



fit@hcmus

BÁO CÁO MÔN HỌC PROJECT 1

SINH VIÊN THỰC HIỆN:

Nguyễn Phúc Hoàng - 23120264

Tống Dương Thái Hoà - 23120262



MỤC LỤC

1. Cài đặt chương trình.....	2
1.1. Cài đặt tập tin header (file .h).....	2
1.1.1. File Header.h.....	2
1.1.2. File function.h.....	3
1.1.2.1. Hàm lưu địa chỉ các phần trong mảng.....	3
1.1.2.2. Hàm đọc dữ liệu từ file input.txt.....	4
1.1.2.3. Hàm cấp phát và giải phóng bộ nhớ.....	4
1.1.2.4. Hàm thay đổi kích thước của mảng.....	5
1.2. Cài đặt tập tin text (file .txt).....	6
2. Mô tả các lệnh và chức năng của chương trình.....	7
2.1. Delete pos (Xóa 1 phần tử tại vị trí pos).....	7
2.2. Insert pos val (Thêm phần tử có giá trị val vào vị trí pos).....	8
2.3. Undo (Phục hồi lệnh gần nhất).....	9
2.3.1. Ý tưởng.....	9
2.3.2. Cài đặt undo_stack để lưu lại các thao tác undo.....	9
2.3.3. Hàm undo_push để thêm phần tử ngoài cùng của RedoStack vào UndoStack....	10
2.4. Redo (Lặp lại lệnh đã phục hồi gần nhất).....	10
2.5. Save (lưu dãy số vào thư mục output.txt trên 1 dòng).....	11
2.6. Reset (Khởi tạo lại phiên làm việc).....	12
2.7. Quit (Kết thúc chương trình).....	13
2.8. Các hàm hỗ trợ cho chương trình:.....	13
2.8.1. Hàm current_arr.....	13
2.8.2. Hàm menu.....	14
2.8.3. Hàm size_file_data.....	15
2.8.4. Hàm input_checking.....	15
2.8.5. Hàm input_pos và input_pos_and_val.....	16
3. Các bước thực hiện của chương trình.....	17

1.Cài đặt chương trình.

1.1.Cài đặt tập tin header (file .h).

1.1.1.File Header.h

Đầu tiên, tạo dòng code số 4 và 5 của file Header.h, thêm vào 2 thư viện là *iostream* để sử dụng các hàm chuẩn của C++ như cin, cout,... và thư viện *fstream* dùng để thao tác với file.

```
#include <iostream>
#include <fstream>
```

Dòng số 7 để khai báo biến *headsize* với kiểu dữ liệu integer, biến này dùng để lưu trữ kích thước của kiểu dữ liệu int. Từ khoá “extern” dùng để khai báo biến toàn cục cho toàn bộ chương trình, bao gồm cả các file khác cùng đuôi. Việc này sẽ giúp tiết kiệm bộ nhớ.

```
extern int headsize;
```

Từ dòng số 9 đến 11 dùng để khai báo kiểu dữ liệu enum Action dùng để lưu các tập giá trị hằng số được định sẵn có liên quan với nhau, cụ thể ở đây *delete_action* sẽ được gán giá trị 0 và được định nghĩa là thao tác xóa phần tử, *insert_action* sẽ được gán giá trị 1 và được định nghĩa là thao tác thêm phần tử.

```
enum Action {
    delete_action, insert_action
};
```

Từ dòng số 13 đến dòng số 25, khai báo kiểu dữ liệu cấu trúc mang tên Stack mang kiểu dữ liệu T là template với mục đích có thể tái sử dụng lại hàm với nhiều kiểu dữ liệu khác. Stack với các kiểm dữ liệu con là history để lưu lịch sử thao tác, pos để lưu vị trí phần tử, val để lưu giá trị phần tử. Trong struct Stack có phần định nghĩa lại operator = để overloading toán tử, việc này tránh gây lỗi bộ nhớ khi thao tác trên các con trỏ.

```
template <class T>
struct Stack {
    Action history;
    int pos;
    T val;
    Stack& operator = (const Stack& obj) {
        this->history = obj.history;
        this->pos = obj.pos;
        this->val = obj.val;
        return *this;
    }
}
```

