NGÔN NGỮ LẬP TRÌNH C++

Chương 3: Kiểu dữ liệu cấu trúc

Nội dung

- Định nghĩa cấu trúc
- Các thao tác trên cấu trúc
- Con trỏ cấu trúc và mảng cấu trúc
- Các kiểu dữ liệu trừu tượng

Cấu trúc trong C++

- Kiểu dữ liệu có cấu trúc được dùng khi ta cần nhóm một số biến dữ liệu luôn đi kèm với nhau. Khi đó, việc xử lí trên một nhóm các biến được thực hiện như trên các biến cơ bản thông thường
- Khai báo cấu trúc

```
struct <Tên cấu trúc>{
        <Kiểu dữ liệu 1> <Tên thuộc tính 1>;
        <Kiểu dữ liệu 2> <Tên thuộc tính 2>;
        …
        <Kiểu dữ liệu n> <Tên thuộc tính n>;
};
```

Thao tác trên cấu trúc trong C++

- Khai báo

```
struct Employee{
    char name[20]; // Tên nhân viên
    int age; // Tuổi nhân viên
    char role[20]; // Chức vụ của nhân viên
    float salary; // Lương của nhân viên
};
```

- Sử dụng

```
Employee myEmployee;
```

Thao tác trên cấu trúc trong C++

- Khai báo với typedef

```
typedef struct Employee{
        char name[20]; // Tên nhân viên
        int age; // Tuổi nhân viên
        char role[20]; // Chức vụ của nhân viên
        float salary; // Lương của nhân viên
} Employee;
```

Sử dụng

- Employee myEmployee;

Thao tác trên cấu trúc trong C++

- Khởi tạo giá trị khi khai báo
- Employee myEmployee = {
- “Nguyen Van A”, 27, “Nhan vien”, 300f
- };
- Truy cập đến thuộc tính
- myEmployee.name

Con trỏ cấu trúc

- Con trỏ cấu trúc là một con trỏ trỏ đến địa chỉ của một biến có kiểu cấu trúc. Cách khai báo và sử dụng con trỏ cấu trúc được thực hiện như con trỏ thông thường
- Khai báo

  struct <Tên cấu trúc> *<Tên biến>;
- Gán địa chỉ con trỏ cho cấu trúc

  <Tên biến con trỏ> = &<Tên biến thường>;
- Ví dụ

```
Employee *ptrEmployee, myEmployee;
ptrEmployee = &myEmployee;
```

Con trỏ cấu trúc

- Cấp phát bộ nhớ động cho con trỏ cấu trúc

  // Cấp phát bộ nhớ

  <Tên biến con trỏ> = new <Kiểu cấu trúc>;

  delete <Tên biến con trỏ>;

- Ví dụ

  Employee *ptrEmployee = new Employee;

  delete ptrEmployee;

-

Con trỏ cấu trúc

- Truy cập thuộc tính con trỏ cấu trúc
  - Cách 1: <Tên biến con trỏ>-><Tên thuộc tính>;
  - Cách 2: (*<Tên biến con trỏ>).<Tên thuộc tính>;
- Ví dụ

```
Employee *ptrEmployee = new Employee;
cin >> ptrEmployee->name;
hoặc: cin >> (*ptrEmployee).name;
```

Mảng cấu trúc

- Khi cần xử lí nhiều đối tượng có dùng kiểu dữ liệu cấu trúc, ta có thể sử dụng mảng các cấu trúc
- Khai báo mảng tĩnh

  <Tên kiểu cấu trúc> <Tên biến mảng>[<Số phần tử mảng>];
- Ví dụ

  Employee employees[10];

Mảng cấu trúc

- Khai báo mảng động

  <Tên kiểu cấu trúc> *<Tên biến>;

- Ví dụ

  Employee *employees;

  Employee *employees = new Employee[10];

- Truy cập đến phần tử

  employees[i].name;

-

Bài tập

- Cho số N. Danh sách N sinh viên gồm các thông tin: Mã SV, Họ Tên, Lớp, Điểm TB Môn THCS2, Điểm TB Môn C++. Thực hiện các nhiệm vụ sau:
- Nhập N và các thông tin cho N sinh viên từ bàn phím.
- In ra màn hình danh sách các sinh viên có điểm môn C++ >= 7.0
- Sắp xếp và in danh sách sinh viên với họ tên theo thứ tự từ điển.
- Sắp xếp và danh sách sinh viên theo điểm trung bình chung các môn giảm dần.

