THỰC HÀNH KỸ THUẬT LẬP TRÌNH

ĐỒ ÁN MÔN HỌC #1

REPORT

Tài liệu này mô tả nội dung đồ án môn học cho môn học Thực hành Kĩ Thuật Lập Trình.

Nhóm 1

23120207 – Nguyễn Bảo An

23120190 – Nguyễn Lê Thế Vinh

Khoa Công nghệ Thông tin
Đại học Khoa học Tự nhiên TP HCM
Tháng Tháng 4-24

MỤC LỤC

1

Tổng quan

Thông tin nhóm

MSSV	Họ tên
23120207	Nguyễn Bảo An
23120190	Nguyễn Lê Thế Vinh

Thông tin đồ án

Đây là Project 1 – Bài tập Thực hành nhóm của môn Thực hành Kĩ thuật lập trình.

Giáo viên hướng dẫn: Thầy Trần Huy Quang.

Trong Project này, sẽ bao gồm một chuỗi các bài tập liên quan đến kiến thức con trỏ, cấp phát động, mảng động để xử lý mảng chứa một dãy số trong tập tin **input.txt** . Trong chương trình, user có thể thực hiện những lệnh, bao gồm các lệnh xử lý dữ liệu và các lệnh chức năng.

2

Nội dung đồ án

2.1

Mô tả các lệnh và chức năng của chương trình

Chương trình bao gồm 2 nhóm lệnh chính: các lệnh xử lý dữ liệu và các lệnh chức năng.

Các lệnh xử lý dữ liệu

1. **delete <pos>**
   Khi user nhập vào lệnh delete <pos>, pos ở đây chính là một số nguyên chỉ vị trí trong dãy mà người dùng muốn xoá ($1 \leq pos \leq n$). Khi nhập vào lệnh này, màn hình console sẽ hiển thị một dãy số mới không bao gồm số mà user đã yêu cầu xoá.

   Minh hoạ:

   ```
   12 78 1 43 90 56 78
   > delete 3
   12 78 43 90 56 78
   ```

2. **insert <pos> <val>**
   Khi user nhập vào lệnh insert <pos> <val>, pos ở đây cũng là một số nguyên chỉ vị trí trong dãy mà người dùng muốn thêm ($1 \leq pos \leq n$), val là giá trị nguyên mà user muốn thêm vào dãy. Khi nhập vào lệnh này, màn hình console sẽ hiển thị một dãy số mới bao gồm giá trị val mà người dùng muốn thêm ngay tại vị trí pos.
   Minh hoạ:

   ```
   12 78 43 90 56 78
   > insert 4 5
   12 78 43 5 90 56 78
   ```

Các lệnh chức năng

1. **undo**

   Khi user nhập vào lệnh này, chương trình sẽ phục hồi lại lệnh gần nhất và màn hình console sẽ hiển thị dãy số sau khi thực hiện phục hồi.

   Minh hoạ:

```
12 78 43 90 56 78
> insert 4 5
12 78 43 5 90 56 78
> undo
12 78 43 90 56 78
```

   Khi đã phục hồi đến bước đầu tiên (dãy số ban đầu), chương trình không thể phục hồi được nữa.

   Minh hoạ:

```
12 78 1 43 90 56 78
> undo
No more commands to undo...
12 78 1 43 90 56 78
```

2. **redo**

   Khi user nhập vào lệnh này, chương trình sẽ lặp lại lệnh đã phục hồi gần nhất và màn hình console sẽ hiển thị dãy số khi thực hiện lặp lại lệnh đã phục hồi.

   Minh hoạ:

```
12 78 1 43 90 56 78
> insert 3 5
12 78 5 1 43 90 56 78
> undo
12 78 1 43 90 56 78
> redo
12 78 5 1 43 90 56 78
```

3. **save**

   Khi user nhập vào lệnh này, chương trình sẽ lưu lại dãy số hiện tại vào tập tin **output.txt** trên 1 dòng, màn hình console sẽ hiển thị dòng Numbers has been stored.

   Minh hoạ:

```
12 78 5 1 43 90 56 78
> save
Numbers have been stored...
```

4. **reset**

   Khi user nhập vào lệnh này, chương trình sẽ khởi tạo lại phiên làm việc. Nghĩa là quay về với dãy số ban đầu được đọc từ **input.txt** . Màn hình console sẽ hiển thị dãy số ban đầu.

