

Structure types and classes

- *Data abstraction*: defining new types
- A *structure type* describes compound data
- Passing structures to functions
- *Object*: model of a thing in the real world
- *Class*: a structure type with operations
- *Encapsulation*: hiding data
- *Constructors* initialize class instances
- Object-oriented design

David Keil Computer Science II 9/04 1

typedef defines a new type

```
typedef float distances;  
distances miles_to_boston = 17.2;
```

- *distances* here is a programmer-defined type identifier that is a synonym for *float*
- Syntax:
typedef *type-spec type-id* ;

David Keil Computer Science II 9/04 2

Enumerated types

```
enum seasons
{ SPRING, SUMMER, FALL, WINTER };
seasons this_season = WINTER;
```

enumerated-type ID

enumerated type *instance of enumerated type* *enumerated-type constant*

Equivalent code:

```
typedef int seasons;
const seasons
    SPRING=0, SUMMER=1, FALL=2, WINTER=3;
seasons this_season = WINTER;
```

- An enumeration is a listing of values

David Keil Computer Science II 9/04 3

Data types

- **Simple**

int	unsigned
float	long
char	short
double	pointers
enumerated types	

- **Compound**

strings	structures
arrays	linked data structures

- A data type is a description of a data item (instance of the type)

David Keil Computer Science II 9/04 4

Enumerated-type constants are integers

```
enum seasons {SPRING, SUMMER, FALL, WINTER};
void display_season_name(seasons season);
void main()
{
    for (int i = SPRING; i <= WINTER; i++)
        display_season_name((seasons)i);
}
[season.cpp]
void display_season_name(seasons season)
// Displays the name associated with season <season>.
{
    switch(season)
    {
        case SPRING:      cout << "Spring ";      break;
        case SUMMER:     cout << "Summer ";      break;
        case FALL:       cout << "Fall ";      break;
        case WINTER:     cout << "Winter ";      break;
    }
}
```

Output: Spring Summer Fall Winter

David Keil Computer Science II 9/04 5

A C-language structure variable declaration

```
struct {
    char id[8];
    double price;
} item;
```

anonymous structure type

structure variable

Sample usage:

```
strcpy(item.id, "X15");
item.price = 150000000;
cout << item.id << endl
     << item.price << endl;
```

Output:

```
X15
150000000
```

member access operator

David Keil Computer Science II 9/04 6

C++ structure type declaration and instance declaration

```

struct items structure type
{
    char id[8];
    float price;
};

void main (void)
{
    items item1, item2; instances of structure type
}

```

item

id	<input type="text"/>	price	<input type="text"/>
----	----------------------	-------	----------------------

David Keil Computer Science II 9/04 7

Initializing and using a structure

```

struct items
{ char id[8]; float price; };

void main() Structure-type instance declaration
{
    items item1 = {"modem23", 49.95};
    strcpy(item1.id, "CPU"); assignment is by
    item1.price = 44.99; member-to-member
    items item2;
    item2 = item1; copy
    cout << item2.id << " "
         << item2.price << endl;
}

```

Output: CPU 44.99

Keil Computer Science II 9/04 8

C-language structure type declaration

```

struct Items
{
    char id[8];
    double price;
};
typedef struct Items items;

void main()
{
    items it = {"Scanner", 79.95};
    printf("%s", it.id);
}

```

structure tag (points to 'Items' in struct Items)

structure type (points to 'items' in typedef struct Items items;)

Output: Scanner

David Keil Computer Science II 9/04 9

Nested structures

```

struct locations { int x,y; };

struct circles
{
    locations loc;
    int radius;
};

void main(void)
{
    circles C = {{200,100},50};
    cout << C.loc.x << C.loc.y << C.radius);
}

```

Nested structure (points to 'locations loc;' in struct circles)

Diagram illustrating the nested structure:

```

circles
├── loc
│   ├── x
│   └── y
└── radius

```

[See *circle.cpp*] **Output: 200 100 50**

Computer Science II 9/04 10

A union stores *one* of its members

```


union quantities
{
    float pounds;
    int units;
};

struct sales_items
{
    char name[20];
    quantities qty;
    bool is_in_units;
    float unit_price;
};

