ECE 220 Computer Systems & Programming

Lecture 20 – Intro to C++

Lecture#20 slides are adapted from Prof. Moon

The type journey

Objects

struct *

struct []

struct, typedef, enum

int *, char *, float *

int[], char[], float[]

int, char, float

Recall: C Structure

```c
#include <stdio.h>
#include <math.h>

//Example: a vector struct in C, which has 2 members: angle and length
typedef struct VectorStruct vector;

struct VectorStruct {
        double angle;
        double length;
 };

//function scaleVector to scale the length of a vector
void scaleVector(vector *v, double alpha) {
        v->length *= alpha;
 }
```

```c
int main(){
        vector a = {1.0,1};
        vector *d=&a;
        scaleVector(d, 2);
        printf("d's length is %f, d's angle is %f\n", d->length, d->angle);

        d->length=5;
        printf("d's length is %f, d's angle is %f\n", d->length, d->angle);
        return 0;
}
```

Note:

- scaleVector() is not part of the struct

No privacy for the members in a struct

- Anyone can access any member

C++ Class

Class can have access privileges for members

Class can contain function (we call it methods in C++)

Note:
Filename in C++:
file_name.cpp
To compile C++ file:
g++ file_name.cpp

```
#include <iostream>
//Example: a class in CPP, which has 2 members: angle and length
class vector{
private:
        double angle;
        double length;
public:
        //constructor to create an object
        vector(double a, double l){
        angle=a;
        length=l;
 }

//function scaleVector to scale the length of a vector
void scaleVector(double alpha) {
        length *= alpha;
 }

void printVector()
{
std::cout<<"Vector length is: "<<length<<std::endl;
std::cout<<"Vector angle is: "<< angle<<std::endl;
}
};
```

C++ Class

```cpp
int main(){
        vector a = {1.0,1};
        vector *d=&a;

// no access to the private member
/*
        a.length=3;
        d->length=5;
        printf("d's length is %f, d's angle is %f\n", d->length, d->angle);
*/
//use C++ methods/functions

        d->scaleVector(5);
        d->printVector();
        return 0;
}
```

```cpp
#include <iostream>
//Example: a class in CPP, which has 2 members: angle and length
class vector{
private:
        double angle;
        double length;
public:
        //constructor to create an object
        vector(double a, double l){
        angle=a;
        length=l;
 }

//function scaleVector to scale the length of a vector
void scaleVector(double alpha) {
        length *= alpha;
 }

void printVector()
{
std::cout<<"Vector length is: "<<length<<std::endl;
std::cout<<"Vector angle is: "<< angle<<std::endl;
}
};
```

C++

- **Object Oriented Programming** (OOP)
  Programming style associated with **class** and **objects** and other concepts like
  - Encapsulation
  - Inheritance
  - Polymorphism
  - Abstraction

- Class – a blueprint for object (*vector*).
  Similar to Struct in C except it defines
  - control "who" can access the data
  - provide functions specific for the class

Concepts Related to Class

- Object – an instance of the class (`vector a = {1.0,1};`)
  - shares the same function with other objects of the same class
  - but each object has its own copy of the data

- Member functions (methods) – functions that are part of a class

- Private vs. Public members
  - **private** members can only be accessed by member functions (default)
  - **public** members can be accessed by anyone

What do we need to know now?

- I/O function
- Namespace
- Dynamic allocation (malloc to *new*, free to *delete*)
- Function overloading
- Operator overloading
- Default Arguments

  ...

Basic Input & Output

C

```
#include <stdio.h>


printf(”Hello World : %d\n", a);

scanf(”%d", &a);
```

C++

```
#include <iostream>


std::cout<<"Hello World : "<<a<<std::endl;

std::cin >> a;
```

- cin: standard input stream (use with >>)
- cout: standard output stream (use with <<)
- endl: standard end line

** You can still use the c-style I/O functions by including <cstdio>

Using namespace

"using namespace" directive tells the compiler that the subsequent code is using names in a specific namespace

C

```
#include <stdio.h>


printf("Hello World : %d\n", a);

scanf("%d", &a);
```

C++

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

cout<<"Hello World : "<<a<<endl;

cin >> a;
```

- cin: standard input stream (use with >>)
- cout: standard output stream (use with <<)
- endl: standard end line

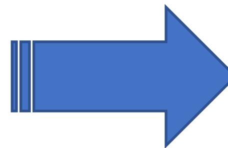
** You can still use the c-style I/O functions by including <cstdio>

Namespace

- A method for preventing name conflict.