Minh hoạ:

```
12 78 5 1 43 90 56 78
> insert 2 5
12 5 78 5 1 43 90 56 78
> reset

12 78 1 43 90 56 78
```

5. **quit**

Khi user nhập vào lệnh này, đồng nghĩa sẽ có chức năng kết thúc chương trình.

Minh hoạ:

```
12 78 1 43 90 56 78
> quit

Program ended with exit code: 0
```

2.2
Các bước thực hiện trong chương trình

Ý tưởng

Ý tưởng thực hiện của chương trình này bao gồm 2 phần chính: một mảng 1 chiều để lưu dãy số và 1 mảng 2 chiều để lưu lịch sử thực hiện chương trình. Bên cạnh đó là các hàm **Delete, Insert, Undo, Redo, Save, Reset** để thực hiện các lệnh của chương trình.

Đầu tiên, chúng em sẽ thực hiện đọc tập tin **input.txt** để có được dãy số ban đầu, và lưu nó vào một mảng 1 chiều được cấp phát động bởi 1 biến con trỏ cấp 1. Sau đó, cho thực hiện một vòng lập thực hiện đọc lệnh từ người dùng. Người dùng yêu cầu và thực hiện các hàm tương ứng **Delete, Insert, ...**

Sau mỗi lần thực hiện xong 1 lệnh, dãy số sau khi thực thi lệnh sẽ được lưu vào trong một mảng 2 chiều được cấp phát động bởi 1 biến con trỏ cấp 2. Điều này dễ dàng cho việc thực hiện các lệnh chức năng **Undo, Redo, Reset**. **Undo** thì sẽ thực hiện gán biến con trỏ mảng chính bằng con trỏ liền trước trong mảng 2 chiều. **Redo** thì sẽ thực hiện gán biến con trỏ mảng chính bằng con trỏ liền sau trong mảng 2 chiều. **Reset** thì sẽ giải phóng bộ nhớ của cả 2 vùng nhớ và đọc lại tập tin **input.txt** lại từ đầu.

Chúng em đã có rất nhiều ý tưởng khác, nhưng khi xem xét lại yêu cầu của Project là sử dụng kiến thức con trỏ, cấp phát động và mảng động. Vì vậy, chúng em đã có ý tưởng thục hiện chương trình như trên.

Hàm main, hàm chính của chương trình

```
int main() {
    int warning = 0; // Cờ cảnh báo
    int arr_length = 0; // Độ dài mảng
    int* arr = nullptr; // Con trỏ tới mảng chính
    if (!InputArray(arr, arr_length, "input.txt")) // Đọc mảng từ tập tin
        return 1; // Thoát nếu không thể mở tập tin

    int storage_arr_index = 0; // Chỉ số mảng lưu trữ
    int storage_arr_length = 1; // Độ dài mảng lưu trữ
    int** storage_arr = nullptr; // Con trỏ tới mảng lưu trữ

    Storage(storage_arr, storage_arr_index, storage_arr_length, arr, arr_length); // Lưu trữ trạng
thái ban đầu

    string command_input; // Chuỗi nhập lệnh
    string command; // Lệnh được phân tích
    int index_1 = 0, index_2 = 0; // Các chỉ số
    while (getline(cin, command_input)) { // Đọc lệnh từ người dùng
        TokenCommand(command_input, command, index_1, index_2, arr_length, warning); // Phân tích lệnh
        if (command == "delete") { // Lệnh xóa
            Delete(arr, arr_length, index_1); // Xóa phần tử
            OutPutArray(arr, arr_length); // In mảng
```

```
            Storage(storage_arr, storage_arr_index, storage_arr_length, arr, arr_length); // Lưu trạng thái
        }
        else if (command == "insert") { // Lệnh chèn
            Insert(arr, arr_length, index_1, index_2); // Chèn phần tử
            OutPutArray(arr, arr_length); // In mảng
            Storage(storage_arr, storage_arr_index, storage_arr_length, arr, arr_length); // Lưu trạng thái
        }
        else if (command == "undo") { // Lệnh undo (hoàn tác)
            Undo(storage_arr, storage_arr_index, arr, arr_length); // Thực hiện undo
            OutPutArray(arr, arr_length); // In mảng
        }
        else if (command == "redo") { // Lệnh redo (làm lại)
            Redo(storage_arr, storage_arr_index, storage_arr_length, arr, arr_length); // Thực hiện redo
            OutPutArray(arr, arr_length); // In mảng
        }
        else if (command == "save") { // Lệnh save (lưu)
            Save(arr, arr_length, "output.txt"); // Lưu mảng vào tập tin
            OutPutArray(arr, arr_length); // In mảng
        }
        else if (command == "reset") { // Lệnh reset (đặt lại)
            Reset(storage_arr, storage_arr_index, storage_arr_length, arr, arr_length); // Thực hiện reset
        }
        else if (command == "quit") { // Lệnh thoát
            break; // Thoát khỏi vòng lặp
        }
        else { // Lệnh không hợp lệ
            if (warning != 1) {
                cout << "Command not found...\n"; // Thông báo lệnh không hợp lệ
                OutPutArray(arr, arr_length); // In mảng
            }
            else {
                OutPutArray(arr, arr_length); // In mảng nếu có cảnh báo
            }
            warning = 0; // Đặt lại cờ cảnh báo
        }
    }

    Free(storage_arr, storage_arr_index); // Giải phóng mảng lưu trữ
    delete[] arr; // Giải phóng mảng chính
    return 0; // Kết thúc chương trình
}
```

