointer points to** a **position in a file**, **which determined** by the **offset** (number of bytes) from the beginning or from the end.
- When the file is **opened in the read mode** or **write mode** the **file pointer** is ***positioned*** at the beginning of file.

- The **input pointer** which is used for **reading** the **file** **from** a ***particular location*** is called **get pointer**.
- The **output pointer** which is used for **writing** **in the** **file** is called **put pointer**

# THE FUNCTIONS USED TO ACCESS DATA RANDOMLY

| Function | Class | Description |
|---|---|---|
| seekp() | ofstream | It moves the file **put pointer** to specified location by the argument. **Used for writing** |
| tellp() | ofstream | It returns the current position of **put pointer** |
| seekg() | ifstream | It moves the file **get pointer** to specified location by the argument. **Used for reading** |
| tellg() | ifstream | It returns the current position of **get pointer** |

- **seekg** and **seekp** functions ***have* two arguments**.
- The **first** is an **integer number *which* <u>specifies</u> the offset**
- **Secondly specifies <u>reference</u>** position from where the offset is measured (i.e. **from file beginning , from the current position or from file end)**

| Code for reference | Description |
|---|---|
| ios::beg | From he beginning of file |
| ios::cur | From the current position of file |
| ios::end | From end of file |

- Syntax
- seekg(int offset,ref_pas);
- seekp(int offset,ref_pas);

# Use of tellp() and seekp()

# Use of tellp()

seekptellp.cpp

```cpp
#include<iostream>
#include<fstream>
using namespace std;
int main()
{
ofstream fil("sample.txt",ios::out);
cout<<fil.tellp()<<endl;            //pointer location is 0
fil<<"Hello World";
cout<<fil.tellp()<<endl;     //after writing data to file pointer loction is 11

/*fil.seekp(-5,ios::end);       //after moved pointer backward from end to the 5th location using seekp
cout<<fil.tellp()<<endl;      //now loction will be 6
fil<<"India";                 //now on 6th location changed the data i.e from world to india
fil.close();

ifstream r("sample.txt",ios::in);      // To see the changed which has been done in file for that read operation performed
char ch;
while(!r.eof())
{
    ch=r.get();
    cout<<ch;
}
r.close();*/
return 0;
}
```

- Output:

F:\PATKAR_BSCIT\F.Y.B.Sc.IT\OOP's\NIKITA_OOPs\Practical_Related_Data\Practicals\seekptellp.exe

```
0
11


--------------------------------
Process exited after 0.2582 seconds with return value 0
Press any key to continue . . .
```

sample - Notepad

File Edit Format View Help

Hello World

Ln 1, Col 1 | 100% | Windows (CRLF) | UTF-8

# Use of seekp()

seekptellp.cpp

```cpp
#include<iostream>
#include<fstream>
using namespace std;
int main()
{
ofstream fil("sample.txt",ios::out);
cout<<fil.tellp()<<endl;            //pointer location is 0
fil<<"Hello World";
cout<<fil.tellp()<<endl;     //after writing data to file pointer loction is 11

fil.seekp(-5,ios::end);       //after moved pointer backward from end to the 5th location using seekp
cout<<fil.tellp()<<endl;      //now loction will be 6
fil<<"India";                 //now on 6th location changed the data i.e from world to india
fil.close();

/*ifstream r("sample.txt",ios::in);      // To see the changed which has been done in file for that read operation performed
char ch;
while(!r.eof())
{
    ch=r.get();
    cout<<ch;
}
r.close();*/
return 0;
}
```

- Output:

