Template in C++ :

- the keycoord Template is used to define function template and class template.
- It is a way to make your function or class generalize as far as data type is concern.

- function Template -> function template is also known as generic function
- class Template

Syntax
template <class type> type func_name (type arg1, type arg2)
(keyword) or
for more than one type
template <class type, class type...>

ex
```
#include<conio.h>
using namespace std;
template <class X>
X big (X a, X b)
{ if (a>b)
      return(a);
  else return(b);
} int main()
{
  cout<<big(4,5);
  cout<<big(5.6,3.4);
}
```

X -> type -> place holder. It replaces by proper data type at the time of function call..

Output

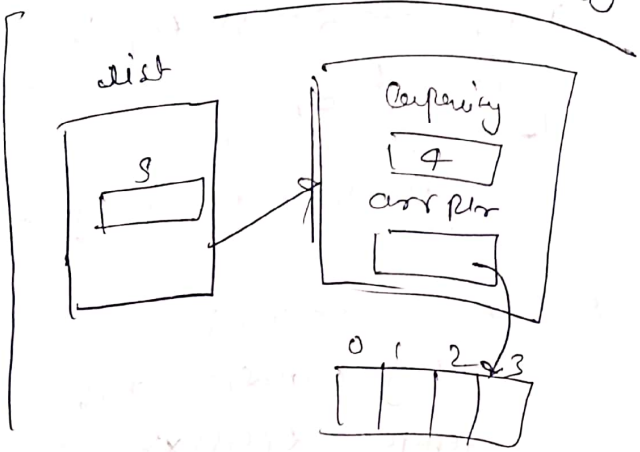
5 . 5.6

- Class Template - class Template is also known as generic class.

template < class type> class class_name{ ... };

```
#include<iostream>
using namespace std;
template <class X> class Arraylist
{
  private:
    struct ControlBlock
    { int capacity;
      X *arr_ptr;
    };
    ControlBlock *s;
  public:
  Arraylist(int capacity)
  {
    s = new ControlBlock
    s-> capacity = capacity;
    s-> arr_ptr = new X[s-> capacity];
  void addElement(int index, X data)
  { if (index >=0 && index <= s-> capacity - 1)
      s-> arr_ptr [index] = data;
    else
      cout << "\n Array index is not valid";
  }
  void viewElement (int index, X& data)
  { if (index >= 0 && index <= s-> capacity - 1)
      data = s-> arr_ptr [index];
    else
      cout << "\n Array index is not valid";
  }
  void displist()
  { int i;
    for(i=0; i<s-> capacity; i++)
      cout << "\n" << s-> arr_ptr [i];
  }
```

diagram overview



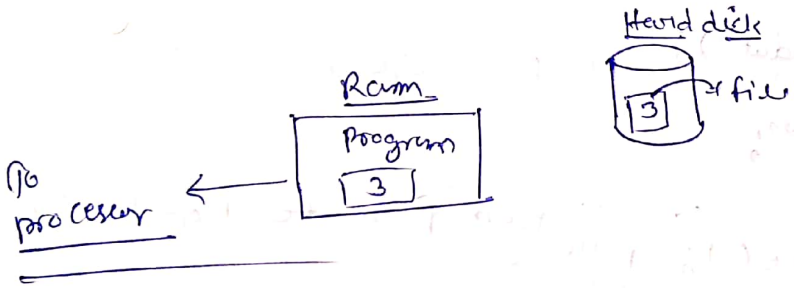
```
int main ()
{ int data;
~~Array list list (4);~~
Array list < int > list1(4);
        or
        arraylist
list1. addElement (0, 32);
list1.addElement (1, 41);
list1. add Elemt (2, 55);
list. viewList ();
```

* → Concrete class
→ polymorphism
→ file Handling
→ Random Access
→ object Serialization
→ Namespaces,
→ Exception Handling,
→ Over loading template function
→ Constructor and Destructor and Exception Handling.