Ở hàm chính của chương trình, chúng em khởi tạo các biến để lưu trữ các dữ liệu như: `int arr_length`, `int* arr`, `int storage_arr_index`, `int** storage_arr`, … Sau đó, thực hiện hàm `Storage` để lưu trữ trạng thái ban đầu. Sau đó, thực hiện vòng lặp để lấy lệnh nhập vào từ người dùng, thực hiện hàm `TokenCommand` để phân tích lệnh được nhập vào từ người dùng. Sau khi có được lệnh chính của người dùng, chương trình sẽ thực thi các hàm tương ứng. Nếu yêu cầu là quit thì sẽ thoát ra khỏi vòng lặp. Cuối cùng, khi kết thúc vòng lặp thì sẽ thực hiện hàm để giải phòng mảng lưu trữ và thực hiện giải phóng mảng dãy số chính.

Hàm Storage để lưu trữ lịch sử các mảng số nguyên

```
void Storage(int**& storage_arr, int& storage_arr_index, int& storage_arr_length, int* arr, int
arr_length) {
    // Tạo mảng mới để lưu các trạng thái cũ
    int** arr_temp = new int* [storage_arr_index + 1];

    // Sao chép dữ liệu cũ vào mảng tạm thời
    if (storage_arr != nullptr && storage_arr_index > 0) {
        memmove(arr_temp, storage_arr, storage_arr_index * sizeof(int*));
    }

    // Tạo mảng mới với phần tử đầu là độ dài mảng
    int* temp = new int[arr_length + 1];
    temp[0] = arr_length;

    // Sao chép các giá trị từ mảng chính vào mảng mới
    for (int i = 0; i < arr_length; ++i) {
        temp[i + 1] = arr[i];
    }

    // Gán mảng mới cho mảng tạm thời
    arr_temp[storage_arr_index] = temp;
    delete[] storage_arr; // Xóa mảng cũ
    storage_arr = arr_temp; // Gán mảng tạm thời cho storage_arr
    storage_arr_index++; // Tăng chỉ số index của storage_arr
    storage_arr_length = storage_arr_index; // Cập nhật độ dài storage_arr
}
```

Ở hàm này, chương trình sẽ thực hiện cấp phát động cho biến `int** arr_temp`, nhằm tạo mảng mới để lưu các trạng thái trước đây, sau đó sao chép dữ liệu cũ vào mảng tạm thời bằng hàm `memmove`. Tiếp tục, tạo mảng mới `int* temp` với phần tử đầu là độ dài mảng, sao chép các giá trị từ mảng chính vào mảng mới, gán mảng mới cho mảng tạm thời và giải phóng mảng cũ. Thực hiện các bước trên bằng kĩ thuật nhằm tránh việc tràn bộ nhớ khi lưu quá nhiều mảng dư thừa.