```cpp
// code from Alice
void sayHello(){
    std::cout<<"Hello from Alice";
}

// code from Bob
void sayHello(){
    std::cout<<"Hello from Bob";
}

int main()
{
    sayHello();
}
```

Compile Error!



```cpp
namespace A{
// code from Alice
    void sayHello(){
        std::cout<<"Hello from Alice";
    }
}
namespace B{
// code from Bob
    void sayHello(){
        std::cout<<"Hello from Bob";
    }
}
int main(){
    A::sayHello();
    B::sayHello();
}
```

Use :: to resolve scope

Namespace with using

- We can use *using* keyword so that we don't have to use complete name all the time.

```
using namespace A;
int main()
{
   A::sayHello();
   B::sayHello();
   sayHello();
}
```

A::sayHello();

```
using namespace std;
      or
using std::cout;
using std::cin;
using std::endl;
```

```
cout<<"Hello from Alice"<<endl;
```

No more "std::" needed.

Dynamic Memory Allocation

- new – operator to allocate memory (similar to malloc in C)
- delete – operator to deallocate memory (similar to free in C)

C

```
int *ptr;
ptr = (int*) malloc(sizeof(int));
free(ptr);
```

C++

```
int *ptr;
ptr = new int;
delete ptr;
```

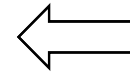
- To allocate/deallocate an array of memory,

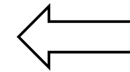
```
int *ptr;
ptr = new int[10];
delete []ptr;
```

Pass by Pointer(address) vs by Reference

```
void swap(int *a, int *b){
    cout<<"Pass by pointer"<<endl;

    int temp = *a;
    *a = *b;
    *b = temp;
}


void swap(int &a, int &b){
    cout<<"Pass by reference"<<endl;

    int temp = a;
    a = b;
    b = temp;
}
```

```
int val1, val2;
val1 = 10, val2 = 20;
swap(&val1, &val2);

swap(val1, val2);
```





Which function is called?

Reference

- Alias for a variable/object.
- A variable can be declared as reference by ‘&’ in the declaration.
- A reference must be initialized when declared.

```
int val = 10;

int *ptr = &val;// & to get address
int &ref = val;// & to declare reference

cout<<val<<endl;
cout<<*ptr<<endl;
cout<<ref<<endl;

ref = 20;
cout<<val<<endl;

val = 30;
cout<<ref<<endl;
```

Function Overloading

- Two or more functions can have the same name but different parameters (type & number, not return type)

```
int f(void){
    cout<<"int f(void)"<<endl;
}
int f(int a){
    cout<<"int f(int a)"<<endl;
}
int f(int a, int b){
    cout<<"int f(int a, int b)"<<endl;
}
int f(char a, char b){
    cout<<"int f(char a, char b)"<<endl;
}
```

```
int main()
{
    f();
    f(10);
    f(10, 20);
    f('a', 'b');
}
```

```
double f(char a, char b){
    cout<<"double f(char a, char b)"<<endl;
}
```

<- Can we add this function?

Default Arguments

- If the caller function does not provide a value for the arguments, then it is automatically assigned by the compiler with a default value.

```
int volume(int length, int width = 1, int height = 1){
   return length * width * height;
}

int main(){
   cout << volume(4) << endl;
}
```

Initialize Objects

```
class Person{
    char name[20];
    int age;
public:
    void ShowData();
};

int main(){
    Person *ptr = new Person(“Alice”, 20);
    Person p = {"Alice", 20};
}
```

Try to initialize just like structure.

Compile error

because the members (name and age) are *private*!

To solve,

1. Make the members *public* (not recommended)
2. Use “constructor”

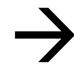
Constructor

- A special method which is invoked automatically at the time of object creation.
- Used to initialize the data members.
- It has the same name as class.
- 2 types: default constructor & parameterized constructor
- Overloading and default arguments are possible.
- No return value

default constructor:

compiler implicitly declare <u>if no constructor provided by user</u>.

```
class Person{
    char name[20];
    int age;
public:
    void ShowData();
};
```



```
class Person{
    char name[20];
    int age;
public:
    Person(){};
    void ShowData();
};
```

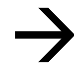
Constructor

- A special method which is invoked automatically at the time of object creation.
- Used to initialize the data members.
- It has the same name as class.
- 2 types: default constructor & parameterized constructor
- Overloading and default arguments are possible.
- No return value

default constructor:

compiler implicitly declare if no constructor provided by user.

