

Declaring classes in C++

```
#include <iostream.h>
#include <string.h>

class book {
public:
    book(char *, char *, int);
    void show_book(void);
private:
    char title[64];
    char author[64];
    int pages;
};

class catalog_card : public book {
public:
    catalog_card(char *, char *, int, char *, int);
    void show_card(void);
private:
    char catalog[64];
    int checked_out;    // 1 if checked out, otherwise 0
};
```

Filling in the functions

Constructors

```
book::book(char *title, char *author, int pages)
{
    strcpy(book::title, title);
    strcpy(book::author, author);
    book::pages = pages;
}
```

```
catalog_card::catalog_card(char *title, char *author,
int pages, char * catalog, int checked_out) :
book(title, author, pages)
{
    strcpy(catalog_card::catalog, catalog);
    catalog_card::checked_out = checked_out;
}
```

Filling in the functions

```
void book::show_book(void)
{
    cout << "Title: " << title << endl;
    cout << "Author: " << author << endl;
    cout << "Pages: " << pages << endl;
}

void catalog_card::show_card(void)
{
    show_book();
    cout << "Catalog: " << catalog << endl;
    if (checked_out)
        cout << "Status: Checked out" << endl;
    else
        cout << "Status: Available" << endl;
}
```

Multiple inheritance in C++

Declaring the two base classes

```
class computer_screen {
public:
    computer_screen(char *, long, int, int);
    void show_screen(void);
private:
    char type[32];
    long colors;
    int x_resolution;
    int y_resolution;
};
```

```
class mother_board {
public:
    mother_board(int, int, int);
    void show_mother_board(void);
private:
    int processor;
    int speed;
    int RAM;
};
```

Multiple inheritance in C++

Using both base classes in a new derived class

```
class computer: public computer_screen, public
mother_board
{
    public:
        computer(char *, int, float,
                char*, long, int,
                int, int, int, int);
        void show_computer(void);
    private:
        char name[64];
        int hard_disk; //size of
        float floppy;
};
```

Creating objects at run-time, & allowing for shadowing

First: Let's do it obvious, but wrong

```
#include <iostream.h>
class railroad_car {
    public:
    void display_short_name() {cout << "rrc";}
};

class box_car: public railroad_car {
    public:
    void display_short_name() {cout << "box";}
};

//..Similar definitions of tank_car, engine, and
caboose go here
```

```

//Define railroad car array
railroad_car *train[100];
main() {
    int car_count, type_code, n;

    for (car_count = 0; cin >> type_code; ++car_count)
        if (type_code == 0) train[car_count] = new engine;
        else if (type_code == 1) train[car_count] = new
box_car;
        else if (type_code == 2) train[car_count] = new
tank_car;
        else if (type_code == 3) train[car_count] = new
caboose;

    cout << "There are " << car_count
        << " cars in the array." << endl;

    for (n=0; n < car_count; ++n) {
        train[n] -> display_short_name();
        cout << endl;
    }
}

```

Running the *wrong* version

Input: 0 1 1 2 3

Output:

There are 5 cars in the array.

rrc

rrc

rrc

rrc

rrc

Why didn't each type of car do its own `display_short_name`?

- Because C++ can't determine the type of each `railroad_car` at compile time, you must tell it to look them up at run time

Fixing the class declarations

```
#include <iostream.h>
class railroad_car {
    public:
        virtual void display_short_name() {cout << "rrc";}
};
class box_car: public railroad_car {
    public:
        virtual void display_short_name() {cout << "box";}
};
//..Similar definitions of tank_car, engine, and
caboose go here
```

Input: 0 1 1 2 3

Output:

There are 5 cars in the array.

eng

box

box

tnk

cab

If you never expect to need the base virtual function...

```
#include <iostream.h>
class railroad_car {
    public:
        virtual void display_short_name() = 0
};
class box_car: public railroad_car {
    public:
        virtual void display_short_name() {cout << "box";}
};
//..Similar definitions of tank_car, engine, and
caboose go here
```

Overloading functions

```
#include <iostream.h>
```

```
int add_values(int a, int b)
{
    return (a + b);
}
```

```
int add_values(int a, int b, int c)
{
    return(a + b + c);
}
```

```
void main(void)
{cout << "200 + 801=" << add_values (200, 801) << endl;
cout << "100+201+700=" << add_values(100,201,700) <<
endl;
}
```

Overloading operators

```
#include <iostream.h>
#include <string.h>

class string {
public:
    string(char *); //Constructor
    void operator +(char *);
    void show_string(void);
private:
    char data[256];
};

string::string(char *str)
{strcpy(data, str);}

void string::operator +(char *str)
{strcat(data, str);}

void string::show_string(void)
{cout << data << endl;}
```

```
void main(void)
{
    string title("CS 2390");
    string name("Modeling and Design");
    title.show_string();
    title + name;
    title.show_string();
}
```

Class templates

You can even parameterize *types*

```
class array{
public:
    array(int size);
    long sum(void);
    int average_value(void);
    void show_array(void);
    int add_value(int);
private:
    int *data;
    int size;
    int index;
};

long array::sum(void)
{
    long sum=0; int i;
    for (i=0; i< index; i++)
        sum = sum + data[i]; //sum += data[i];
    return(sum);
}
```

Class templates

```
template<class T, class T1>
class array{
public:
    array(int size);
    T1 sum(void);
    T average_value(void);
    void show_array(void);
    int add_value(T);
private:
    T *data;
    int size;
    int index;};

template<class T, class T1>
T1 array<T, T1>:sum(void)
{
    T1 sum = 0; int i;
    for (i = 0; i < index; i++)
        sum += data[i];
    return(sum);}
```