C++

پریسا حامد روح بخش

موسسه ی پارس پژوهان

فصل هشتم

- **Templates (Function Templates)**
- **Exceptions**
- **Files**

Function Templates

- A function template starts with the keyword **template** followed by template parameter(s) inside **<>** which is followed by the function definition.
- In the above code, **T** is a template argument that accepts different data types (**int**, **float**, etc.), and type name is a keyword.

```
template <typename T>
T functionName(T parameter1, T parameter2, ...) {
    // code
}
```

Function Templates

```
template <typename T>
T myMax(T x, T y)
{
    return (x > y)? x: y;
}

int main()
{
   cout << myMax<int>(3, 7) << endl;
   cout << myMax<char>('g', 'e') << endl;
   return 0;
}
```

Compiler internally generates and adds below code

```
int myMax(int x, int y)
{
    return (x > y)? x: y;
}
```

Compiler internally generates and adds below code.

```
char myMax(char x, char y)
{
    return (x > y)? x: y;
}
```

Function Templates

- Template functions can save a lot of time, because they are written only once, and work with different types.
- Template functions reduce code maintenance, because duplicate code is reduced significantly.

Function Templates

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

template<class T>
T  sum(T a, T b)
{
    return a + b;
}

int  main()
{
    int  x = 7, y = 15;

    cout << sum(x, y) << endl;
}
```

Function Templates

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

// One function works for all data types.  This would work
// even for user defined types if operator '>' is overloaded
template<typename T>
T  myMax(T x, T y)
{
    return (x > y) ? x : y;
}

int  main()
{
    cout << myMax<int>(3, 7) << endl;   // Call myMax for int
    cout << myMax<double>(3.0, 7.0)
         << endl;   // call myMax for double
    cout << myMax<char>('g', 'e')
         << endl;   // call myMax for char

    return 0;
}
```

Exceptions

- Problems that occur during program **execution** are called **exceptions**.
- In C++ exceptions are **responses** to anomalies that arise while the program is running, such as an attempt to **divide by zero.**

Exceptions

- **C++ exception handling is built upon three keywords:**

  try , catch, and throw.

The `try` statement allows you to define a block of code to be tested for errors while it is being executed.

The `throw` keyword throws an exception when a problem is detected, which lets us create a custom error.

The `catch` statement allows you to define a block of code to be executed, if an error occurs in the try block.

The `try` and `catch` keywords come in pairs:

Exceptions

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

int  main()
{
    int  age;

    cout << "Enter your age = ";
    cin >> age;

    try
    {
        if (age >= 18)
        {
            cout << "Access granted - you are old enough." << endl;
        }
        else
        {
            throw (age);
        }
    }
    catch (int myNum)
    {
        cout << "Access denied - You must be at least 18 years old.\n";
        cout << "Age is: " << myNum;
    }

    cout << endl;
}
```

Handle Any Type of Exceptions (...)

- If you do not know the throw type used in the try block, you can use the "three dots" syntax (...) inside the catch block, which will handle any type of exception:

Exceptions

```
try
{
    if (age >= 18)
    {
        cout << "Access granted - you are old enough.";
    }
    else
    {
        throw (age);
    }
}
catch (...)
{
    cout << "Access denied - You must be at least 18 years old.\n";
    cout << "Error number: " << age;
}
```

Exceptions

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

int  main()
{
    int  num1;

    cout << "Enter the first number:";
    cin >> num1;

    int  num2;
    cout << "Enter the second number:";
    cin >> num2;

    if (num2 == 0)
    {
        throw 0;
    }

    cout << "Result:" << num1 / num2;
}
```