Bài tập

- Cho số N, M. Danh sách N môn học bao gồm các thông tin: Mã MH, Tên MH, Số TC, Điểm CC, Điểm KT, Điểm Thi; M sinh viên gồm các thông tin: Mã SV, Họ Tên, Lớp. Mỗi sinh viên sẽ học một số môn nhất định. Thực hiện các nhiệm vụ sau:
- Nhập N, M và các thông tin cho N môn học, M sinh viên từ bàn phím. Mỗi sinh viên nhập K môn học.
- Tính điểm trung bình các môn học của từng sinh viên theo trọng số Điểm CC 10%, Điểm KT 20%, Điểm Thi 70% .
- In ra danh sách sinh viên có điểm TBC >= 7.0

Các kiểu dữ liệu trừu tượng

- Ngăn xếp
- Hàng đợi
- Danh sách liên kết

Ngăn xếp

- Ngăn xếp (stack) là một kiểu danh sách cho phép thêm và bớt các phần tử ở một đầu danh sách, gọi là đỉnh của ngăn xếp.
- Ngăn xếp hoạt động theo nguyên lí: LIFO: Last in first out

Ngăn xếp

- Định nghĩa cấu trúc ngăn xếp:
  - Danh sách các phần tử của ngăn xếp
  - Vị trí đỉnh của ngăn xếp

```
typedef SIZE 100;
typedef struct {
int top; // Vị trí của đỉnh
int nodes[SIZE]; // Danh sách các phần tử
} Stack;
```

Ngăn xếp

- Định nghĩa cấu trúc ngăn xếp:
  - Danh sách các phần tử của ngăn xếp
  - Vị trí đỉnh của ngăn xếp

```
typedef struct {
int top; // Vị trí của đỉnh
int *nodes; // Danh sách các phần tử
} Stack;
```

Ngăn xếp

- Các thao tác cơ bản trên ngăn xếp
  - Thêm một phần tử mới vào đỉnh ngăn xếp, gọi là push.
  - Lấy ra một phần tử từ đỉnh ngăn xếp, gọi là pop.

Ngăn xếp

- Thêm một phần tử mới vào đỉnh ngăn xếp, gọi là push.
  - Số phần tử trong ngăn xếp cũ là (top+1). Do đó, ta cấp phát một vùng nhớ mới để lưu được (top+1+1) = (top+2) phần tử.
  - Sao chép (top+1) phần tử cũ sang vùng mới. Nếu danh sách ban đầu rỗng (top = -1) thì không cần thực hiện bước này.
  - Thêm phần tử mới vào cuối vùng nhớ mới
  - Giải phóng vùng nhớ của danh sách cũ
  - Cho danh sách nodes trỏ vào vùng nhớ mới.

Ngăn xếp

```
void push(Stack *stack, int node){
        int *tmpNodes = new int[stack->top + 2];// Cấp phát vùng nhớ mới
        stack->top ++; // Tăng chỉ số của node đỉnh
        for(int i=0; i<stack->top; i++) // Sao chép sang vùng nhớ mới
        tmpNodes[i] = stack->nodes[i];
        tmpNodes[stack->top] = node; // Thêm node mới vào đỉnh
        delete [] stack->nodes; // Giải phóng vùng nhớ cũ
        stack->nodes = tmpNodes; // Trỏ vào vùng nhớ mới
        return;
}
```

Ngăn xếp

- Lấy ra một phần tử từ đỉnh ngăn xếp, gọi là pop.
  - Kiểm tra xem ngăn xếp có rỗng (top = -1) hay không. Nếu không rỗng thì thực hiện các bước tiếp theo.
  - Lấy phần tử ở đỉnh ngăn xếp ra
  - Cấp phát một vùng nhớ mới có (top+1) -1 = top phần tử
  - Sao chép top phần tử từ danh sách cũ sang vùng nhớ mới (trừ phần tử ở đỉnh). Giải phóng vùng nhớ cũ
  - Cho con trỏ danh sách trỏ vào vùng nhớ mới. Trả về giá trị phần tử ở đỉnh đã lấy ra.

Ngăn xếp

```
int pop(Stack *stack){
    if (stack->top < 0){ // Kiểm tra ngăn xếp rỗng cout << “Stack is empty!” << endl;
    return 0;
        }
    int result = stack->nodes[stack->top];// Lưu giữ giá trị đỉnh
    int *tmpNodes = new int[stack->top];// Cấp phát vùng nhớ mới
    for(int i=0; i<stack->top; i++) // Sao chép sang vùng nhớ mới
    tmpNodes[i] = stack->nodes[i];
    stack->top --; // Giảm chỉ số của node đỉnh
    delete [] stack->nodes; // Giải phóng vùng nhớ cũ
    stack->nodes = tmpNodes; // Trỏ vào vùng nhớ mới
    return result; // Trả về giá trị node đỉnh
}
```

Áp dụng

Anh/chị hãy cài đặt chương trình C++ dùng ngăn xếp để đảo ngược một xâu kí tự được nhập vào từ bàn phím.

