name = name;

return 1;

grade = in_grade;

Strogy (email, in.email);

email = new char [strlen (in.email)] + 1;

Strogy (name, in.name);

name = new char [strlen (in.name)] + 1;

int *steve = (Student *) copy (student, "copy");

#include "students.h"
cpl

enroll = enroll;

name = null;

grade = \phi;

delc[k] enroll;

delc[k] name;

Student :: enroll ();

grade = \phi;

cmnd = null;

name = null;

Student :: enroll (cplt;
```c
return f;

grade = grade - 1;

strcpy (emain, eman - 1);
emain = new char [strlen (emain) + 1];
strcpy (new_name, new_name - 1);
new_name = new char [strlen (new_name) + 1];
float grade - 1;

// Code for storing student information
in Student: store (char *name, char *email, float grade);
```
```
if (grade == failure) {
    return false;
}

int student::check_grade (float newgrade) {
    return (newgrade > grade); // compare (char *name)
}

int student::compare (char *name) {
    return 0; // comparison function
}

if (strcmp (name, surname) == 0) {
    return 0; // return false
}

if (strcmp (name, surname) != 0) {
    return 1; // return true
}
```

class roster //List
{
    public:
        roster();
    ~roster();
    int enroll (student &);
    int drop (student &);
    int drop (char * name);
    int display_all ( );
    int find (char * name);
        : int change_grade (char * name, float grade);

    private:
        node * head;
        node * tail;
};
```cpp
return int(current)

current = current next

break

the student

count < a-student change-grade

if (current < a-student compare (same))

while (current)

node = current head

float new grade

int restored: change-grade (char * name,

...
while (current != null) {
    if (current->student->name == name) {
        return (current->student).change-grade(count); // change grade
    }
    current = current->next;
}
return true;
}
```c
Head = current;
delete head;

current = current->next;
} else // many nodes... but delete the first
{

head = tail = NULL;
delete head;
}

if (head == tail) // 1 node
{
if (head->a->student. compare(name-dropped))

node & previous = NULL;
node * current = head;

int result :: deep (chain * name-dropped)

```
else {
    while (current != nullptr) {
        if (strcmp(name, current->name) == 0) {
            previous = current;
            current = current->next;
        }
    }
    if (current) { // found a match
        previous->next = current->next;
        if (current == tail) { // Last node:
            tail = previous;
        }
        delete current;
    } else return false;
}
return 1;