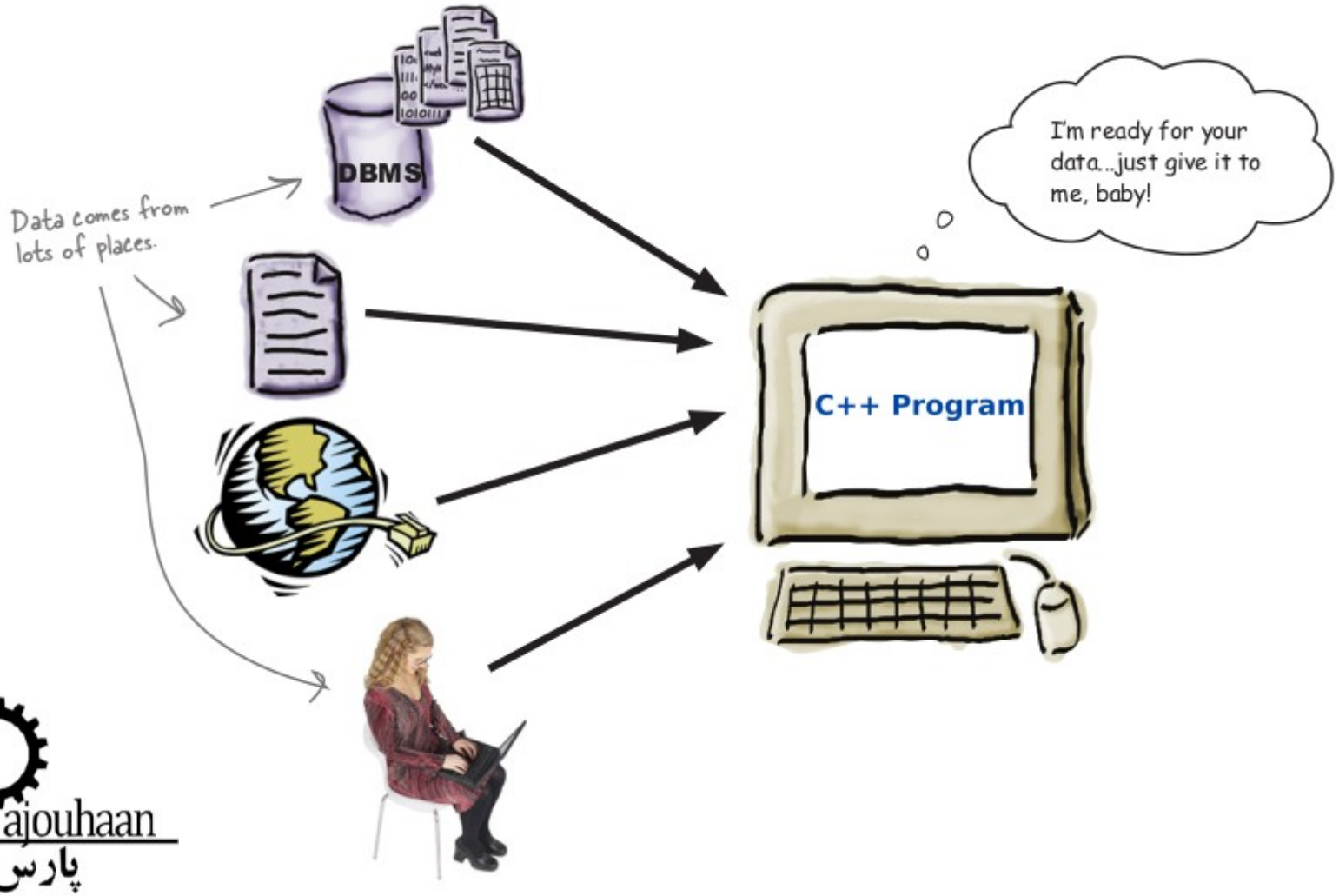
Exceptions

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

int  main()
{
    string  menu[] = { "fruits", "chicken", "fish", "cake" };

    try
    {
        int  x;
        cout << "Enter Number = ";

        cin >> x;

        // your code goes here
        if ((x >= 0) && (x < 4))
        {
            cout << menu[x] << endl;
        }
        else
        {
            throw 404;
        }
    }
    catch (...)
    {
        // and here