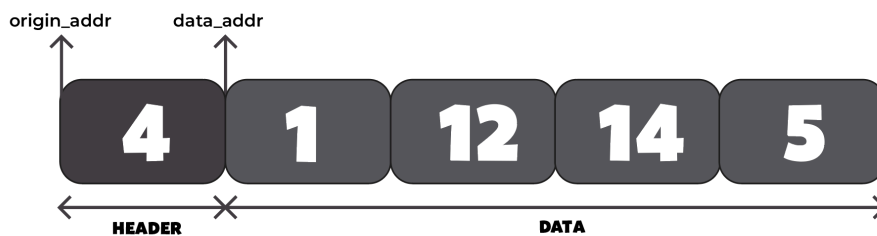
```
};
```

Các dòng còn lại sẽ định nghĩa nguyên mẫu hàm (prototype), cụ thể là định nghĩa các nguyên mẫu hàm có trong chương trình.

1.1.2.File function.h

1.1.2.1.Hàm lưu địa chỉ các phần trong mảng.

Với ý tưởng dành 1 số byte trong vùng nhớ được cấp phát để lưu trữ kích thước mảng, chính vì vậy mảng sẽ gồm 2 phần là header để lưu trữ số lượng phần tử mảng và data để lưu trữ các phần tử mảng, hình dưới minh hoạ ý tưởng trên:



Từ dòng 9 đến dòng 15 là nội dung hàm `origin_addr` với tham số là một con trỏ `aData` lưu địa chỉ của phần tử đầu tiên trong `data`. Hàm này trả về địa chỉ gốc của mảng tức `origin_addr` trên hình bằng phép tính $(T*)((char*)(aData) - \text{headsize})$. $(char*)$ dùng để ép kiểu `aData` về con trỏ `char*`, điều này sẽ giúp việc tính toán địa chỉ chính xác hơn. giả sử `aData` có địa chỉ `0x12` thì phép tính trên sẽ trả về `0x8` (vì `headsize` chứa 4 byte) chính là địa chỉ của `origin_addr`. Nếu như `aData` chưa tồn tại hay lưu giá trị `nullptr` thì hàm sẽ trả về `nullptr`.

```
template <class T>
T* origin_addr(T*& aData) {
    if (aData != nullptr) {
        return (T*)((char*)aData - headsize);
    }
    return nullptr;
}
```

Từ dòng 17 đến 23 là nội dung của hàm `data_addr` với tham số là một con trỏ `aOrigin` lưu địa chỉ gốc của mảng. Tương tự như hàm trên nhưng hàm này sẽ trả về địa chỉ của phần tử đầu tiên trong `data`.

```
template <class T>
T* data_addr(T*& aOrigin) {
```

```

        if (aOrigin != nullptr) {
            return (T*)((char*)aOrigin + headsize);
        }
        return nullptr;
    }
}

```

1.1.2.2.Hàm đọc dữ liệu từ file input.txt.

Từ dòng 25 đến dòng 43 là nội dung hàm read_data với tham số là con trỏ lưu trữ địa chỉ của mảng. Hàm này chỉ đơn giản là mở file input.txt để đọc và lưu phần tử vào mảng aData.

```

    template <class T>
    void read_data(T*& aData) {
        if (!aData) return;

        int index = 0;

        std::fstream fin
        fin.open("resources/input.txt", std::ios::in);

        if (!fin.is_open()) {
            std::cout << "Fail to open file!\n";
            return;
        }
        while (!fin.eof()) {
            fin >> aData[index++];
        }

        fin.close();
    }
}

```

1.1.2.3.Hàm cấp phát và giải phóng bộ nhớ.

