

Chapter 26

Program Design

Design Goals

- Reliability
- Economy
- Ease of Use

Design Factors

- Simplicity
- Information Hiding
- Expandability
- Testable
- Reusability / generality

Design Principles

1. Think – Then code!
2. Be Lazy (aka. Efficient)

Procedure Design

- Procedures should do one thing well.
- Interface should be as simple as possible.
- Global interactions should be as limited as possible.
- Details are hidden.

Modules

- Organize (Disorganization = government)
- Minimal connections between modules
- Consistency.

Object Design

- Design a generic base class
(I.E. Locomotive)
- Specialize it in the derived classes
(Steam Locomotive, Diesel, Electric)

The Linked List Problem

C Language Solutions

- 1) Create 47 different structures and an insert/delete function for each. (Bad solution).

`0 insert_msg / remove_msg`

`insert_run / remove_run`

`insert_kbd / remove_kdb`

`insert_idle / remove_idle`

(If you really want to be rotten, use as many different words for "insert" and "remove" as you can when you name your functions.)

"C" Linked List Solution

- Define a generic header

```
struct list_head {  
    struct list_head *next, *prev;  
}
```

- Use this at the beginning of all your structures.

```
struct run_list {  
    struct list_head head;  
    // Run list stuff  
};
```

"C" Solution

- Items can now be inserted or removed using generic functions and casting.

```
insert_node(  
    (struct list_head*)run_list,  
    (struct list_head*)new_run);
```

- Works, but is a "clever" trick
- This is a "C" implementation of a class derivation mechanism

C++ Solution

```
class list {  
    private:  
        list* next, prev;  
        // ...  
};
```

```
class pending_message_node: public_list {  
    // .. message stuff  
};
```

Not well designed.

Templates to the rescue

```
template class list<typename data> {  
    private:  
        list* next, prev;  
    public:  
        data node;  
};
```

Better yet, let someone else write the list functions. (They are part of the STL.)

Callbacks

Command table:

```
struct cmd_info {
    const char* command;
    void (*function)();
}[] cmd_table[] = {
    {"delete", do_delete},
    {"search", do_search},
    {"exit", do_exit},
    ....
};
```

V.S.

Event Registration

```
keyboard_module::register_command("exit", &do_exit);
```

C++ Couples Interface and Implementation

phone_book.h

```
class phone_book {
    public:
        // (Interface function)
        void store(const std::string &name, ....);

    private:
        // (Implementation functions)
        void internal_consistency_check();
        void save_internal_state();
};
```

Decoupled Implementation / Interface

phone_book.h

```
// No information about this class is in this file
// except that it's some sort of class
class phone_book_implementation;

class phone_book {
public:
    // (Interface function)
    void store(const std::string &name, ....);

private:
    phone_book_implementation*
        the_implementation;

};
```