Introdcution to Object-Oriented Programming

Programming Languages

- Machine-level Language programming
- Assembly-language programming
- High-level language programming
  - Procedural Programming
  - Structured programming
  - Object-Oriented Programming

Structured Programming

- Write moderately complex programs with an ease.
- Characterized by stand-alone subroutines, local variables, rich control constructs, and in general, less usage/no usage of GOTO.
- Structured languages reach their limit when a project becomes too large.
- Examples: C (Structured), C++, Java (Structured + OOP)

- Organization of the program (two ways)
  - Around its code (what is happening) or around its data (who is being affected).

- Organization of programs around the code, "code acting on data."
  - Example: a program written in a structured language such as C is defined by its functions, any of which may operate on any type of data used by the program.

Object-Oriented Programming Concepts

- Object-oriented programming:
  - Structured programming + several new concepts (encapsulation, abstraction, generic programming, etc.)

- Object-oriented programs are organized around data
  - Controlled access to data.
  - Define the data and the routines that are permitted to act on that data.
  - A data type defines what sort of operations can be applied to that data.

Encapsulation

- Binding of code and the data it manipulates.
- No outside interference and misuse, thus safe.
- The linking of code and data together forms an object.
- Within object, data, code or both private to that object or public.
- Private data/code can be accessible only by another part of the object, but not accessible to (a piece of code) from outside the object.
- Public data/code can be accessible within object as well as outside the object.
- An object is a variable of a user-defined type.

Abstraction

- Data abstraction is one of the important features of object oriented programming.

- **Abstraction in simple words**
  - For a particular functionality, what is to be done is known to the user but how it is done is not known to the user.
  - Ex. If you would like send an email to your friend, then you write an email and simply click **Send** button and an email gets sent to your friend (**What part**)
  - How an email mail was sent (**How part)** is hidden from you. The message is prepared in the format desired by the underlying network before the message is actually sent.

Example- abstraction

```
#include <iostream>
#include <string>
using namespace std;
class employee{
   int EId;
   string Ename;
   double salary, basic, allowance1,
allowance2;
   double computeSalary(int EId) {
   salary = basic + allowance1+
                 allowance2;
   return salary;
}
```

```
public:
setSalary(int Id, string name,
double b, double a1, double
a2)
{
EId =Id;
Ename = name;
basic =b;
allowance1=a1;
allowance2=a2;
computeSalary(EId);
}
```

```
void display(){
  cout<<"EmpId = "<<EId<<"\tName
=
  "<<Ename<<endl;
   cout<<"Employee Salary =
"<<salary;
  }
};
int main()
{
employee E1,E2;
E1.setSalary(1, “Mahesh”, 50000,
10000, 5000);
E1.display();
   }
```

Output: EmpId=1, Name:Mahesh

Salary: 65000

Example - Encapsulation

```
#include<iostream>
using namespace std;
 class sample  {
    private:
 // data is hidden from outside world
        int num;
  public:
        // function to set value of variable num
      void set(int x)  {
            num =x;    }
    // function to return value of variable num
        int get()         {
            return num;
        }
};
```

```
// main function
int main()
{
    sample obj;

    obj.set(50);

    cout<<obj.get();
    return 0;
}
```

Polymorphism

- One interface, multiple methods
  - Ex. A thermostat.
  - No matter what type of furnace our house has (gas, oil, electric, etc.), the thermostat works the same way.
  - The thermostat - same interface
  - Furnace - method
- Programming:
  - A stack of integer values, character values, and one for floating-point values.
  - One set of names, push( ) and pop( ), used for all 3 types of stacks.
  - Operators too can be overloaded : + used for all integers, floats, etc.
  - Operators used for user defined types.
  - Types: Compile time and run time.

Inheritance

- One object can acquire the properties of another object.
- Concept of classification - hierarchical classifications.
- Ex. a Red Delicious apple - a part of apple, which is a part of the fruit class, which is a part of the larger class food.
- Without the use of classifications, each object would have to define explicitly all of its characteristics.
- An object need only define those qualities that make it unique within its class.
- It is the inheritance mechanism that makes it possible for one object to be a specific instance of a more general case.
- Ex. Employee class - A general employee, special employee
- **Application:** Code reuse

Modular Programming

- A set of related procedures with the data they manipulate is often called a module.
- Decide which modules you want; partition the program so that data is hidden within modules.

[1] Provide a user interface for the stack (e.g., functions push () and pop ()).

[2] Ensure that the representation of the stack (e.g., an array of elements) can be accessed only through this user interface.

[3] Ensure that the stack is initialized before its first use.