Dòng 45 đến dòng 59 là dòng cấp phát bộ nhớ cho mảng với 2 tham số là con trỏ aData lưu trữ địa chỉ của mảng và n là kích thước mảng. Hàm này đơn giản là cấp phát cho mảng $[n * \text{sizeof}(T) + \text{sizeof}(\text{int})]$ ô nhớ, tức là giả sử mảng có 7 phần tử kiểu int thì sẽ cấp phát cho $(7 * 4 + 4)$ byte trong vùng nhớ Heap để lưu trữ số lượng phần tử mảng (phần $\text{sizeof}(\text{int})$) và các phần tử của mảng. dòng 56

để lưu địa chỉ đã được cấp phát vào mảng aData và 57 dùng dài tham chiếu tới con trỏ mảng để lưu trữ phần tử vào phần Header của mảng.

```
template <class T>
void allocate_data(T*& aData, const int& n) {
    if (aData) return;

    aData = (T*)malloc(n * sizeof(T) + sizeof(int));

    if (!aData) {
        std::cout << "Fail allocate!\n";
        return;
    }

    aData = data_addr<T>(aData);
    *((int*)origin_addr<T>(aData)) = n;
}
```

Dòng 60 đến 73 đơn giản là giải phóng toàn bộ mảng trong vùng nhớ Heap đã xin cấp phát lúc trước.

```
template <class T>
T* free_arr(T*& aData) {
    if (!aData) {
        return nullptr;
    }

    aData = origin_addr(aData);

    free(aData);

    aData = nullptr;

    return aData;
}
```

1.1.2.4. Hàm thay đổi kích thước của mảng.

Từ dòng 75 đến dòng 94 là hàm resize_arr với 2 tham số là con trỏ lưu địa chỉ mảng và giá trị mới của kích thước mảng. Dòng 77 khai báo 1 biến sizeArr và gán giá trị bằng cách dài tham chiếu đến origin_addr của aData để lấy số lượng phần

tử của mảng cũ. Dòng 81 sẽ tạo ra 1 mảng mới với kích thước mới và sẽ cấp phát cho mảng này newSize ô nhớ. Dòng 85 dùng toán tử 3 ngôi ?:, toán tử này tương tự if else, nếu newSize < sizeArr thì cập nhật sizeArr = newSize còn không thì vẫn giữ nguyên giá trị sizeArr, mục đích việc này dùng để sao chép các phần tử của mảng mới sang mảng với số phần tử sao chép là min của newSize và sizeArr. Sau khi thực hiện sao chép xong thì sẽ giải phóng aData và trả về mảng mới với kích thước mới.

```
template <class T>
T* resize_arr(T*& aData, const int& newSize) {
    int sizeArr = *((int*)origin_addr<T>(aData));

    if (sizeArr == newSize || aData == nullptr) return
aData;

    T* aNew = nullptr;

    allocate_data(aNew, newSize);

    sizeArr = (newSize < sizeArr) ? newSize : sizeArr;

    for (int i = 0; i < sizeArr; i++) {
        aNew[i] = aData[i];
    }

    aData = free_arr<T>(aData);

    return aNew;
}
```

1.2. Cài đặt tập tin text (file .txt).

Tại đường dẫn ../resources thêm 2 file là input.txt và output.txt. File input.txt dùng để lưu các phần tử của mảng 1 chiều. Cụ thể file input.txt có nội dung:

1 25 84 95 33 44 5

Nội dung trên là các con số cách nhau bởi dấu cách, tức là một mảng có 7 phần tử và các phần tử lần lượt là 1, 25, 84, 95, 33, 44, 5.

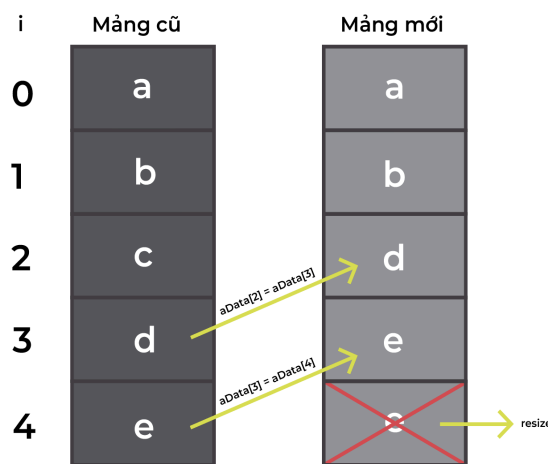
File output.txt dùng để lưu lại mảng khi người dùng thực hiện chức năng Save. Mảng sẽ được lưu dưới dạng từng phần tử hay con số của mảng cách nhau bởi 1 dấu cách.