        cout << "404 - not found" << endl;
    }
}
```

Files

- Data is external to your program.
- Most of your programs conform to the input-process-output model: data comes in, gets manipulated, and then is stored, displayed, printed, or transferred.

Files

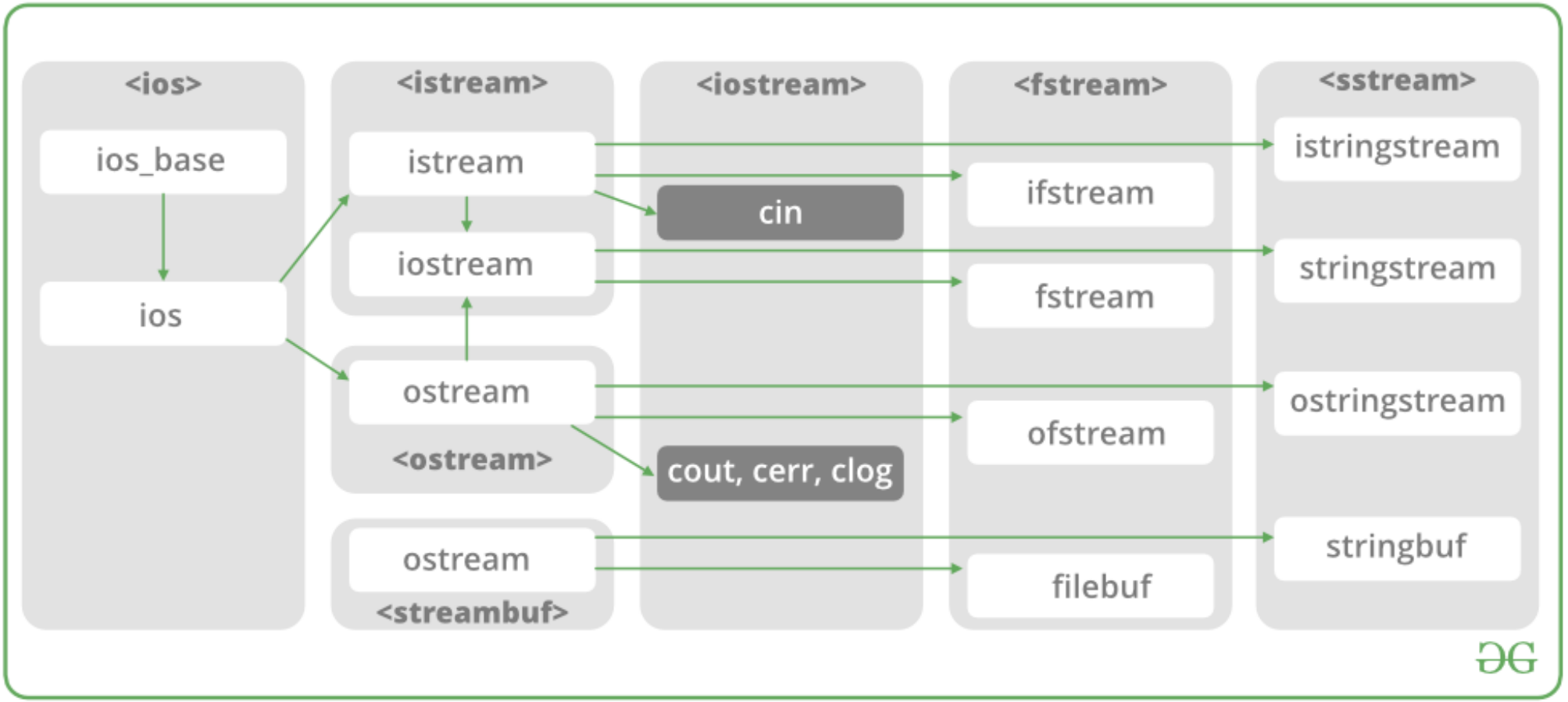


Files

- The **fstream** library allows us to work with files.
- To use the **fstream** library, include **both** the standard **<iostream> AND the <fstream>** header file:

```
#include <iostream>
#include <fstream>
```

Files


<ios>
ios_base
ios
<istream>
istream
iostream
ostream
<ostream>
ostream
<streambuf>
<iostream>
cin
cout, cerr, clog
<fstream>
ifstream
fstream
ofstream
filebuf
<sstream>
istringstream
stringstream
ostringstream
stringbuf

Files

Class	Description
`ofstream`	Creates and writes to files
`ifstream`	Reads from files
`fstream`	A combination of ofstream and ifstream: creates, reads, and writes to files

File handling is used for store a data permanently in computer.

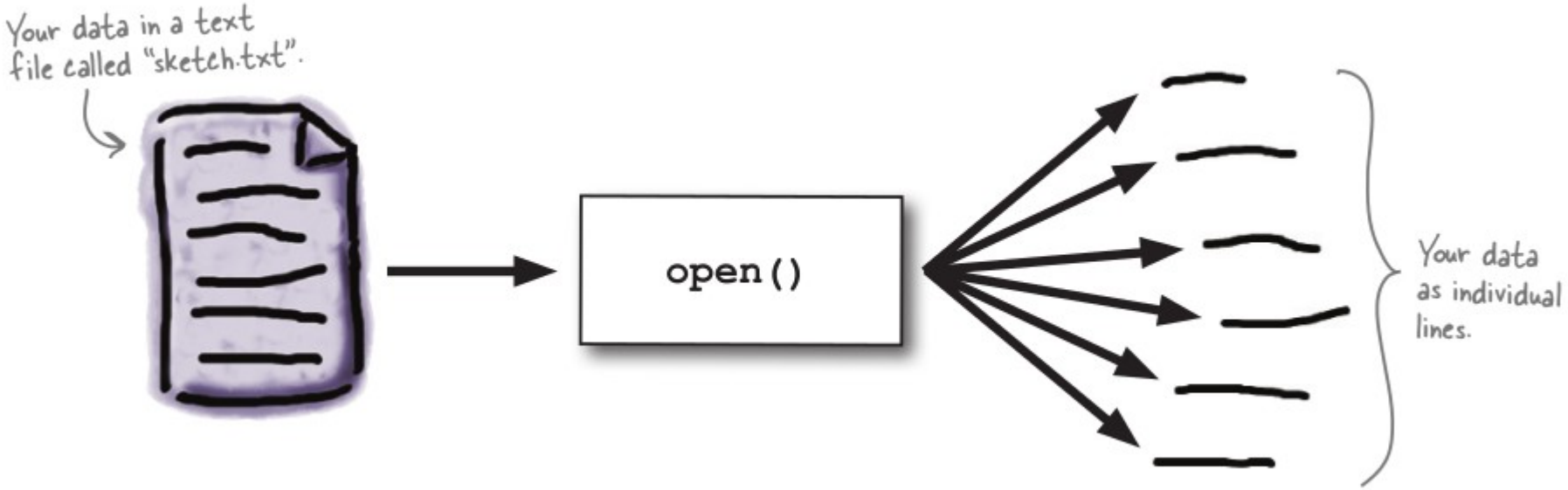
Using file handling we can store our data in secondary memory (Hard disk).

Files

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

int  main()
{
    // Create and open a text file
    ofstream  MyFile("filename.txt");

    // Write to the file
    MyFile << "Files can be tricky, but it is fun enough!";

    // Close the file
    MyFile.close();
}
```

