

CSE 333 – SECTION 5

C++ Review

Overview

- Classes, Constructors, new, delete, etc.
- More operator overloading

C++ classes

- Encapsulation and Abstraction
- Access specifiers:
 - Public: anything outside the class can access it
 - Protected: only this class and derived classes can access it
 - Private: only this class can access it
- Polymorphism
- Multiple Inheritance

Constructors and Destructors

- Function called when an object of a class is created
- Initializes the data members of a class
- Has the same name as the class
- Types –
 - Default – also called the empty constructor
 - Parameterized – Has arguments
 - Copy – Pass another already constructed object of the same class
- Destructors are invoked implicitly when a class instance is deleted / goes out of scope

new and delete

- new is used to allocate objects and primitive data types on the heap
- delete is used to deallocate these heap allocated objects
- Use “delete [] array” on an array
- Unlike malloc() and free(), new and delete are operators

Initialization vs Assignment

```
#define MAXSIZE 3

class IntArrayList {
public:
    IntArrayList() : array_(new int[MAXSIZE]), len_(0), maxsize_(MAXSIZE) { }

    IntArrayList(const int *const arr, size_t len) : len_(len), maxsize_(len*2) {
        array_ = new int[maxsize_];
        memcpy(array_, arr, len * sizeof(int));
    }

    IntArrayList(const IntArrayList &rhs) {
        len_ = rhs.len_;
        maxsize_ = rhs.maxsize_;
        array_ = new int[maxsize_];
        memcpy(array_, rhs.array_, maxsize_ * sizeof(int));
    }
    ...
private:
    int *array_;
    size_t len_;
    size_t maxsize_;
};
```

Memory diagram

Memory diagram

```
class Wrap {  
public:  
    Wrap() : p_(nullptr) { }  
    Wrap(IntArrayList *p)  
        : p_(p) { *p_ = *p; }  
    IntArrayList *p() const  
        { return p_; }  
private:  
    IntArrayList *p_  
}  
  
struct List {  
    IntArrayList v;  
}
```

```
int main() {  
    Wrap a;  
    Wrap b(new IntArrayList);  
    struct List c { };  
    struct List d {*b.p()};  
    a = b;  
    c = d;  
    Wrap *e;  
    e = &a;  
    Wrap *f = new Wrap(&d.v);  
    struct List *g =  
        new struct List;  
    g->v = *(new IntArrayList);  
    delete f;  
    delete g;  
    return 0;  
}
```


Operator Overloading

- A form of polymorphism.
- Give special meanings to operators in user-defined classes
- Special member functions in classes with a particular naming convention
- For E.g., for overloading the '+' operator, define a member function named operator+

Common operators

- The most commonly overloaded operators are
 - = (assignment operator)
 - + - * (binary arithmetic operators)
 - += -= *= (compound assignment operators)
 - == != (comparison operators)