```

Union type

Union data item

quantity pounds or units 

David Keil Computer Science II 9/04 11

Using a union

(assumes declarations of *quantities*, *sales_items*)

```

void main()
{
    sales_items item =
        { "CD", 1, true, 0.02 };
    item_display(item);
}

void item_display(sales_items i)
{
    cout << i.name << " " << i.unit_price
    cout << (i.is_in_units ?
                qty.units : qty.pounds);
}

```

David Keil Computer Science II 9/04 12

Implementing an abstract data type using structure parameters

```

struct items {
    int ID,
        price;
};

void input_item(items& itm)
// Prompts for members of <itm>.
{
    cout << "ID and price: ";
    cin >> itm.id >> itm.price;
}

void show_item(items itm)
// Displays members of <itm>.
{ cout << itm.id << itm.price << endl;}

void main()
{
    items it;
    input_item(it);
    show_item(it);
}

```

A structure type and functions that operate on instances, form an *abstract data type* (ADT), or *class*

David Keil Computer Science II 9/04 13

Passing a structure to a function

```

struct employees { char name[40]; int hours; };
void employee_display(employees e);           [employee.cpp]
void main()
{
    employees emp1 = {"Dale",49}, emp2 = {"Lin",35};
    employee_display(emp1);
    employee_display(emp2);
}

void employee_display(employees e)
{ cout << e.name << " worked " << e.hours << "
  hours.\n"; }

```

Output: Dale worked 49 hours.
Lin worked 35 hours.

- Function *employee_display* implements an operation on any instance of the structure type *employees*.

David Keil Computer Science II 9/04 14

Classes and objects

Class: a compound type defined by data attributes and operations

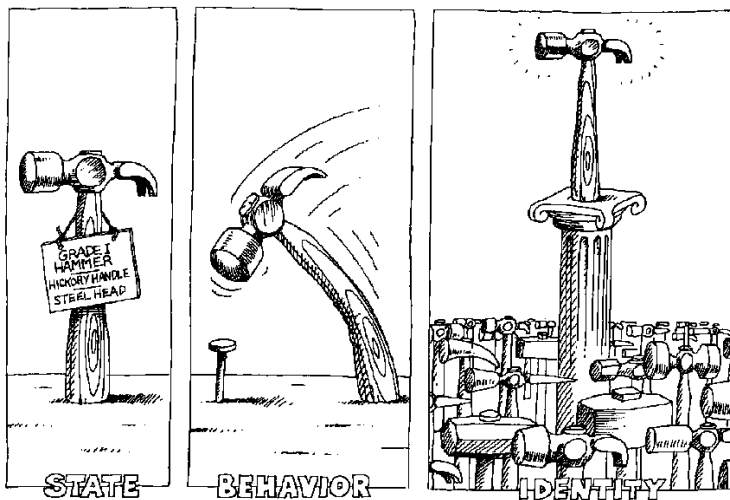
```
struct employees
{
    char name[40];
    int hours;
    void display();
};
```

Object: an instance of a class

```
employees emp1, emp2;
```

David Keil Computer Science II 9/04 15

Objects: state, behavior, identity

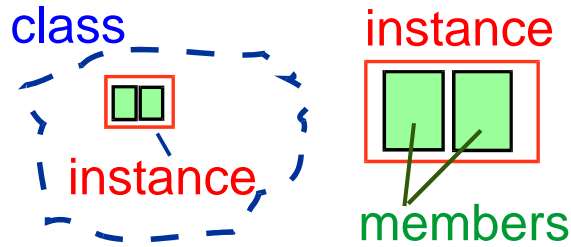


An object has state, exhibits some well-defined behavior, and has a unique identity.

(Booch, *OO Design with Applications*)

David Keil Computer Science II 9/04 16

Classes, instances, members



- A class has 0 or more instances
- A class occupies no memory
- A member is part of an instance; both occupy space

Using a class

```

struct employees
{
    char name[40];
    int hours;
    void input();
    void display();
};

void main()
{
    employees emp;
    emp.input();
    emp.display();
}

void employees::input()
{
    cout << "Enter name, hrs: ";
    cin >> name >> hours;
}

void employees::display()
{
    cout << name << " worked "
        << hours << " hrs.\n";
}
    
```

Sample I/O: Enter name, hrs: Musa 40
Musa worked 40 hours.