2. Mô tả các lệnh và chức năng của chương trình.

Các dòng code lệnh và chức năng của chương trình được cài đặt ở file *function.h*

2.1.Delete pos (Xóa 1 phần tử tại vị trí pos).

Ý tưởng: Thực hiện vòng lặp từ chỉ số pos đến chỉ số sizeArr - 1 (phần tử kế cuối của mảng) để gán giá trị chỉ số (i) cho (i + 1), sau đó giảm kích thước mảng đi 1 phần tử bằng hàm `resize_arr()`. Minh hoạ bởi hình dưới đây, giả sử cho mảng có 5 phần tử là a, b, c, d, e, thực hiện xóa phần tử ở vị trí số 2:



Đoạn code từ dòng 96 đến 115 thực hiện ý tưởng trên, đầu tiên tại dòng 98 tạo 1 biến `sizeArr` gán giá trị của header trong mảng (tức số lượng phần tử hiện có trong mảng). Dòng 100 đến 104 để kiểm tra giá trị hợp lệ của vị trí xóa. Dòng 106 đến 108 thực hiện ý tưởng, dòng 110 giảm đi kích thước mảng 1 đơn vị và cập nhật lại kích thước mảng ở dòng 112.

```
template <class T>
void delete_pos(T*& aData, int& pos) {
    int sizeArr = *((int*)origin_addr<T>(aData));

    if (pos > sizeArr || pos < 0) {
        system("cls");
        std::cout << "\n\t Invalid pos!\n";
        return;
    }
}
```



```

    for (int i = pos; i < sizeArr - 1; i++) {
        aData[i] = aData[i + 1];
    }

    aData = resize_arr<T>(aData, sizeArr - 1);

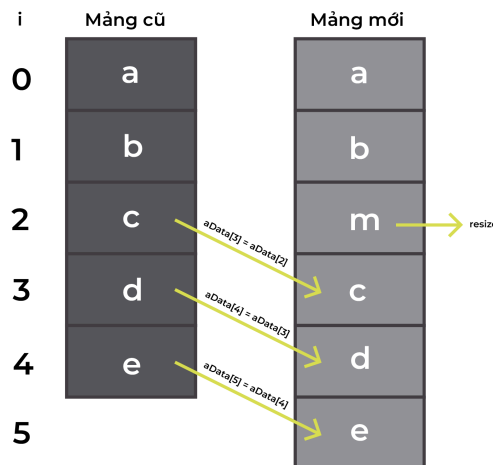
    *((int*)origin_addr<T>(aData)) = sizeArr - 1;

    system("cls");
}

```

2.2. Insert pos val (Thêm phần tử có giá trị val vào vị trí pos).

Ý tưởng: Tăng kích thước cho mảng thêm 1 phần tử. Sau đó, thực hiện vòng lặp lùi từ sizeArr (phần tử cuối trong mảng) về vị trí pos + 1, mỗi vòng lặp thực hiện gán giá trị của chỉ số (i) cho (i - 1). Cuối cùng gán giá trị val cho chỉ số pos trong mảng. Minh họa ý tưởng bên dưới (Giả sử cho một mảng như 2.1. và thêm phần tử m vào vị trí 2):



Đoạn code từ dòng 117 đến 135 thực hiện ý tưởng trên. Tương tự như xóa phần tử, tạo 1 biến sizeArr lưu số lượng phần tử của mảng bằng cách dài tham chiếu đến phần Header của aData. dòng 121 đến 125 kiểm tra giá trị hợp lệ của pos (không nằm ngoài mảng). Dòng 127 cập nhật lại kích thước mảng thêm 1 phần tử. Các dòng còn lại thực hiện ý tưởng bên trên.

2.3.Undo (Phục hồi lệnh gần nhất).

2.3.1.Ý tưởng.

Ở phần 1.1.1. đã giới thiệu đến cấu trúc Stack, ở phần này sẽ sử dụng cấu trúc đó để lưu lại những thao tác của undo, cụ thể là sẽ tạo UndoStack mang cấu trúc Stack đã được định nghĩa.

Trước khi thực hiện các lệnh xóa hay thêm phần tử thì ta sẽ push vào UndoStack, tức là thêm phần tử gồm các giá trị là history (như đã định nghĩa thì lưu vào history 0 là delete_pos và 1 là insert_pos_val), pos, val vào stack.

