

CS481 - Object-Oriented Programming

ADT's in C++

- ❖ Very similar to C example
- ❖ Language support: `class`
 - Member functions (`obj.init()`)
 - Name scoping (`::`)
 - No explicit structure pointer (`->`)

1

C++ class

- ❖ Syntax like a struct
 - C++ struct adds member functions
 - ◆ But seldom use struct this way
- ❖ Inheritance & polymorphism
 - But let's defer that discussion

2

ADT Data Members

C	<pre>typedef struct { int max_elem; int current_elem; int* elems; ... }* Stack;</pre>
C++	<pre>class Stack { int max_elem; int current_elem; int* elems; ... }</pre>

3

User-defined Types in C

- ❖ ... are not first class citizens
- ❖ struct must precede user types
- ❖ unless:


```
typedef struct mytype mytype;
```

4

ADT Operations

Initialization & cleanup

C	<pre>extern Stack CreateStack(void); extern void DestroyStack (Stack stack);</pre>
C++	<pre>class Stack {... public: void init(); void free(); ...};</pre>

5

ADT Operations

By default, members not visible

C++	<pre>extern Stack CreateStack(void); extern void DestroyStack (Stack stack);</pre>
C++	<pre>class Stack {... public: void init(); void free(); ...};</pre>

6

CS481 - Object-Oriented Programming

ADT Operations
"stack" argument implicit

```
extern Stack CreateStack(void);
extern void DestroyStack
  (Stack stack);
```

```
class Stack
{...
public:
  void init();
  void free();
...};
```

C++

7

ADT Operations
Names don't collide

```
extern Stack CreateStack(void);
extern void DestroyStack
  (Stack stack);
```

```
class Stack
{...
public:
  void init();
  void free();
...};
```

C++

8

ADT Operations
Don't need void for "no args"

```
extern Stack CreateStack();
extern void DestroyStack
  (Stack stack);
```

```
class Stack
{...
public:
  void init();
  void free();
...};
```

C++

9

ADT Operations
Push & pop

```
extern void PushStack
  (Stack stack, int element);
extern int PopStack(Stack stack);
```

```
void push(int element);
int pop();
...};
```

C++

10

ADT Operations
Observer (query) operations

```
extern Boolean IsStackEmpty
  (const_Stack stack);
extern int TopStack
  (const_Stack stack);
```

```
bool isEmpty() const;
int top() const;
};
```

C++

11

ADT Operations
New "constant access" syntax

```
extern Boolean IsStackEmpty
  (const_Stack stack);
extern int TopStack
  (const_Stack stack);
```

```
bool isEmpty() const;
int top() const;
};
```

C++

12

CS481 - Object-Oriented Programming

"const" Declarations

- ❖ `const int* p;` //can't change data (*p)
- ❖ `int* const p;` //can't change pointer (p)
- ❖ `const int* const p;` // neither!
- ❖ `class c1 { ...`
`int func1(int) const;`
 - member fn. can't change instance vars.

13

"const" Declarations

```
❖ class c1 {
    int i;
public:
    int func1(int) const;
    const int func2();
    const int func3() const;
};
```

14

"const" Declarations

```
❖ class c1 {
    int i;
public:
    int func1(int) const;
    const int func2();
    const int func3() const;
};
```

15

"const" Declarations

```
❖ class c1 {
    int i;
public:
    int func1(int) const;
    const int func2();
    const int func3() const;
};
```

16

ADT Usage

```
Stack stack1;
int ii;
int elem;

stack1 = CreateStack();
```

C++

```
Stack stack1;
int ii;
int elem;

stack1.init();
```

17

ADT Usage

```
Stack (stack1);
int ii;
int elem;

(stack1) = CreateStack();
```

Pointer versus instance

```
Stack (stack1);
int ii;
int elem;

stack1.init();
```

18

CS481 - Object-Oriented Programming

ADT Usage Member function call

```
Stack stack1;
int ii;
int elem;
stack1 = CreateStack();
```

C++

```
Stack stack1;
int ii;
int elem;
stack1.init();
```

19

ADT Usage

```
Stack stack1;
int ii;
int elem;
stack1 = CreateStack();
```

Member selection operator

```
Stack stack1;
int ii;
int elem;
stack1.init();
```

20

ADT Usage

```
if (IsStackEmpty(stack1))
    printf("Stack 1 is empty.\n");
for (ii = 1; ii <= 12; ii++)
    PushStack(stack1, ii);
```

```
if (stack1.isEmpty())
    printf("Stack 1 is empty.\n");
for (ii = 1; ii <= 6; ii++)
    stack1.push(ii);
```

21

Implementation

```
void Stack::init()
{
    max_elem = STACK_SIZE_INCR;
    current_elem = -1;
    elems = (int*)
        malloc(max_elem * sizeof(int));

    assert(elems != NULL);
    assert(isEmpty());
}
```

22

Implementation Name scoping

```
void Stack::init()
{
    max_elem = STACK_SIZE_INCR;
    current_elem = -1;
    elems = (int*)
        malloc(max_elem * sizeof(int));

    assert(elems != NULL);
    assert(isEmpty());
}
```

23

Implementation Direct member access

```
void Stack::init()
{
    max_elem = STACK_SIZE_INCR;
    current_elem = -1;
    elems = (int*)
        malloc(max_elem * sizeof(int));

    assert(elems != NULL);
    assert(isEmpty());
}
```

No "stack->"

24

CS481 - Object-Oriented Programming

Object Pointer: *this*

Hidden pointer argument passed to member function

```
stack1.Push(ii);
```

```
void Stack::Push(int element)
{...}
```

25

Name Mangling

- ❖ What if two classes have same member function name?
- ❖ Compiler knows, but ...
- ❖ Linker is dumb!
 - Compiler creates replacement name

26

Header File Nesting

Only one inclusion, even if multiple references to header file.

```
/* boole.h */
#ifndef BOOLE_H
#define BOOLE_H

typedef int Boolean;

#endif /* BOOLE_H */
```

"idempotence"

27

C++ Access Control

- ❖ **Public**
 - Member visible outside class
- ❖ **Private** (default)
 - Visible only inside member functions
 - Friends
- ❖ **Protected** (more later - inheritance)

28

C++ Access Control

- ❖ **Multiple access blocks**
 - Each preceded by keyword
 - No required order, duplicate blocks ok
- ❖ **Default access**
 - class - private
 - struct - public

29

C++ Access Control

- ❖ **Multiple access blocks**
 - Each preceded by keyword
 - No required order, duplicate blocks ok
- ❖ **Default access**
 - class - private
 - struct - public

Only difference between class and struct!!

30

CS481 - Object-Oriented Programming

Friendship & Access

- ❖ Normal access control
 - By class member functions - `private`
 - By all functions - `public`
- ❖ Access by specific functions?
 - Class can declare as a `friend`
 - ◆ Single function or all functions in a class

31

Nested Classes

- ❖ Can declare one class within another
- ❖ Nested class name not global
 - Reference with “`::`” operator
- ❖ No direct impact on access
 - Can still declare `friend` or not

32

Global Scope

- ❖ `class c1 { ...
 int printf(list*) const;
};`
- ❖ `printf("hi")`
 - calls the member function
- ❖ `::printf("hello world\n");`
 - calls the global (library) function

33