F:\PATKAR_BSCIT\F.Y.B.Sc.IT\OOP's\NIKITA_OOPs\Practical_Related_Data\Practicals\seekptellp.exe

```
0
11
6

--------------------------------
Process exited after 0.1422 seconds with return value 0
Press any key to continue . . .
```

sample - Notepad

File Edit Format View Help

```
Hello India
```

Ln 1, Col 1 | 100% | Windows (CRLF) | UTF-8

# Final Program which is displaying changes in console screen also

seekptellp.cpp

```cpp
#include<iostream>
#include<fstream>
using namespace std;
int main()
{
ofstream fil("sample.txt",ios::out);
cout<<fil.tellp()<<endl;            //pointer location is 0
fil<<"Hello World";
cout<<fil.tellp()<<endl;     //after writing data to file pointer loction is 11

fil.seekp(-5,ios::end);       //after moved pointer backward from end to the 5th location using seekp
cout<<fil.tellp()<<endl;      //now loction will be 6
fil<<"India";                 //now on 6th location changed the data i.e from world to india
fil.close();

ifstream r("sample.txt",ios::in);      // To see the changed which has been done in file for that read operation performed
char ch;
while(!r.eof())
{
    ch=r.get();
    cout<<ch;
}
r.close();
return 0;
}
```

- Output:

```
F:\PATKAR_BSCIT\F.Y.B.Sc.IT\OOP's\NIKITA_OOPs\Practical_Related_Data\Practicals\seekptellp.exe
0
11
6
Hello India
--------------------------------
Process exited after 0.2399 seconds with return value 0
Press any key to continue . . .
```

# Use of tellg() and seekg()

# Use of tellg()

seekgtellg.cpp

```
#include<iostream>
#include<fstream>
using namespace std;
int main()
{
ofstream fil("tt.txt",ios::out);   //first file will have data
fil<<"Today is a great day";
fil.close();

ifstream r("tt.txt",ios::in);
cout<<r.tellg()<<endl;    //tellg will give strating location of pointer i.e 0

/*r.seekg(5,ios::beg);      //using seekg the pointer moved to the 5 location
cout<<r.tellg()<<endl;     //now tellg will give pointer loction as 5

char ch;
while(!r.eof())         //here will print data after 5th location
{
    ch=r.get();
    cout<<ch;
} */
r.close();
return 0;
}
```

- Output:

C:\Users\Admin\Desktop\C++Practical\seekgtellg.exe

```
0

--------------------------------
Process exited after 0.01639 seconds with return value 0
Press any key to continue . . .
```

tt - Notepad

File Edit Format View Help

```
Today is a great day
```

Ln 1, Col 1 | 100% | Windows (CRLF) | UTF-8

# Use of seekg()

seekgtellg.cpp

```
#include<iostream>
#include<fstream>
using namespace std;
int main()
{
ofstream fil("tt.txt",ios::out);   //first file will have data
fil<<"Today is a great day";
fil.close();

ifstream r("tt.txt",ios::in);
cout<<r.tellg()<<endl;    //tellg will give strating location of pointer i.e 0

r.seekg(5,ios::beg);      //using seekg the pointer moved to the 5 location
cout<<r.tellg()<<endl;     //now tellg will give pointer loction as 5

/*char ch;
while(!r.eof())         //here will print data after 5th location
{
    ch=r.get();
    cout<<ch;
} */
r.close();
return 0;
}
```

- Output:

C:\Users\Admin\Desktop\C++Practical\seekgtellg.exe

```
0
5

--------------------------------
Process exited after 0.01334 seconds with return value 0
Press any key to continue . . .
```

tt - Notepad

File Edit Format View Help

```
Today is a great day
```

Ln 1, Col 1 | 100% | Windows (CRLF) | UTF-8

# Final Program which is displaying changes in console screen

seekgtellg.cpp

```
#include<iostream>
#include<fstream>
using namespace std;
int main()
{
ofstream fil("tt.txt",ios::out);   //first file will have data
fil<<"Today is a great day";
fil.close();

ifstream r("tt.txt",ios::in);
cout<<r.tellg()<<endl;    //tellg will give strating location of pointer i.e 0

r.seekg(5,ios::beg);      //using seekg the pointer moved to the 5 location
cout<<r.tellg()<<endl;     //now tellg will give pointer loction as 5

char ch;
while(!r.eof())         //here will print data after 5th location
{
    ch=r.get();
    cout<<ch;
}
r.close();
return 0;
}
```