The object that calls a member function is an implicit parameter

- It need not be named inside member function to access the object's members
- The object's address is always stored in the standard ID *this* inside every member function
- *Example*: in a member function of class *employees*, *ID* is equivalent to *this->ID*.

David Keil Computer Science II 9/04 19

Encapsulation hides members

- The keyword *class* may replace *struct*.
- With *class*, members are hidden unless declared *public*.

```

class employees
{
public:
    void input();
    void display();
private:
    char name[40];
    int hours;
};

```

hidden

```

void main()
{
    employees emp;
    emp.hours = 40;
    emp.input();
}

```

valid *invalid*

David Keil Computer Science II 9/04 20

A class has an interface and an implementation

- *Interface*: public member function declarations
- *Implementation*: private members and definitions of member functions
- A programmer using a class needs to know only its *interface*.

David Keil Computer Science II 9/04 21

Constructors initialize members

```
class employees
{
public:
    employees();
    employees(char nm[],int hrs);
    void input();
    void display();
private:
    char name[40];
    int hours;
};

employees::employees()
{ nm[0] = hours = 0; }
```

default constructor

constructor with parameters

David Keil Computer Science II 9/04 22

Constructors

- Execute automatically when an object is declared
- Take name of class:
`employees::employees()`
- Have no return value or type
- May take parameters:
`employees emp("Dass",43);`
- May be overloaded:
`employees();`
`employees(char nm[],int hrs);`

David Keil Computer Science II 9/04 23

A class of rational numbers

```
class rationals
{
public:
    rationals(int num,int den) { numer = num; denom = den; }
    rationals plus(rationals term)
    {
        int num = numer * term.denom + denom * term.numer,
            den = denom * term.denom;
        return rationals(num,den);
    }
    void display() { cout << numer << "/" << denom; }
private:
    int numer,denom;
};

void main()
{
    rationals a(1,2), b(1,3), c = a.plus(b);
    a.display(); cout << " + "; b.display();
    cout << " = "; c.display(); cout << endl;
}
```

Output:
1/2 + 1/3 = 5/6

[rational.cpp]

David Keil Computer Science II 9/04 24

A course-registrations class

```
class registrations                                     [coursreg.cpp]
{
public:
    registrations() { student_ID = course_ID = 0; }
    void input()
    {
        cout << "Student ID and course ID: ";
        cin >> student_ID >> course_ID;
    }
    void display() { cout << student_ID << " " << course_ID; }
private:
    int student_ID, course_ID;
};

void main()
{
    registrations reg;
    reg.display();
    reg.input();
    reg.display();
}
```

Sample I/O:

```
0 0
Student ID and course ID: 9421 63252
9421 63252
```

David Keil Computer Science II 9/04 25

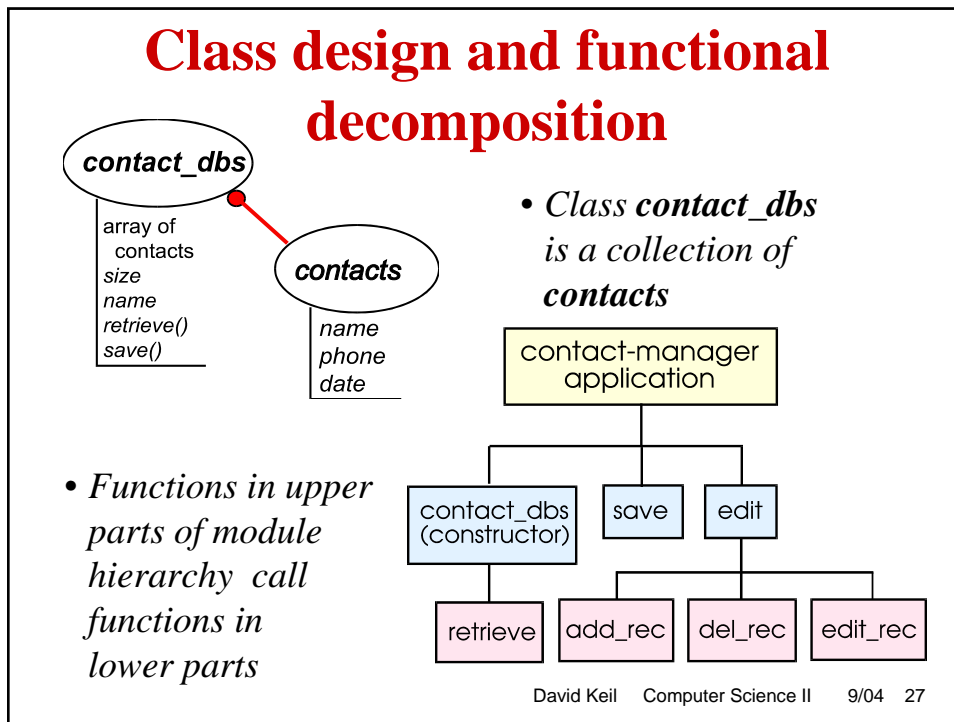
Why use classes?

