Object Oriented Programming using C++
Overview

- Problem Solving
- Features of an OOL
- Basic Syntax
- Programming Paradigms
Solving a Programming Problem

- Analysis
- Design
- Coding
- Management

- Programming paradigms
- Programming languages
Concepts & Relationships

- A rectangle uses lines
- A circle is an ellipse
- A wheel is part of automobile
- A set creates its elements
Some Features of OOP languages

● An OOP language should support
  – Easy Representation of
    ● Real-world objects
    ● Their States and Abilities
  – Interaction with objects of same type
  – Relations with objects of other type
  – Polymorphism and Overloading

● Reusability of code
● Convenient type definitions
Basic Syntax

- Same as C
- Additional operators
- Additional keywords
Programming Paradigms

- Procedural Programming (functions)
- Modular Programming (namespaces)
- Object Oriented Programming (classes)
- Generic Programming (templates)
Procedural Programming

- A program is a list of instructions
- Concentration is on *what is to be done?*
- Problems created by global data
- No access control!!
Modular Programming

- namespace
- Example to learn the syntax: A toy math library.
  - Matrix Module
    - Allocate
    - Transpose
    - Print
  - Vector Module
    - Allocate
    - Transpose
    - Print
Declaration of namespaces

// in file stdmatrix.h
namespace Matrix
{
    int** allocate(int r, int c);
    void print(int **matrix, int r, int c);
};

// in file stdvector.h
namespace Vector
{
    int *allocate(int size);
    void print(int *vector, int size);
};
Definition of Matrix functions

```cpp
int **Matrix::allocate(int r, int c)
{
    int **mat;
    mat = new int*[r];
    for (int i = 0; i < r; i++)
        mat[i] = new int[c];
    return mat;
}

void Matrix::print(int **matrix, int r, int c)
{
    for (int i = 0; i < r; i++)
    {
        for (int j = 0; j < c; j++)
            printf("%d ", matrix[i][j]);
        printf("\n");
    }
    printf("\n");
}
```
Definition of Vector functions

```c
int *Vector::allocate(int size) {
    int *vec = new int[size];
    return vec;
}

void Vector::print(int *vector, int size) {
    for (int i = 0; i < size; i++)
        printf("%d ", vector[i]);
    printf("\n");
}
```
// in the file main.c
#include <stdmatrix.h>
#include <stdvector.h>

using namespace Matrix;
using namespace Vector;

main()
{
    int **m = Matrix::allocate(3,4);
    int *v  = Vector::allocate(3);
    Matrix::print(m,3,4);
    Vector::print(v,3);
}
Adding functions to namespaces

- How does user add the transpose function now?

```cpp
// in the file mymatrix.h
#include "stdmatrix.h"
namespace Matrix
{
    int **transpose(int **matrix, int r, int c);
}
```
Adding definitions

// In the file mymatrix.cpp
#include “mymatrix.h”
int **Matrix::transpose(int **matrix, int r, int c)
{
  // Code for transposing and returning the matrix.
}
Using the transpose()

// in the file main.c
#include <mymatrix.h>
#include <myvector.h>

using namespace Matrix;
using namespace Vector;

main()
{
    int **m = Matrix::allocate(3,4);
    // fill with some random data in the matrix.
    int **tm = Matrix::transpose(m, 3, 4);
}
Object Oriented Programming

- Class
- Object
- Overloading (operators, functions)
- Inheritance
Class

- **class**: Definition of the entity
  - A tool for creating new types
    - Ex: Matrix and Vector
  - Access Control
    - public, private, and protected
  - Constructors, Destructors
  - Member functions
class Matrix
{
    private:
        int **m;
    public:
        Matrix();
        void print();
        void transpose();
        Matrix *transpose();
        ~Matrix();
};
Constructors

- Different Constructors for a class
  - Matrix();
  - Matrix(int init);  // 0 = zero matrix, 1 = Identity!!
  - Matrix(Matrix m);

- Writing a constructor
Objects

• Creation
  - Named object, free-store object (new & delete), non-static member object, an array element, temporary object, …etc

• Destruction

• Copying Objects
  - Direct assignment
  - Copy Constructors - Matrix::Matrix(Matrix &m);
Creating Matrix Objects