Hàng đợi

- Hàng đợi (queue) cũng là một cấu trúc tuyến tính các phần tử.
- Trong đó, các phần tử luôn được thêm vào ở một đầu, gọi là đầu cuối hàng đợi, và việc lấy ra các phần tử luôn được thực hiện ở đầu còn lại, gọi là đầu mặt của hàng đợi.
- Hàng đợi hoạt động theo nguyên lí: FIFO – first in first out

Hàng đợi

- Định nghĩa cấu trúc hàng đợi
  - Một danh sách các phần tử có mặt trong hàng đợi.
  - Chỉ số của phần tử đứng đầu của danh sách (front). Chỉ số phần tử cuối của danh sách (rear).
- Nếu dùng cấu trúc tĩnh để định nghĩa, hàng đợi có cấu trúc như sau:

```
typedef SIZE 100;
typedef struct {
int front, rear; // Vị trí của đỉnh đầu, đỉnh cuối
int nodes[SIZE]; // Danh sách các phần tử
} Queue;
```

Hàng đợi

- Nếu dùng bộ nhớ động để lưu giữ hàng đợi, thì phần tử front luôn là phần tử thứ 0 của danh sách. Và rear sẽ bằng độ dài danh sách trừ đi 1.

```
typedef struct {
int front, rear; // Vị trí của đỉnh đầu, đỉnh cuối
int *nodes; // Danh sách các phần tử
} Queue;
```

Hàng đợi

- Thao tác trên hàng đợi:
  - Thêm một phần tử vào cuối hàng đợi
  - Lấy một phần tử ở vị trí đầu của hàng đợi

Hàng đợi

- Thêm một phần tử vào cuối hàng đợi:

Thao tác thêm một phần tử vào cuối hàng đợi với bộ nhớ động được thực hiện tương tự với ngăn xếp.

Hàng đợi

- Lấy một phần tử ở vị trí đầu của hàng đợi
  - Kiểm tra tính rỗng (front = rear = -1) của hàng đợi. Nếu không rỗng mới thực hiện tiếp
  - Lấy phần tử nodes[0] ra.
  - Sao chép danh sách còn lại sang vùng nhớ mới
  - Giải phóng vùng nhớ cũ
  - Đưa danh sách trỏ vào vùng nhớ mới
  - Trả về giá trị phần tử lấy ra

Danh sách liên kết

- Danh sách liên kết là một kiểu dữ liệu bao gồm một dãy các phần tử có thứ tự, các phần tử có cùng cấu trúc dữ liệu, ngoại trừ node đầu tiên của danh sách là node lưu thông tin về danh sách.
- Có hai loại danh sách liên kết:
  - Danh sách liên kết đơn: mỗi node có một con trỏ trỏ đến node tiếp theo trong danh sách
  - Danh sách liên kết kép: mỗi node có hai con trỏ, một trỏ vào node trước, một trỏ vào node tiếp theo trong danh sách.

Danh sách liên kết

- Định nghĩa danh sách đơn
- Mỗi node của danh sách đơn chứa dữ liệu của nó, đồng thời trỏ đến node tiếp theo trong danh sách, cấu trúc một node như sau:

```
struct simple{
Employee employee; // Dữ liệu của node có kiểu Employee
struct simple *next; // Trỏ đến node kế tiếp
};
typedef struct simple SimpleNode;
```

Danh sách liên kết

- Node đầu của danh sách đơn có cấu trúc riêng, nó không chứa dữ liệu như node thường mà chứa các thông tin:
  - Số lượng node trong danh sách (không kể bản thân nó – node đầu)
  - Con trỏ đến node đầu tiên của danh sách
  - Con trỏ đến node cuối cùng của danh sách
- Do vậy, cấu trúc node đầu của danh sách đơn là:

```
typedef struct{
int nodeNumber; // Số lượng các node
SimpleNode *front, *rear;// Trỏ đến node đầu và cuối danh
sách } SimpleHeader;
```

Danh sách liên kết

- Các thao tác cơ bản trên danh sách đơn bao gồm:
  - Chèn thêm một node vào vị trí thứ n trong danh sách
  - Loại ra một node ở vị trí thứ n trong danh sách

Danh sách liên kết

- Việc chèn thêm một node vào vị trí thứ n trong danh sách được thực hiện theo các bước:
  - Nếu n<=0, chèn vào đầu. Nếu n>số phần tử của danh sách, chèn vào cuối. Trường hợp còn lại, chèn vào giữa.
  - Tìm node thứ n: giữ vết của hai node thứ n-1 và thứ n.
  - Tạo một node mới: cho node thứ n-1 trỏ tiếp vào node mới và node mới trỏ tiếp vào node thứ n.

Danh sách liên kết

- Việc xoá một node ở vị trí thứ n trong danh sách được thực hiện theo các bước:
  - Nếu n<0 hoặc n>số phần tử của danh sách, không xoá node nào.
  - Tìm node thứ n: giữ vết của ba node thứ n-1, thứ n và thứ n+1.
  - Cho node thứ n-1 trỏ tiếp vào node thứ n+1, xoá con trỏ của node thứ n.
  - Trả về node thứ n.