- Output:

```
C:\Users\Admin\Desktop\C++Practical\seekgtellg.exe
0
5
 is a great day
--------------------------------
Process exited after 0.0288 seconds with return value 0
Press any key to continue . . .
```

# ERROR HANDLING

- When dealing with the file errors might occurs such as :
  - File does not exist
  - Reading from file which is opened for writing only.
  - Path is not valid.
  - File already exist etc.
- To cope up with all these error we check whether file is opened successfully or what type of error has been generated.
- Some of the error handling functions with their description is given below :

| Name | Meaning |
| --- | --- |
| eof | It **returns a non-zero (true)** value when **end-of-file is encountered** while reading; ***otherwise returns zero (false).*** |
| fail | It **returns a non-zero (true)** value when an **input or output operation has failed**. |
| bad | If an **invalid operation is performed** or **any irrecoverable error has occurred** then this function returns **non-zero (true)** value.<br>However it ***returns zero (false)*** if it ***is possible to recover from any other reported error*** and continue the operation. |
| good | If **no error has occurred** then this function **returns non-zero(true)** value. If this function ***returns zero (false)*** then it means that the ***program can't perform further Operations*.** |

- E.g.

fileerror.cpp

```
1   #include<iostream>
2   #include<fstream>
3   using namespace std;
4   int main()
5   {
6       cout<<"---- File Error Handling ----";
7       ifstream in;
8       in.open("text12.txt",ios::in);
9           cout<<"\n eof() = "<<in.eof();
10          cout<<"\n fail() = "<<in.fail();
11          cout<<"\n bad() = "<<in.bad();
12          cout<<"\n good() = "<<in.good();
13          in.close();
14      return 0;
15  }
```

- Output:

```
C:\Users\Admin\Desktop\C++Practical\fileerror.exe
---- File Error Handling ----
 eof() = 0
 fail() = 1
 bad() = 0
 good() = 0
--------------------------------
Process exited after 0.01676 seconds with return value 0
Press any key to continue . . .
```

# COMMAND LINE ARGUMENTS

- If any **input value** is **passed through command prompt** at the **time of running** of program is **known as command line argument**
- It is **concept of passing the arguments <u>to</u>** the **main() function** by **using command prompt**
- In command line arguments application **main() function** will **takes two arguments** that is: **argc and argv**
- **General form** : main(int argc, char* argv[ ])

- **argc**
  - It is known as **argument counter**
  - It is an **integer type** variable
  - It **holds total number of arguments** which is passes into main function.
  - It takes number of arguments in the command line including program name.
- **argv**
  - It is know as **argument vector**
  - It is an **array of char* type** variable
  - It **holds actual arguments** which is passed to main function.

- E.g.

cmd1.cpp

```
1   #include <iostream>
2   using namespace std;
3
4   int main(int argc,char* argv[])
5   {
6       cout << "Number of command line arguments (argc) entered: " <<argc<<endl;
7
8       for (int i=0;i<argc;i++)
9           cout<<"argv["<<i<<"]"<<argv[i] << "\n";
10      return 0;
11  }
```

- Output:

C:\Users\Admin\Desktop\C++Practical\cmd1.exe

```
Number of command line arguments (argc) entered: 1
argv[0]C:\Users\Admin\Desktop\C++Practical\cmd1.exe

--------------------------------
Process exited after 0.01631 seconds with return value 0
Press any key to continue . . .
```

Command Prompt

```
Microsoft Windows [Version 10.0.19045.2604]
(c) Microsoft Corporation. All rights reserved.

C:\Users\Admin>cd desktop

C:\Users\Admin\Desktop>cd c++practical

C:\Users\Admin\Desktop\C++Practical>cmd1.cpp

C:\Users\Admin\Desktop\C++Practical>cmd1 n i k i t a
Number of command line arguments (argc) entered: 7
argv[0]cmd1
argv[1]n
argv[2]i
argv[3]k
argv[4]i
argv[5]t
argv[6]a

C:\Users\Admin\Desktop\C++Practical>_
```