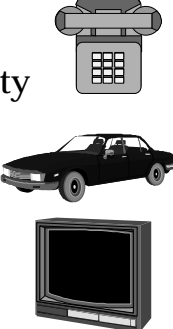


CS481 - Object-Oriented Programming

Abstraction

- ❖ Simplify complex reality
 - View at high level
 - ◆ Ignore low-level detail
 - Facilitate understanding
- ❖ External appearance
 - Hide internal operation



Types of Abstraction

- ❖ Procedural
 - Well-defined operation
 - Viewed as a whole
 - ◆ Complex implementation?
- ❖ Data structuring
 - Complex data type viewed as entity

Data Type Hierarchy

- ❖ Hardware data types
 - Integers, characters, reals
- ❖ Virtual data types
 - Arrays, pointers (language/compiler)
- ❖ Abstract data types (ADT)
 - Application-specific (programmer)

ADT Specification

- ❖ Domain of values
- ❖ Operations
- ❖ Components (if structured)
 - Data types
 - Relationships

ADT & OO Terms

- ❖ ADT instances
 - Variables of ADT type
 - Also known as objects
- ❖ ADT operations
 - ADT functions (or member functions)
 - Messages ("send a message to object")

ADT & OO Terms

ADT: no inheritance or polymorphism

- ❖ ADT instances
 - Variables of ADT type
 - Also known as objects
- ❖ ADT operations
 - ADT functions (or member functions)
 - Messages ("send a message to object")

CS481 - Object-Oriented Programming

Information Hiding

- ❖ ADT defined by externals
 - Opaque data type
- ❖ Limited interface access
 - Often, no direct access to data
 - Defined operations only
 - ◆ Implementation can change

7

ADT Advantages

Courtesy Admiral Grace Hopper, USN

- ❖ Maintainability
 - Limit "domino effect"

$$q = p(n-1)$$

n = number of modules in a system

$$m = 1 + q + q^2 + q^3 + \dots$$

$$= \frac{1}{1-q}$$

8

ADT Advantages

p = probability that change in one module will cause change in another

- ❖ Maintainability
 - Limit "domino effect"

$$q = p(n-1)$$

$$m = 1 + q + q^2 + q^3 + \dots$$

$$= \frac{1}{1-q}$$

9

ADT Advantages

m = number of changes resulting from single initial change

- ❖ Maintainability
 - Limit "domino effect"

$$q = p(n-1)$$

$$m = 1 + q + q^2 + q^3 + \dots$$

$$= \frac{1}{1-q}$$

10

ADT Advantages

- ❖ Maintainability
 - Limit "domino effect"

$$q = p(n-1)$$

$$m = 1 + q + q^2 + q^3 + \dots$$

What if $q \rightarrow 1$?

11

ADT Advantages

- ❖ Maintainability
- ❖ Security and integrity
 - Limited set of operations
 - Validity checking

12

CS481 - Object-Oriented Programming

ADT Advantages

- ❖ Maintainability
- ❖ Security and integrity
- ❖ Sharing and re-use

13

ADT Advantages

- ❖ Maintainability
- ❖ Security and integrity
- ❖ Sharing and re-use
- ❖ Intellectual manageability

14

Building ADT's

- ❖ Two different programmers (?)
 - ADT implementor
 - ADT user
- ❖ Public interface (".h" file)
- ❖ Hidden implementation (library)

