Dt. 27th June 2022
Pg. B+

Exception Handling in C++

(1) Exceptions is any abnormal behaviour, runtime error. They are off beat situation in your programs where your program should be ready to handle it with appropriate response.

(2) C++ provides a built in error handling mechanism called exception handling. Using exception handling, you can more easily manage & respond to runtime errors.

Try, Catch & Throw

① Program statements that you want to monitor for exceptions are contained in a try block.
② If any exception occurs within the try block, it is thrown (using throw)
③ The exception is caught using catch & processed.

Syntax

```
try {
}
catch (type1 arg) {
}
catch (type2 arg) {
}
.
.
.
catch (typeN arg) {
}
```

Dt.
Pg. B+

```
#include <iostream>
using namespace std;
int main(){
    cout << "Welcome";
    try {
      throw 10;
      cout << "In In Try";
    }
    catch (int e) {
      cout << "In Exception no.: " << e;
    }
    cout << "In LastLine";
    getch();
}
```

Output

Welcome
ExceptionNo. - 10
LastLine

Try and catch can't exist alone but throw can.
More than one catches possible.

Slight Modification

```
main(){
    int i=3;
    cout << "Welcome";
    try {
      if(i==1)
        throw 1;
      if (i==2)
        throw 2;
      if(i==3)
        throw 3;
```

```
catch (double e) {
  cout << "In Exception No: " <<
    e << endl; }
catch (int e) {
  cout << "In Exception No.: " <<
    e << endl; }
cout << "In LastLine" << endl;
getch
```

EOY

Dt.
Pg. B+

If we want to catch any type of data then we need to mention (...) in catch block in order to catch all types of data.

```
catch(...){
}
```

If we do not mention anything after throw keyword then program gets terminated.

If we try to put the throw keyword inside any f^n which is called in try block then catch block will work perfectly fine.

There is no need to put the [redacted] argument inside the catch block only data type int

Catch

(1) When an exception is caught, argument will receive its value.

(2) If you don't need access to the exception itself, specify only type in the catch clause - arg is optional.

(3) Any type of data can be caught, including classes that you create.

Throw

(1) The general form of throw statement is,

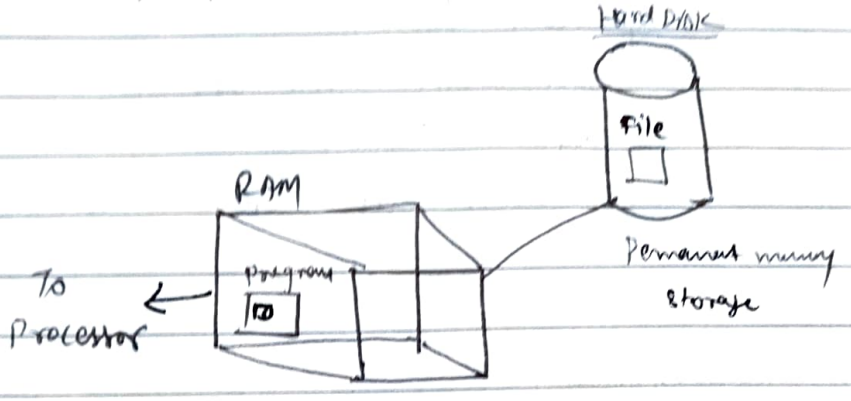
```
throw exception;
```

Throw must be executed either within the try block proper or from any f^n that code within the block calls.

Dt.
Pg. B+

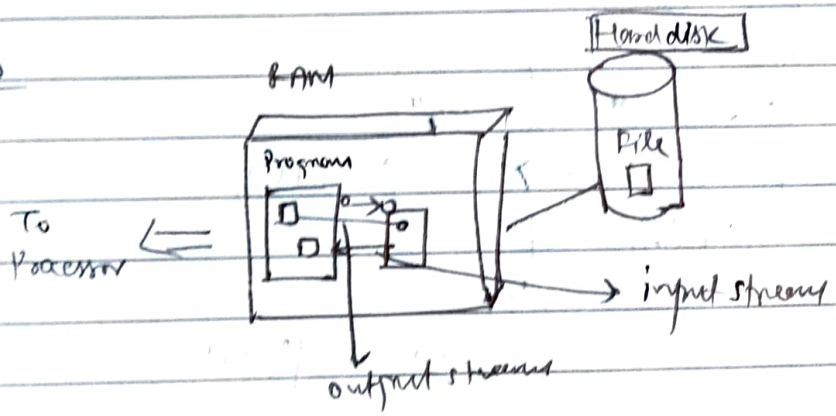
"File handling in C++"

