

## CMSC433, Spring 2001 Programming Language Technology and Paradigms

Alan Sussman  
January 30, 2001

# C++

### C++ without Classes

- Don't need to say "struct"
- New libraries
- function overloading
  - confusing link messages
- default parameters
- **void \***
  - C++ requires explicit conversion from **void \***

CMSC 433, Spring 2001 - Alan Sussman

3

### More C++ without Classes

- Dynamic initialization of globals
  - no way to control order between files
- References
  - Compiler treats as pointers
  - Programmer doesn't
  - Once initialized to reference an address, cannot be made to reference a different address
  - Used for pass-by-reference parameters

CMSC 433, Spring 2001 - Alan Sussman

4

## CMSC433, Spring 2001 Programming Language Technology and Paradigms

Alan Sussman  
February 1, 2001

### Administrivia

- Quiz handed back on Tuesday
  - answers posted on Exams web page
- C++ readings from textbook posted on Readings web page
  - supplements material in lectures
- C++ lecture notes will be posted on Lectures web page soon

CMSC 433, Spring 2001 - Alan Sussman

6

## Administrivia (cont.)

- Class account info emailed yesterday
  - if you didn't get it, see me after class or in office hours
- My office hours posted on class home page
  - Tu 2:30-3:30, Wed 10:30-11:30, Th 4-5
  - TA's coming soon
- Project 1 - C++ - due Thursday, Feb. 15

CMSC 433, Spring 2001 - Alan Sussman

7

## Project 1

- C++ Linked List and Expanding Int Array
  - don't use other C++ libraries
  - Expanding Int array should use shared representation, copy on write
    - so that copying is fast
- Info on web page under Projects link

CMSC 433, Spring 2001 - Alan Sussman

8

## Quiz answers

## Copy constructor

```
SimpleArray(const SimpleArray & that) {  
    length = that.length;  
    data = new int[length];  
    for(int i = 0; i < length; i++)  
        data[i] = that.data[i];  
}
```

CMSC 433, Spring 2001 - Alan Sussman

10

## BAD Copy constructor

```
// DO NOT DO THIS  
SimpleArray(const SimpleArray & that) {  
    *this = that;  
}
```

CMSC 433, Spring 2001 - Alan Sussman

11

## Operator=

```
SimpleArray& operator=(const SimpleArray & that)  
{  
    if (this == &that) return *this;  
    delete [] data;  
    length = that.length;  
    data = new int[length];  
    for(int i = 0; i < length; i++)  
        data[i] = that.data[i];  
    return *this;  
}
```

CMSC 433, Spring 2001 - Alan Sussman

12

## operator= notes

- operator= is not operator==
- generally better to define as member function than standalone
- check for this == &that
- free storage for this

CMSC 433, Spring 2001 - Alan Sussman

13

## Destructor

```
~SimpleArray() {  
    delete [] data;  
}
```

CMSC 433, Spring 2001 - Alan Sussman

14

## destructor notes

- no need to set length = 0 or data = null
- Everything is much simpler if you never allow data==null

CMSC 433, Spring 2001 - Alan Sussman

15

## implementing OnePlaceBuffer

```
public class Buf implements cmcsc433.OnePlaceBuffer {  
    private Object data;  
    public synchronized Object take()  
        throws InterruptedException {  
        while (data == null)  
            wait();  
        Object tmp = data;  
        data = null;  
        notifyAll();  
        return tmp;  
    }  
}
```

CMSC 433, Spring 2001 - Alan Sussman

16

## OnePlaceBuffer (cont.)

```
public synchronized void put(Object o)  
    throws InterruptedException {  
    while (data != null)  
        wait();  
    data = o;  
    notifyAll();  
}
```

CMSC 433, Spring 2001 - Alan Sussman

17

## Last Lecture

- C++ without classes
  - new libraries (e.g., STL)
  - function overloading
  - default parameters
  - dynamic initialization of globals
  - references

CMSC 433, Spring 2001 - Alan Sussman

18

## References

```
int x,y;
int *p = &x;
int &q = y;
q++; // increments y
(*p)++; // increments x
q = x; // assigns value of x to y
p=&y; // makes p point to y;
```

CMSC 433, Spring 2001 - Alan Sussman

19

## new/delete

- use **new A** to allocate one A
- use **new A[size]** to allocate an array of **size** A's
- use **delete** to free one A
- use **delete[]** to free an array of A
  - for all elements of an array
  - **delete []** must be given pointer to first element

CMSC 433, Spring 2001 - Alan Sussman

20

## Data Abstraction

- Avoid name clashes
- Hide implementation
- Override “standard” functionality

CMSC 433, Spring 2001 - Alan Sussman

21

## IntArray example

- Without OO:

```
void IA_init(IntArray *ia);
void IA_cleanup(IntArray * ia);
void IA_setSize (IntArray * ia, int value);
int IA_getSize (IntArray * ia);
void IA_setElem(IntArray * ia,
                int index, int value);
int IA_getElem(IntArray * ia, int index);
```

- With OO:

```
class IntArray {
public:
    IntArray();
    ~IntArray ();
    void setSize (int value);
    int getSize();
    void setElem(int index,
                int value);
    int getElem (int index);
private:
    ...
}
```

CMSC 433, Spring 2001 - Alan Sussman

22

## Access control

- A member can be public/protected/private
- An access specification controls access to all following members (until the next access specification)
- For classes, initial/default access is private
  - For structs, initial/default access is public
  - no other difference between classes and structs

CMSC 433, Spring 2001 - Alan Sussman

23

## Access levels

- public – any function/method allowed access
- private - only methods in the class allowed access
- protected - like private, except that subclasses (derived classes) can access inherited members

CMSC 433, Spring 2001 - Alan Sussman

24

## Protected access

- Class A {  
  private: int x;  
  protected: int y; }
- Class B : public A {  
  static int foo(A a, B b) {  
    int i = a.x; // illegal  
    int j = a.y; // illegal  
    int k = b.x; // illegal  
    int n = b.y; // LEGAL  
  ...}

CMSC 433, Spring 2001 - Alan Sussman

25

## Friends

- A class X can declare a class Y or a function f as a friend
  - Gives Y or f access to all of X's private and protected data
- Friendship is not transitive
  - If X declares Y as a friend, and Y declares Z as a friend, Z can't see X's private data
- It's not inherited either
  - a class derived from Y cannot access X's private or protected data

CMSC 433, Spring 2001 - Alan Sussman

26

## Hidden functions

- class A {  
  A(); // void constructor  
  A(const A &); // copy constructor  
  A& operator=(const A &); // operator=  
  ~A(); // destructor  
}

CMSC 433, Spring 2001 - Alan Sussman

27

## void/default constructor

- Invoked whenever an instance is created without being initialized.
- If no constructor is provided, default public one is created
  - invokes void constructor on each base class and instance variable (member)
  - members of built-in types not initialized

CMSC 433, Spring 2001 - Alan Sussman

28

## copy constructor

- Invoked to create a new value from an old value
  - for initialization
  - for parameters passed by value
  - for objects returned as values
- If no copy constructor supplied, default public one created
  - invokes copy constructor for each base class and instance variable (and copies bits for built-in types)
  - almost never correct if you have pointers
- If copy constructor supplied, must also supply void constructor

CMSC 433, Spring 2001 - Alan Sussman

29

## operator=

- If no operator=() defined, defines default public one
  - does operator= on each base class and instance variable
- Watch for a = a
- A::operator=(const A&) usually returns A&
  - allows a = b = c;

CMSC 433, Spring 2001 - Alan Sussman

30