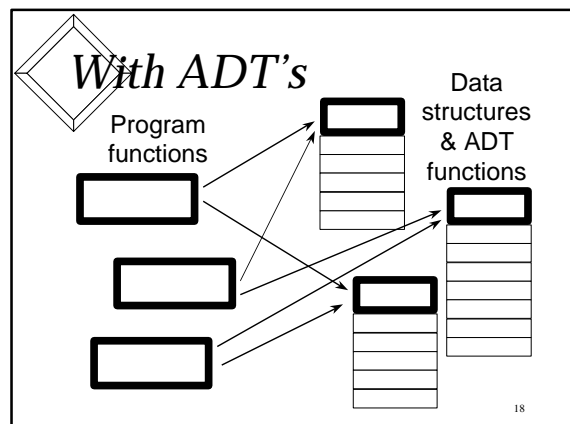
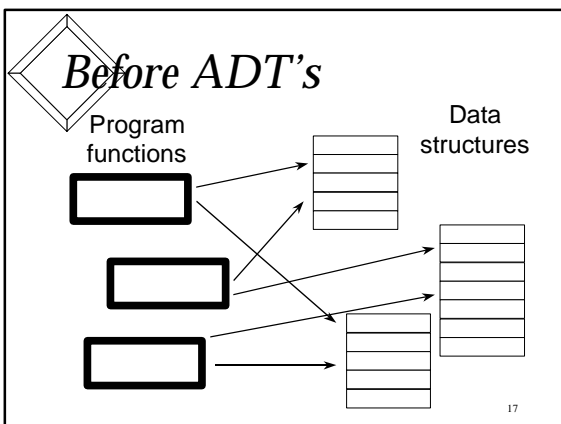
15

Building ADT's

- ❖ Two different programmers (?)
 - ADT implementor
 - ADT user
- ❖ Public interface (".h" file)
- ❖ Hidden implementation (library)

Declaration
versus
definition

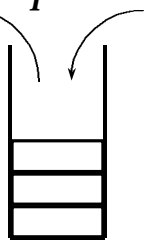
16



CS481 - Object-Oriented Programming

Stack ADT Example

- ❖ Stack of integers
- ❖ Data
 - Set of 0..n integers
- ❖ Operations
 - Create, destroy, push, pop, test empty



19

Stack ADT struct

```
typedef struct
{
    int    max_elem;
    int    current_elem;
    int*   elems;
} stack_struct;

typedef stack_struct * Stack;
typedef const stack_struct *
                    const_Stack;
```

20

"const" Declarations

- ❖ `const int* p;`
 - can't change data (*p)
- ❖ `int* const p;`
 - can't change pointer (p)
- ❖ `const int* const p;`
 - can't change either!

21

Stack Operations

```
extern Stack CreateStack(void);
extern void DestroyStack
(Stack stack);
extern bool IsStackEmpty
(const_Stack stack);
extern void PushStack
(Stack stack, int element);
extern int PopStack(Stack stack);
extern int TopStack
(const_Stack stack);
```

22

Stack Creation

```
extern Stack CreateStack(void)
{
    Stack stack = (Stack) NULL;

    stack = (Stack) calloc
        (1, sizeof(stack_struct));
    stack->max_elem = STACK_SIZE_INCR;
    stack->current_elem = -1;
    stack->elems = (int*)
        malloc(stack->max_elem *
            sizeof(int));
```

23

Stack Creation

```
    assert(stack->elems != NULL);
    assert(IsStackEmpty(stack));

    return stack;
}
```

24

CS481 - Object-Oriented Programming

Stack Push

```
extern void PushStack
    (Stack stack, int element)
{
    if (stack->current_elem >=
        (stack->max_elem - 1))
    {... /* allocate more space */}

    stack->elems
        [++stack->current_elem]
        = element;
    ...
}
```

25

Stack Top Access

```
extern int TopStack
    (const_Stack stack)
{
    assert(!IsStackEmpty(stack));

    return (stack->elems
        [stack->current_elem]);
}
```

26

Destroying a Stack

```
extern void DestroyStack
    (Stack stack)
{
    free(stack->elems);
    free(stack);
}
```

27

Stack Usage

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

stack1 = CreateStack();
if (IsStackEmpty(stack1))
    printf("Stack 1 is empty.\n");

for (ii = 1; ii <= 12; ii++)
    PushStack(stack1, ii);
```

28