- To be able to conveniently create instances of a concept
- To associate operations (functions) directly with the relevant objects
- To have these functions share data without passing parameters

Guideline:

A class should have as data members the attributes of *one* object

David Keil Computer Science II 9/04 26



Converting code to use classes

Before

```

void main()
{
[1]  int x,y;
[2]  cout << "Coordinates: ";
[3]  cin >> x >> y;
[4]  cout << "(" << x << "," << y << ")";
}
    
```

After

```

class Coordinates
{
public:
    Coordinates() { x = y = 0; }
    void display() { [4] }
    void input() { [2-3] }
private:
    [1]
}

void main()
{
    Coordinates
    c;
    c.input();
    c.display();
}
    
```

David Keil Computer Science II 9/04 28

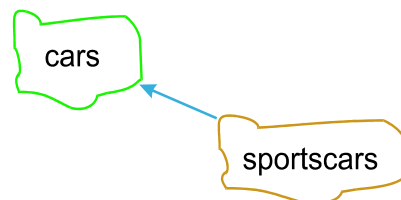
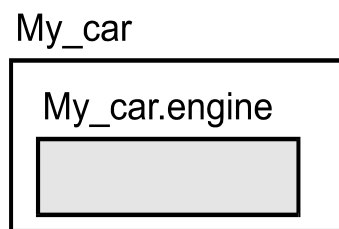
Object-oriented design

- Software engineers today look first for concepts or objects (nouns) in the problem description
- For each important concept, a class is defined
- We consider processes (functions) *after* classes
- Classes may be related by containment or inheritance

David Keil Computer Science II 9/04 29

Containment vs. inheritance

- *Containment* is a *part-of* relationship between *instances* of classes
- *Inheritance* is a *kind-of* relationship between *classes*



David Keil Computer Science II 9/04 30

Three kinds of class

- *Container* (collection): Class whose instances contain instances of another class
- *Derived class*: Class that inherits from another class
- *Base class*: Class from which derived classes may inherit

David Keil Computer Science II 9/04 31

Object-oriented design concepts

- Does each class represent a distinct concept in the problem domain?
- Relationships between classes: containment, inheritance, linking
- Does each class have a clear and easily understood interface?
- Are classes independent enough to be easily modified on their own?

David Keil Computer Science II 9/04 32

Structure types and classes

- Are a form of data abstraction
- Package parts of a concept together
- May be used to declare instances
- Give access to members via dot operator
- May have own functions to operate on instances
- *Constructors* enable us to initialize variables automatically
- *Encapsulation* separates class interface from implementation

David Keil Computer Science II 9/04 33

Problems

Define structure type or class for:

1. Date
2. Time of day
3. Employee
4. Student
5. Book
6. Holiday
7. County fair
8. Bank transaction
9. Sales transaction
10. Sales transaction detail
(1 item on cash-register receipt)

David Keil Computer Science II 9/04 34

Problems

- Define a structure type or class for a graphical image's frame. A frame has a height and a width. Define a Boolean function or method that tells whether the frame is portrait (vs. landscape). Define another function that returns the aspect ratio (vertical \div horizontal).
- Write a program that uses a structure type or class to represent a time of day. Declare, input, and display the times when a task begins and ends.