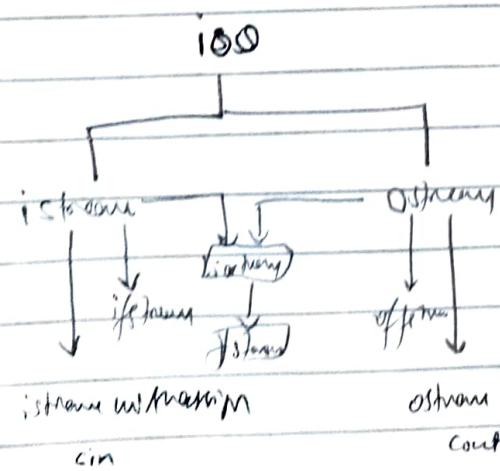
Data Persistence - life of Data



→ Variable's life is less than Program's life.
→ Handling of file → major operations → Read & Write
input

Streams





<< → insertion
>> → Extraction

Dt.
Pg. B+

①
```
#include <fstream.h>
#include <iostream>
using namespace std;
void main(){
        ofstream fout;
        fout.open("myfile.data");
        fout << "Hello";
        getch(); fout.close(); getch();
}
```

②
```
#include<fstream.h>
#include<iostream>
using namespace std;
void main(){ clrscr();
        ifstream fin; char ch;
        fin.open("myfile.data");
        fin>>ch; ⟶ ch = fin.get();
        while(!fin.eof()){
            cout<<ch;
            fin>>ch; ⟶ ch=fin.get();
        }
    fin.close()
    getch();
}
```

File Opening modes

① ios::in (Read/input) ✓
② ios::out (output/write) ✓
③ ios::app (append) ✗
④ ios::ate (update) ✓
⑤ ios::binary (binary) ✗

Dt. ____________ B+

Pg. ____________

```
#include <fstream.h>
void main() {
        ofstream fout;
        fout.open("myfile.dat", ios::out);
```

→ File open means bring file from hard disk to RAM for operations performing on them.

ios::out → blank

for adding new data and keeping previous data

Text mode v/s binary mode

1. Text mode is the default opening mode.
2. Binary mode can be specified with ios::binary.

tellg() and tellp()

1. tellg()
   - → Defined in istream class
   - → Its prototype is – streampos tellg();
   - → Returns the position of current character in the input stream.

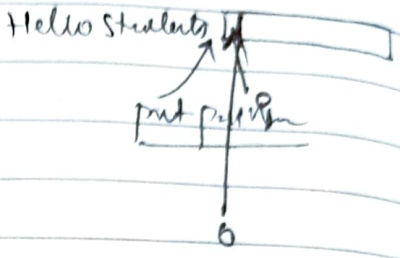
```
#include <iostream>
#include <fstream.h>
#include <conio.h>
using namespace std;
int main() {
        ifstream fin; char ch;
        fin.open("abc.txt"); int pos; pos = fin.tellg();
        cout << pos << endl; fin>>ch; pos=fin.tellg(); cout<<pos;
        getch();
}
```

5/9/

Dt.
Pg. : B+

put pointer

(2) tellp
- Define in ostream class
- tells the position from where we can write the text.




```
#include<iostream>
#include<fstream.h>
#include<conio.h>
using namespace std;
int main(){
    ofstream fout;
    char ch;
    fout.open("abc.txt", ios::app);
    int pos;
    pos = fout.tellp();
    cout<<pos<<endl; fout<<"mystring"; fout.close();
    getch();
    }
```

pos = fout.tellp();
cout<<pos

(1) seekg()
- In istream class
- Its prototype is
  - istream& seekg(streampos pos);
  - istream& seekg(streamoff off, ios_base::seekdir way);
  - pos is a new absolute position within the stream (relative to the beginning)
  - off is a offset value, relative to the way parameter.
  - way values ios_base::beg, ios_base::cur & ios_base::end.

Dt. _____ B+
Pg. _____

seekg → सिकजी

```
#include < iostream >
#include < fstream >
using namespace std;
int main(){
    ifstream fin;
    fin.open("abc.txt");
    cout << fin.tellg();   cout << fin.get();   fin.seekg(0);
    getch();
}
```

(char)

fin.seekg() → to point to any index

```
fin.seekg(2, ios_base::cur);
```

→ from current position to two places afterwards.

(2) seekp()

```
main(){
    ofstream fout;
    fout.open("abc.txt", ios::ate | ios::app);
    cout << fout.tellp();
    fout << "ABCDEFGH";
    cout << fout.tellp();
    fout.close();
    getch();
}
```

```
fout.seekp(2, ios_base::beg);
```