Matrix m;
Matrix *m = new Matrix();
Matrix *n = m;
Matrix *n = new Matrix(m);
Matrix ma[10];
Matrix *mp = new Matrix[10];
Matrix *np = new Matrix[10](m);
Interacting with Matrix Objects

Matrix m;
m.transpose();

Matrix *n = m.transpose();
Functions

- Static functions
- Constant functions and mutable
- Inline functions
- Helper functions
Operators

- Member and Non-Member Operators
- Unary
- Binary
  - Matrix & operator+= (Matrix a) // member
  - Matrix Matrix::operator + (Matrix b) // member
  - Matrix operator + (Matrix a, Matrix b) // non-member
A Code Example

class Complex
{
    private:
        int real;
        int imaginary;
    public:
        Complex(int r, int i= 0): real(r), imaginary(i);
        Complex(const Complex& a);
        int re() const;
        int im() const;
        void operator += (Complex a);
        Complex operator + (Complex b);
        void print();
};
Complex(const Complex& a) {
    real = a.real;
    imaginary = a.imaginary;
}
int re() const {
    return real;
}
int im() const {
    return imaginary;
}
void operator += (Complex a) {
    real += a.re();
    imaginary += a.im();
}
Complex operator + (Complex b) {
    Complex nc(*this);
    nc += b;
    return nc;
}
Inheritance

- What is inheritance?
- Parent and Derived
- Some properties of derived classes
  - Constructors
  - Member functions
  - Overloading
Inheritance in C++

Class Rectangle : public Shape
{
}
}
Class Student : public Person
{
}
}
Public, protected and private !!
Multiple Inheritance

class Transmitter {

... 

}

Class Receiver {

...

}

Class Radio : public Transmitter, public Receiver
Abstract Classes

class Shape
{
    public:
        virtual void rotate ( int )=0; // Make it pure !
        virtual void draw()=0;
}

Polymorphism through virtual functions ?
Virtual Base Classes

- class Derived1 : virtual public Base
- class Derived2 : virtual public Base
- class Derived3 : public Derived1, public Derived2
## Member Access Table

<table>
<thead>
<tr>
<th>Derived with</th>
<th>Private Base</th>
<th>Protected Base</th>
<th>Public Base</th>
</tr>
</thead>
<tbody>
<tr>
<td>Private Variables</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>Protected Variables</td>
<td>private</td>
<td>protected</td>
<td>Protected</td>
</tr>
<tr>
<td>Public Variables</td>
<td>private</td>
<td>protected</td>
<td>Public</td>
</tr>
</tbody>
</table>
Templates And Generic Programming

- Generic Functions and Classes
- \texttt{template <class X> FUNCTION\_NAME()}
- Example :
  - \texttt{template <class X> void swap(X \&a, X \&b);}
  - \texttt{template <class X, class Y> void function(X \&a, Y \&b)}
An example

template <class data_type> class List
{
    data_type data;
    List *next;

    public:
        List(data_type d);
        data_type getdata();
}

Creating Template Objects

- A list of integers `List<int> li;`
- A list of doubles `List<double> ld;`
- A list of strings `List<string> ls;`
- A list of addresses `List<address> ls;`
  - (where address is a predefined class/structure)
C++ I/O

- The `<iostream>`
- `using namespace std;`
- `cout, cin, cerr`
- `ifstream, ofstream and fstream`

```cpp
ifstream in;
in.open(const char *filename, openmode mode)
Mode can be ios::in, ios::out, ios::app, ios::binary,…etc
```
File Reading and Writing

```c++
#include <iostream>

int main() {
    std::ifstream in;
    double num;
    char str[20];

    in.read((char *)&num, sizeof(double));
    in.read(str,10);
    in.getline();

    return 0;
}
```
#include <vector>

vector <int> v;

Insertion, Access and Deletion:
    push_back, pop_back, insert, front, size, .. (see the manual)

Iterators vector<int>::iterator p = v.begin();

Other Templates :: List, Deque, Map.... etc!!
Basic Exception Handling

- Try
- Throw
- Catch
Syntax

```plaintext
try
{
    throw something;
}
catch (something)
{
    // Do whatever has to be done here.
}
catch (…) :: Catches everything.
```
try
{
    throw something;
}
catch (something)
{
    // Do whatever has to be done here.
}
catch (…) :: Catches everything.