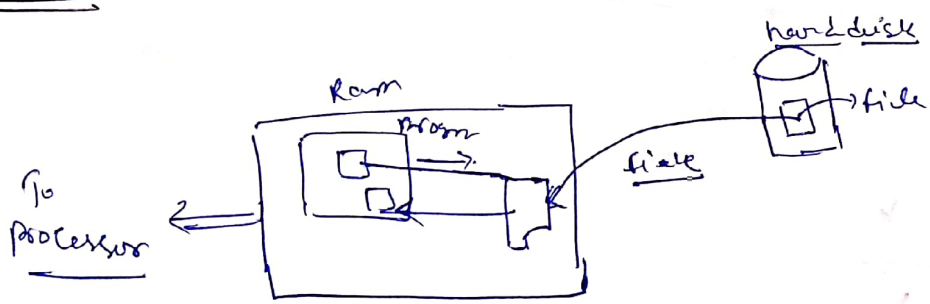
* Concrete class → Concrete classes implements a concrete concept they can be instantiated they may inherit from an abstract class or another concrete class. So far the classes we studied were concrete classes

→ file handling :-

Data persistence — life of data

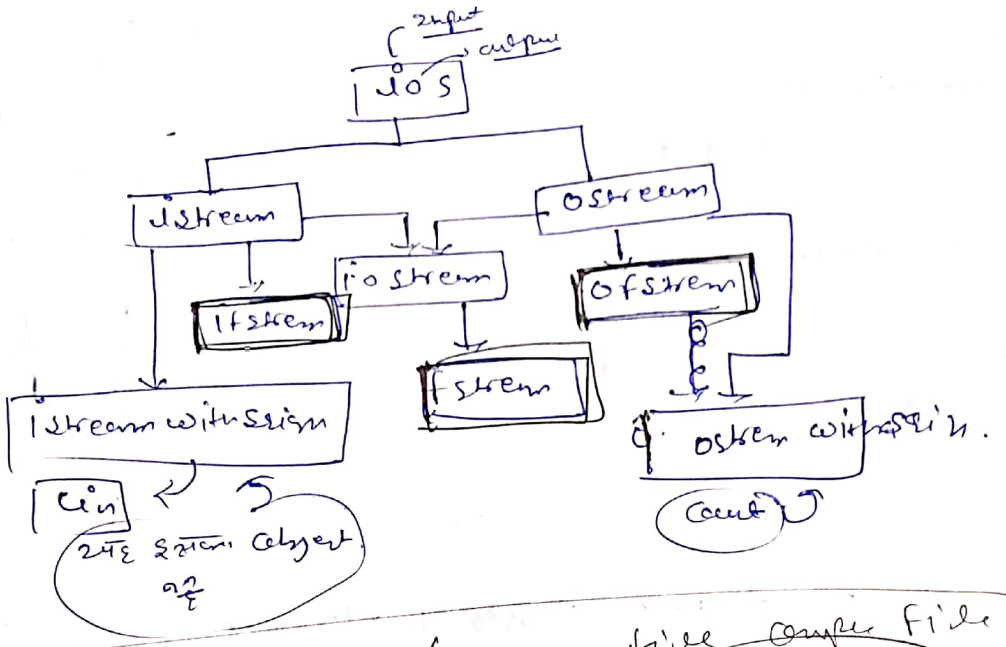


Streams -



Variable to file → output Stream

file → input Stream



for writing in file / create file copy file program to file

ofstream → ifstream, fstream

```
# include<fstream.h>
# include<iostream.h>
# include<conio.h>
void main()
{
  ofstream fout;          → for writing in file
  fout.open("my file.dat");   → hard disk में file बनाएगा
  getch();
  fout<<"hello";
  getch();
  fout.close();
  getch();
}
```

function (member function of ofstream)

file transfer from Ram to hard disk

---

file to program and printer screen
or reading of file

```
# include<fstream.h>
# include<iostream.h>
# include<conio.h>
void main()
{
  clrscr();
```

```
ifstream fin;
fin.open("myfile.dat");
fin>>ch;
while(!fin.eof())
{ cout<<ch;
  fin>>ch;
}
fin.close();
getch();
}
```