Hàm TokenCommand, phân tích lệnh của người dùng.

```
void InvalidInputWarning(const string& message, int& warning) {
    warning = 1;
    cout << message << "\n";
}

// Hàm phân tích lệnh
void TokenCommand(const string& command_input, string& command, int& index_1, int& index_2, int
arr_length, int& warning) {
    istringstream stream(command_input);
    string part;
    int part_count = 0;

    command = "";
    index_1 = 0;
    index_2 = 0;

    while (stream >> part) {
        if (part.find_first_of(",") != string::npos) {
            InvalidInputWarning("Invalid characters in index.", warning);
            command = "";
            return;
        }
```

```
        if (part_count == 0) {
            command = part;
        }
        else if (part_count == 1) {
            index_1 = stoi(part);
        }
        else if (part_count == 2) {
            index_2 = stoi(part);
        }

        part_count++;
    }

    if (part_count > 3) {
        InvalidInputWarning("Too many parts in command.", warning);
        command = "";
        return;
    }

    // Kiểm tra các chỉ số trong phạm vi hợp lệ
    if ((command == "delete" || command == "insert") && (index_1 < 1 || index_1 > arr_length)) {
        InvalidInputWarning("Invalid position input.", warning);
        command = "";
        return;
    }

    if (command == "delete" && index_2 != 0) {
        command = "";
        return;
    }

    // Kiểm tra lệnh đơn giản không có chỉ số
    if ((command == "undo" || command == "redo" || command == "save" || command == "reset") &&
(index_1 != 0 || index_2 != 0)) {
        command = "";
        return;
    }
}
```

Ở hàm này, sử dụng kiến thức lớp `istringstream` để xử lý lệnh đầu vào, thay đổi các chỉ số cần thiết và xét tất cả các trường hợp nhập sai đầu vào.

Hàm Delete, xoá phần tử của dãy

```
void Delete(int*& arr, int& arr_length, int index) {
    int* temp = new int[arr_length - 1]; // Tạo mảng mới với độ dài nhỏ hơn 1
    int pos = 0; // Vị trí trong mảng mới
    index--; // Điều chỉnh index để phù hợp với 0-based index

    for (int i = 0; i < arr_length; ++i) {
        if (i == index) { // Bỏ qua phần tử cần xóa
            i++;
        }
        temp[pos++] = arr[i]; // Sao chép phần tử vào mảng mới
    }

    delete[] arr; // Giải phóng mảng cũ
    arr = temp; // Gán mảng mới cho arr
    arr_length--; // Giảm độ dài mảng
}
```

Hàm này chương trình thực hiện kĩ thuật xoá phần tử trong mảng, tạo mảng mới, xử lí trên mảng mới, giải phóng mảng cũ và gán mảng mới vào mảng chính.

Hàm Insert, thêm phần tử

```
void Insert(int*& arr, int& arr_length, int index_1, int index_2) {
    int* temp = new int[arr_length + 1]; // Tạo mảng mới với độ dài lớn hơn 1
    int pos = 0; // Vị trí trong mảng mới
    index_1--; // Điều chỉnh index để phù hợp với 0-based index

    for (int i = 0; i < arr_length; ++i) {
        if (i == index_1) { // Chèn phần tử tại vị trí này
            temp[pos++] = index_2; // Chèn phần tử mới
        }
        temp[pos++] = arr[i]; // Sao chép các phần tử còn lại
    }

    delete[] arr; // Giải phóng mảng cũ
    arr = temp; // Gán lại mảng
    arr_length++; // Tăng độ dài mảng
}
```

Hàm này chương trình thực hiện kĩ thuật thêm 1 phần tử vào mảng, tạo mảng mới, xử lí trên mảng mới, giải phóng mảng cũ và gán mảng mới vào mảng chính.

Hàm Undo, phục hồi chức năng gần nhất

```
void Undo(int** storage_arr, int& storage_arr_index, int*& arr, int& arr_length) {
    if (storage_arr_index - 2 < 0) { // Kiểm tra chỉ số hợp lệ
        cout << "No more commands to undo...\n";
        return;
    }

    // Tạo mảng mới với độ dài từ mảng lưu trữ
    int* temp = new int[storage_arr[storage_arr_index - 2][0]];

    // Sao chép dữ liệu từ mảng lưu trữ
    memcpy(temp, &storage_arr[storage_arr_index - 2][1], storage_arr[storage_arr_index - 2][0] *
sizeof(int));

    delete[] arr; // Giải phóng mảng cũ
    arr = temp; // Gán lại mảng mới
    arr_length = storage_arr[storage_arr_index - 2][0]; // Cập nhật độ dài mảng
    storage_arr_index--; // Giảm chỉ số lưu trữ
}
```

Ở hàm này, chương trình thực hiện kiểm tra `storage_arr_index` để xem có các bước nào trước đây để phục hồi hay không, nếu không có thì xuất ra màn hình `No more commands to undo...` .Sau đó, tạo một mảng tạm mới từ mảng lưu trữ, sao chép dữ liệu, giải phóng mảng cũ và gán lại mảng mới. Lúc này chỉ số của mảng lưu trữ sẽ giảm xuống.

