Dynamic Binding

• Associate a set of functions — “methods” — with each class

\texttt{shape.h:}

```c
#include "point.h"

typedef struct Methods_T {
    Point_T (*where)(Shape_T s);
    void (*move)(Shape_T s, Point_T to);
    void (*draw)(Shape_T s);
} *Methods_T;

typedef struct Shape_T {
    Methods_T methods;
    Point_T center;
} *Shape_T;

typedef struct WHERE { Shape_T s);
extern void Shape_move(Shape_T s, Point_T to);
extern void *Shape_init(T s, Point_T center, Methods_T methods);
```

• \textit{Indirection} invokes appropriate method

\texttt{Shape_T s;
(*s->methods->move)(s, to);}
Methods

• Subclasses have separate interfaces; types inherit fields and methods

  circle.h:

  `#include "shape.h"

typedef struct Circle_T {
    struct Shape_T super;
    float radius;
} *Circle_T;` 

  extern Circle_T Circle_new(float radius);

  square.h:

  `#include "shape.h"

typedef struct Square_T {
    struct Shape_T super;
    float side;
} *Square_T;`

  extern Square_T Square_new(float side);

• Methods appears only in Shape_T

• Implementations initialize methods to class-specific functions
Implementations

**shape.c:**

```c
#include "assert.h"
#include "shape.h"

Point_T Shape_where(Shape_T s) {
    return s->center;
}

void Shape_move(Shape_T s, Point_T to) {
    s->center = to;
    (*s->methods->draw)(s);
}

void *Shape_init(Shape_T s, Point_T center, Methods_T methods) {
    assert(s);
    assert(methods);
    s->methods = methods;
    s->center = center;
    return s;
}
```
Implementations, cont’d

circle.c: (square.c is similar)

```c
#include "assert.h"
#include "circle.h"

static void Circle_draw (Circle_T s) { ... }  
static struct Methods_T methods = {
    Shape_where, Shape_move, Circle_draw
};

Circle_T Circle_new (Point_T center, float radius) {
    Circle_T s = malloc(sizeof *s);
    assert(s);
    s = Shape_init((Shape_T)s, center, &methods);
    s->radius = radius;
    return s;
}
```

- What functions get called?

```c
Circle_T c = Circle_new(Point_new(0, 0), 2.5);
(*c->super.methods->move)((Shape_T)c, Point_new(1, 1));
```

- OOPLs support this methodology — with much less ink and with type safety
Obect-Oriented Programming in Plain C

- **Objects** are pointers to structures

- **Methods** are invoked by “sending a message to the object”

  ```c
  extern void *send(void *obj, char *msg, ...);
  obj identifies the **object** — the “receiver”
  msg names the **method** — an arbitrary string
  send applies the method to the object with the arguments “...”
  default return value is an object
  ```

- **Classes** — object type definitions — are also objects

  **class methods** are sent to **classes**

  ```c
  Point_T p = send(&Point, "new:", 0.0, 0.0);
  Circle_T c = send(&Circle, "new:", p, 2.5);
  ```

  **instance methods** are sent to **objects**

  ```c
  send(c, "move", send(&Point, "new:", 1.0, 1.0));
  p = send(c, "where");
  ```
Example: Texts (a.k.a. Strings)

- Interface: `text.h`

```c
#include <stdio.h>
#include "object.h"
#define T Text_T
typedef struct T {
    struct Object_T super; /* superclass */
    ...
} *T;
extern struct Class_T Text, metaText;
/*
Class Methods
~~~~~~~~~~~~~~
T new:(char *text);
returns a new text and initialized to a copy of text.

Instance Methods
~~~~~~~~~~~~~~~~
...

Responsibilities
~~~~~~~~~~~~~~~~
It is a checked runtime error to pass a NULL text or char* to
any method in this interface.
*/
```
Reading Words

• A class method “manufactures” instances of its class

• `getword`: returns the next word from the standard input as a `Text_T`

```c
Text_T getword(void) {
    char buf[100], *s;
    int c;

    while ((c = getchar()) != EOF && !isalpha(c))
        ;
    for (s = buf; c != EOF && isalpha(c); c = getchar())
        if (s - buf < sizeof buf - 1)
            *s++ = c;
    *s = 0;
    return s > buf ? send(&Text, "new:", buf) : 0;
}
```
Text Interface

- Instance methods manipulate objects — specific instances of classes

  
  int compare:(T text);
  returns <0, 0, >0 if the receiver’s text is less than, equal to, or greater than text’s text, respectively.

  T free(void);
  deallocates the receiver and returns NULL.

  char *get(void);
  returns a pointer to the receiver’s text.

  unsigned hash(void);
  returns a hash number based on the receiver’s text.

  int length(void);
  returns the number of characters in the receiver’s text.

  T fprint(FILE *fp);
  prints the text on the file fp. If fp is NULL, print it on stderr and returns the receiver.

  T set:(char *text);
  deallocates, reinitializes, and returns a new text

- If s and t are Text_Ts

  send(t, "set:", "hello world");
  (int)send(s, "compare:", t)
Access and Encapsulation in C++

• Class can define public and private members (data or functions)

```cpp
class intArray{
    public:
        void init();
        void setSize(size_t value);
        size_t getSize();
        void setElem(size_t index, int value);
        int getElem(size_t index);
    private:
        int *elems;
        size_t numElems;
}

int intArray::getElem(size_t index) {
    if (index >+ numElems) error(“bad index”);
    return elems[index];
}
```

• What corresponds to private function in C?
• Difference between structs and classes in C++?
A Stack of Integers in C++

class intStack {
    public:
        void init();
        void push (int value);
        int pop();
        void isEmpty();

    private:
        intArray items;
        size_t depth;
}

• What if I want to sum the elements of a Stack?

class intStack {
    public:
        ...
        void goFirst();
        void goNext();
        int getCur();
        int curIsValid();

    private:
        ...
        size_t curItems;
}
Cooperating Classes

class intStackIter {
    public:
        void init();
        void push (int value);
        int setIter(IntStack *stack);
        void goFirst();
        void goNext();
        int getCur();
        int curIsValid();
    private:
        IntStack *iterStack;
        size_t curItem;
}

main {
    IntStack stack;
    IntStackIter iter;
    int sum;
    // ...
    stack.init(); iter.init();
    // ...
    sum = 0; iter.setIter(&stack);
    for (iter.goFirst(); iter.curIsValid(); iter.goNext())
        sum += iter.getCur();
}
How does Iterator Access Stack Internals?

```cpp
int IterStack::getCur() {
    if (!curIsValid()) error("no current item");
    return iterStack->items.getElem(curItem);
}
```

- Problem?
- Solution:
  ```cpp
class intStack {
    friend class IntIterStack;

    public:
        ... // rest all the same
  }
```
  - Friends can access all
  - Friendship is a one-way street
  - A friend of my friend is no friend of mine