Hello Student

or

```
ch = fin.get();
ch = fin.get();
```

Hello Student

(gap)

file opening modes

- ios::in — input / read
- ios::out — output / write → old content erase ho jaata hai
- ios::app — append → ye is used to append the data (without erasing old data)
- ios::ate — update (randomly operation perform)
- ios::binary — binary

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

(→ not necessary here because output mode is default mode)

```
fout.open("myfile.dat",ios::app|ios::binary)
```

(→ file binary mode में open होगी
अगर ये नहीं दिया है तो file text mode में open होगी)

Exceptions handling :-

Exception :-

- Exception is any abnormal behaviour which is run time error.
- Exception are oft beat situation in your program where your program should be ready to handle it with appropriate response

Exception Handling -

- C++ provides a built-in error handling mechanism that is (it is a way) called exception handling.
- using exception handling, you can more easily manage and respond to runtime errors.

try, catch, throw :- all are keywords which are used in Exception handling process..

- program statements that you want to monitor for exception are contained in try block.
- If any exception occurs within the try block, it is thrown (using throw).
- The exception is caught, using catch, and processed.

Syntax :-

```
try {
}
catch (type1 arg) {
}
catch (type2 arg) {
}
---
catch (type n arg) {
```

-> all catch function are in square. i.e no coding are done. in b/w catch and try.

```
#include<iostream>
#include<conio.h>
using namespace std;
int main()
{
  cout<<"welcome";
  try{
    throw 10;
    cout<<"\n in Try";
  }
  catch(int e){
    cout<<"\n Exception no :"<<e;
  }
    cout<<"\n Last line";
    getche();
}
```

It is used for data type of any kind

It is not necessary to write argument

try and catch both are use together

Output → Welcome
Exception no : 10
Last line

Catch

- when an exception is caught, arg will receive its value.
- If you don't need access to the exception itself, specify only type in the catch clause. arg is optional.
- Any type of data can be caught, including classes that you create.

throw :

- the general form of the throw statement is :

  throw exception;

- Throw must be executed either within the try block proper or from any function that the code within the block calls.

File handling (Random Access)

* tellg()

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

* tellp()

→ Defined in ostream class
→ Its prototype is
  - streampos tellp();
→ Returns the position of the current character in the output stream.

tellg →

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

eg. file contains →
Hello students
01234 ...

Output = 0 1 2

cout << pos;
fin >> ch;
pos = fin.tellg();
cout << pos;
getch();

→ tellp:-

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

* seekg()

- Defined in istream class
- its prototype is
  - istream& seekg(streampos pos);
  - istream& seekg(streamoff off, ios_base::seekdir way);

- pos is new absolute position within the stream (relative to the beginning)
- off is offset value, relative to the way parameter
- way values ios_base::beg, ios_base::cur, ios_base::end

→ seekg() ->

file - ABC on open.

Hello Student my Str

```
#include<iostream>
#include<fstream>
#include<conio.h>
using namespace std;
int main()
{
  ifstream fin();
    fin.open("abc.txt");
  cout<<fin.tellg();
  cout<<"\n"<<(char)fin.get();
  cout<<(char)fin.get();
  cout<<"\n"<<fin.tellg();
```

```
fin.seekg(6);
cout<<"\n"<<fin.tellg();
cout<<"\n"<<(char)fin.get();
cout<<"\n"<<fin.tellg();
fin.seekg(-2, ios_base::end);
cout<<"\n"<<fin.tellg();
getch();
```

Output

```
0
He
2
6
S
7
18
```

* seekp()
  - Refined in Ostream class
  - its prototype is
    - ostream& seekp(streampos pos);
    - ostream& seekp(streamoff off, ios_base::seekdir way);
  - pos is new absolute position within the stream (relative to the beginning).
  - off is offset value, relative to the way parameter.
  - way values ios_base::beg, ios_base::cur and ios_base::end.