Files

- **How does c++ read data from a file?**



Files

- **Opening a File**

  A file must be opened before you can read from it or write to it. Either 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.

```
void open(const char *filename, ios::openmode mode);
```

Files

Sr.No	Mode Flag & Description
1	**ios::app** Append mode. All output to that file to be appended to the end.
2	**ios::ate** Open a file for output and move the read/write control to the end of the file.
3	**ios::in** Open a file for reading.
4	**ios::out** Open a file for writing.
5	**ios::trunc** If the file already exists, its contents will be truncated before opening the file.

Files

```
int  main()
{
    char  data[100];

    // open a file in write mode.
    ofstream  outfile;

    outfile.open("afile.dat");

    cout << "Writing to the file" << endl;
    cout << "Enter your name: ";
    cin.getline(data, 100);

    // write inputted data into the file.
    outfile << data << endl;

    cout << "Enter your age: ";
    cin >> data;

    // again write inputted data into the file.
    outfile << data << endl;

    // close the opened file.
    outfile.close();
```

Files

```
    // open a file in read mode.
    ifstream  infile;
    infile.open("afile.dat");

    cout << "Reading from the file" << endl;
    infile >> data;

    // write the data at the screen.
    cout << data << endl;

    // again read the data from the file and display it.
    infile >> data;
    cout << data << endl;

    // close the opened file.
    infile.close();

    return 0;
}
```

File Position Pointers

- Both istream and ostream provide member functions for repositioning the file-position pointer. These member functions are seekg ("seek get") for istream and seekp ("seek put") for ostream.
- The argument to seekg and seekp normally is a long integer. A second argument can be specified to indicate the seek direction. The seek direction can be ios::beg (the default) for positioning relative to the beginning of a stream, ios::cur for positioning relative to the current position in a stream or ios::end for positioning relative to the end of a stream.

File Position Pointers

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

int  main()
{
    // Open a new file for input/output operations
    fstream  myFile("test.txt", ios::in | ios::out | ios::trunc);

    myFile << "Hello World";

    // Seek to 6 characters from the beginning of the file
    myFile.seekg(6, ios::beg);

    // Read the next 5 characters from the file into a
    // buffer
    char  A[6];
    myFile.read(A, 5);

    // End the buffer with a null terminating character
    A[5] = 0;

    // Output the contents read from the file and close it
    cout << A << endl;

    myFile.close();
}
```

File Position Pointers

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

int  main()
{
    std::ofstream  outfile;

    outfile.open("test.txt");

    outfile.write("This is an apple", 16);
    long  pos = outfile.tellp();
    outfile.seekp(pos - 7);
    outfile.write(" sam", 4);

    outfile.close();

    return 0;
}
```

Files

```
#include <fstream>
#include <iostream>
#include <stdexcept>
using namespace std;

int  main()
{
    try
    {
        char      buffer[256];
        ifstream  myfile("test.txt");
        myfile.exceptions(ifstream::eofbit | ifstream::failbit | ifstream::badbit);

        while (myfile)
        {
            myfile.getline(buffer, 100);
            cout << buffer << endl;
        }

        myfile.close();
    }
    catch (std::exception const &e)
    {
        cout << "There was an error: " << e.what() << endl;
    }
    return 0;
}
```

Question ?