Hàm Redo, lặp lại chức năng đã phục hồi gần nhất

```
void Redo(int**& storage_arr, int& storage_arr_index, int storage_arr_length, int*& arr, int&
arr_length) {
    if (storage_arr_index >= storage_arr_length) { // Kiểm tra chỉ số hợp lệ
        cout << "No more commands to redo...\n";
```

```
        return;
    }

    int new_arr_length = storage_arr[storage_arr_index][0]; // Lấy độ dài mảng mới
    int* temp = new int[new_arr_length]; // Tạo mảng mới

    // Sao chép dữ liệu từ mảng lưu trữ
    memcpy(temp, &storage_arr[storage_arr_index][1], new_arr_length * sizeof(int));

    delete[] arr; // Giải phóng mảng cũ
    arr = temp; // Gán lại mảng mới
    arr_length = storage_arr[storage_arr_index][0]; // Cập nhật độ dài mảng
    storage_arr_index++; // Tăng chỉ số lưu trữ
}
```

Ở hàm này, cách thức cũng tương tự như hàm `Undo`, nhưng lại tăng chỉ số lưu trữ để xuất ra màn hình dãy đã Redo.

Hàm Reset, lặp lại chức năng đã phục hồi gần nhất

```
void Reset(int**& storage_arr, int& storage_arr_index, int& storage_arr_length, int*& arr, int&
arr_length) {
    arr_length = 0; // Đặt lại độ dài mảng
    delete[] arr; // Giải phóng mảng chính
    Free(storage_arr, storage_arr_index); // Giải phóng mảng lưu trữ
    storage_arr_index = 0; // Đặt lại chỉ số lưu trữ
    storage_arr_length = 1; // Đặt lại độ dài lưu trữ
    if (!InputArray(arr, arr_length, "input.txt")) // Đọc lại dữ liệu từ tập tin
        cout << "Can't open the file !"; // Thông báo lỗi nếu không thể mở tập tin
    Storage(storage_arr, storage_arr_index, storage_arr_length, arr, arr_length); // Lưu trữ trạng
thái mới
}
```

Ở hàm này, thực hiện hàm `Free` để thực hiện giải phóng mảng lưu trữ, giải phóng mảng chính, đặt lại các chỉ số và đọc lại dữ liệu.

Hàm Save, lưu lại dãy số

```
void Save(int* arr, int arr_length, string file_name) {
    ofstream file(file_name); // Mở tập tin để ghi
    if (file.is_open()) { // Kiểm tra tập tin mở thành công
        // Ghi các phần tử của mảng vào tập tin
        for (int i = 0; i < arr_length; ++i) {
            file << arr[i] << " ";
        }
        file.close(); // Đóng tập tin
        cout << "Numbers have been stored...\n"; // Thông báo lưu thành công
    }
    else {
        cout << "Can't open the file !"; // Thông báo lỗi nếu không thể mở tập tin
    }
}
```

Ở hàm này, chương trình thực hiện ghi dãy số và tập tin **output.tx**t . Đồng thời, màn hình console hiển thị `Numbers have been stored...`

2.3
Kết quả của chương trình trong một số trường hợp lỗi đầu vào

- Nhập lệnh không có thực

```
12 78 1 43 90 56 78
> add 2 3
Command not found...
```

- Nhập sai chỉ số

```
12 78 1 43 90 56 78
> insert
Invalid position input.
12 78 1 43 90 56 78
> insert 2.5 3.5
Invalid characters in index.
12 78 1 43 90 56 78
> insert 2.5
Invalid characters in index.
12 78 1 43 90 56 78
> delete
Invalid position input.
12 78 1 43 90 56 78
> delete 3.5
Invalid characters in index.
12 78 1 43 90 56 78
> delete 100
Invalid position input.
```

- Nhập sai cú pháp

```
12 78 1 43 90 56 78
> insert 2 3 4 5
Too many parts in command.
12 78 1 43 90 56 78
> delete 3 4 5 4
Too many parts in command.
```

- Tất cả các đầu vào sai còn lại

```
Command not found...
```