Example -Modularity

```
namespace Stack { // interface
void push (char );
char pop ();
}
void f ()
{
Stack :: push (´c ´ );
if (Stack :: pop () != ´c ´ )
error ("impossible ");
}
```

```
namespace Stack { // implementation
const int max _ size = 200 ;
char v [max _ size];
int top = 0;
void push (char c ) { /* check for overflow and push c */ }
char pop () { /* check for underflow and pop */ }
}
```

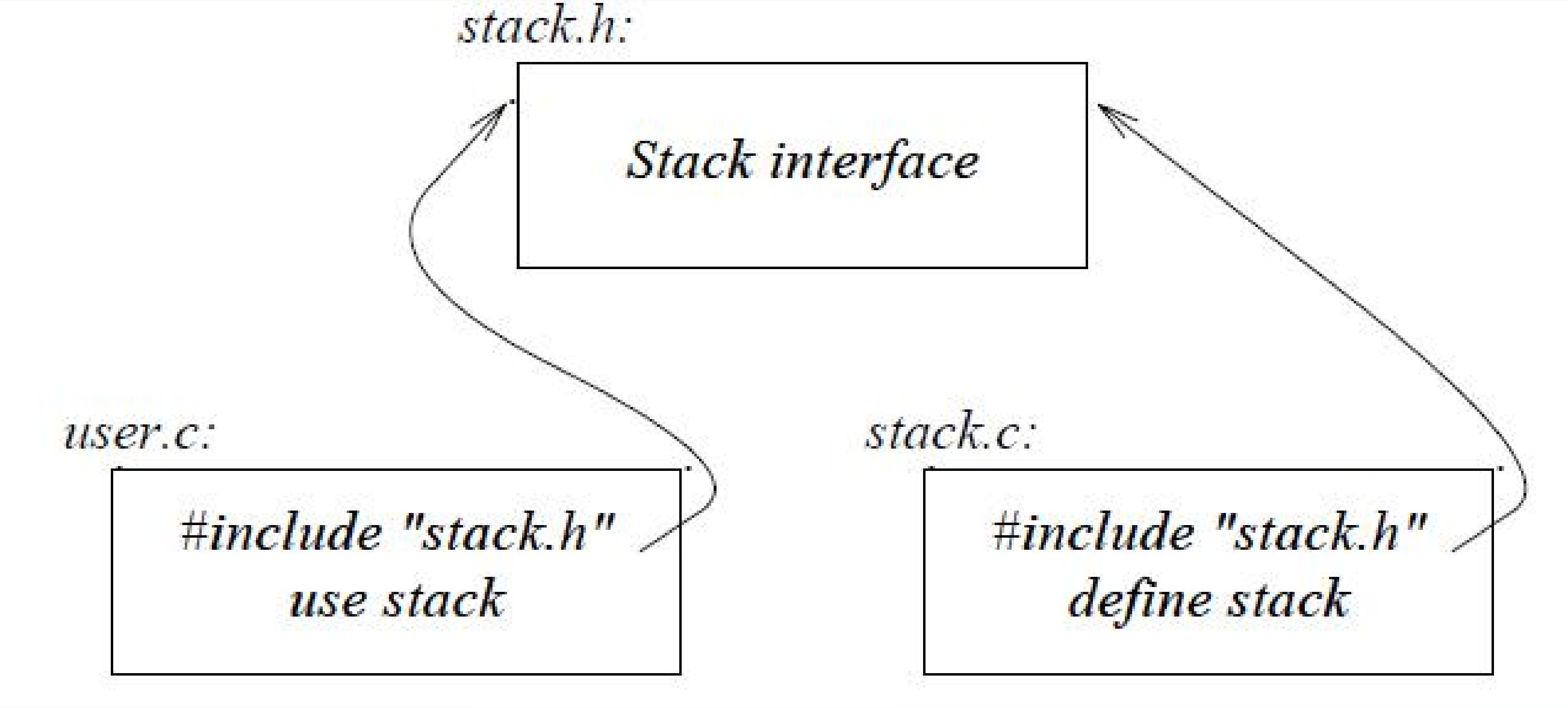
Example with Separate Compilation

```
namespace Stack
{ // interface
void push (char );
char pop ();
}
```

```
#include "stack .h "
// get the interface
void f ()
{
Stack :: push (´c ´ );
if (Stack :: pop () !=
´c ´ )
error ("impossible ");
}
```

```
#include "stack.h " // get the
interface
namespace Stack { //
representation
const int max _ size = 200 ;
char v [max _ size ];
int top = 0 ;
}
void Stack :: push (char c ) { /*
check for overflow and push c
*/ }
char Stack :: pop () { /* check for
underflow and pop */ }
```

Independent Modules



A Sample C++ Program

```
#include <iostream>  // I/O operations, no “.h” with a Standard C++
using namespace std; // std - Standard C++ library included
        A compiler directive
int main() { Note: void as an argument is not needed unlike C
int i;
cout << "This is output.\n"; // this is a single line comment
/* We can still use C style comments */
// input a number using >>
cout << "Enter a number: ";
cin >> i;
// now, output a number using <<
cout << i << "Square of i is " << i*i << "\n";
return 0; }
```

I/O Operators

```
#include <iostream>
using namespace std;
int main() {
float f;
char str [80];
double d;
cout << "Enter two floating point numbers: ";
cin >> f >> d;
cout << "Enter a string: ";
cin >> str;
cout << f << " " << d << " " << str;
return 0;
}
```

cout and cin are objects of ostream and istream classes in C++.

References

- C++: The Complete Reference, 4th Edition by Herbert Schildt , McGraw-Hill

- Teach Yourself C++ 3rd Edition by Herbert Schildt,

- The C+ + Programming Language, Third Edition by Bjarne Stroustrup, Addison Wesley