```
class Person{
   char name[20];
   int age;
public:
   void ShowData();
};
```



```
class Person{
   char name[20];
   int age;
public:
   Person(){};
   void ShowData();
};
```

Constructor

```
class Person{
    char name[20];
    int age;

public:

    Person(char const *_name, int _age);
    Person(){};
    void ShowData();
};
Person::Person(char const *_name, int _age){
    strcpy(name, _name);
    age = _age;
}
int main(){
//    Person p = {"Alice", 20};
    Person p("Alice", 20);
}
```

Or `Person p = Person("Alice", 20);`

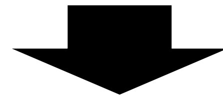
Default Constructor

```
class Person{
    char name[20];
    int age;
public:
    Person(char const *_name, int _age);
    void ShowData();
};
```

```
int main(){
    Person p1("Alice", 20);
    Person p2;
}
```

???

Looking for Person()construct.
But it's not declared.



```
class Person{
    char name[20];
    int age;
public:
    Person(char const *_name, int _age);
    Person(){};
    void ShowData();
};
```

← You need to explicitly declare the default constructor.

Destructor

- Destructor is a member function that destructs an object.
- It is called automatically when the object goes out of scope.
- It has the same name as class, but prefixed with ~.
- **No argument** (Overloading and default arguments are NOT possible).
- No return.

```
public:
    Person(){};
    Person(char const *_name, int _age);
    ~Person(){}; // destructor
```

Destructor

```
class Person{
    char *name;
    int age;
public:
    Person(){};
    Person(char const *_name, int _age);
    void ShowData();
    ~Person();
};
Person::Person(char const *_name, int _age){
    name = new char[strlen(_name)+1];
    strcpy(name, _name);
    age = _age;
}
Person::~Person(){
    delete []name;
}
```

→ Destructor is useful to deallocate memory

Operator Overloading

Not allowed overloading

```
.
::
?:
sizeof
```

- We can "redefine" the built-in operators (+, -, /, *, =,...).
- Overloaded operators are functions with special names: operator followed by the operator symbols.

```
int main(){
    Point p(1,2);

    p + 10;
    ↓
    p.operator+(10);

    p + 10.0;
```

```
class Point{
  private:
    int x,y;
  public:
    Point(int _x=0, int _y=0){x=_x; y=_y;}
    void ShowPosition();
    void operator+(int val){
      x = x + val;
      y = y + val;
    }
    void operator+(double val){
      x = x + val;
      y = y + val;
    }
```

Operator Overloading – A better way

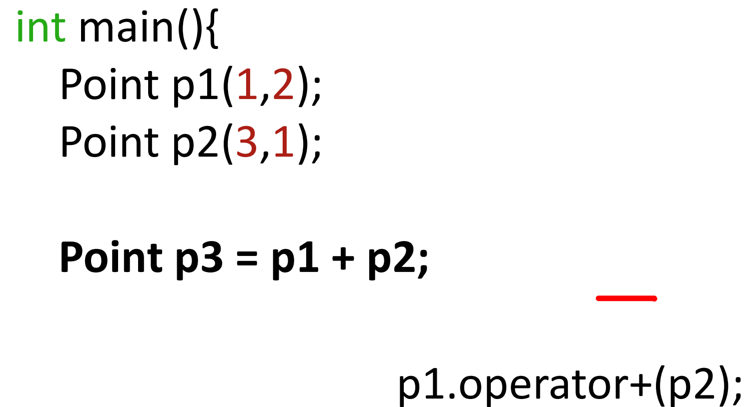
```
int main(){
    Point p1(1,2);

    Point p2 = p1 + 10;
```

```
p1.operator+(10)
```

```
class Point{
    private:
        int x,y;
    public:
        Point(int _x=0, int _y=0){x=_x; y=_y;}
        void ShowPosition();
        Point operator+(int val){
            Point temp(x+val, y+val);
            return temp;
        }
```

Operator Overloading

- You can also define the operators between two objects.




```
class Point{
  private:
    int x,y;
  public:
          . . .
    Point operator+(int val){
      Point temp(x+val, y+val);
      return temp;
    }
    Point operator+(Point p){
      Point temp(x+p.x, y+p.y);
      return temp;
    }
};
```