Khi thực hiện lệnh undo thì hàm undo_stack sẽ được gọi ra với nhiệm vụ pop phần tử ra khỏi UndoStack và thực hiện chức năng phủ định với chức năng được lưu trong history (nếu như history hiện chức năng delete thì undo sẽ thực hiện chức năng của insert và ngược lại), tức là sẽ lấy phần tử cuối ra ngoài và xóa khỏi UndoStack. Phần tử này không bị xóa hoàn toàn mà sẽ được đưa vào RedoStack bằng hàm push_redo (sẽ được trình bày bên dưới) để thuận lợi cho việc cài đặt các chức năng Undo và Redo.

2.3.2.Cài đặt undo_stack để lưu lại các thao tác undo.

Như ý tưởng trên, ta sẽ tận dụng struct Stack đã được tạo ra và khởi tạo 1 kiểu dữ liệu stack là undo. Mỗi khi thực hiện lệnh delete hay insert thì ta sẽ thêm 1 phần tử vào stack. Dòng code 180 đến 200 thực hiện ý tưởng. Đầu tiên dòng 182 sẽ lấy số lượng phần tử trong stack undo để gán vào sizeStack. Dòng 184 đến 187 kiểm tra giá trị hợp lệ của sizeStack và aData, nếu sizeStack bằng 0 tức chưa thực hiện lệnh nào thì hàm sẽ kết thúc do chưa có giá trị để lưu vào UndoStack còn nếu aData bằng nullptr tức cấp phát cho aData thất bại thì hàm cũng sẽ kết thúc. Dòng 189 đến 193 dùng để hoàn tác chức năng delete khi người dùng thực hiện lệnh delete trước đó, cụ thể dòng 189 để kiểm tra xem history có phải là lệnh delete không, nếu có thì sẽ thực hiện chức năng ngược lại của delete là insert vào mảng vị trí và phần tử đã được lưu trong UndoStack. Dòng 191 để push vào redo (hàm push_redo sẽ được trình bày bên dưới) các giá trị của phần tử ngoài cùng của UndoStack và dòng 192 để xóa đi phần tử ngoài cùng nằm trong UndoStack.

```
template <class T>
void undo_stack(T*& aData, Stack<T>*& undo, Stack<T>*&
redo) {
    int sizeStack = *((int*)origin_addr(undo));

    if (sizeStack == 0 || aData == nullptr) {
        system("cls");
        return;
    }
}
```

```

        if (undo[sizeStack - 1].history == delete_action) {
            inset_pos_val(aData, undo[sizeStack - 1].pos,
undo[sizeStack - 1].val);
            redo_push(undo[sizeStack - 1], redo);
            undo = resize_arr(undo, sizeStack - 1);
        }
        else {
            delete_pos(aData, undo[sizeStack - 1].pos);
            redo_push(undo[sizeStack - 1], redo);
            undo = resize_arr(undo, sizeStack - 1);
        }
        system("cls");
    }

```

2.3.3. Hàm undo_push để thêm phần tử ngoài cùng của RedoStack vào UndoStack.

Như đã trình bày bên trên, nhiệm vụ của hàm undo_push là khi thực hiện chức năng Redo, ta sẽ push phần tử ngoài cùng của RedoStack vào UndoStack.

```

template <class T>
void undo_push(const Stack<T>& redo , Stack<T>*& undo) {
    int sizeStack = *((int*)origin_addr(undo));

    undo = resize_arr(undo, sizeStack + 1);

    undo[sizeStack] = redo;
}

```

Dòng 237 đến 244 là nội dung của hàm undo_push. Dòng 239 dùng dải tham chiếu để lấy số lượng phần tử của stack undo và lưu vào biến sizeStack. Dòng 241 sẽ tăng kích thước của UndoStack lên 1 đơn vị với mục đích là push phần tử ngoài cùng của RedoStack vào.

2.4. Redo (Lặp lại lệnh đã phục hồi gần nhất).

Ý tưởng của chức năng Redo tương tự như Undo đã được trình bày ở trên, chỉ thay vị trí của undo và redo nên sẽ không trình bày thêm. Dưới đây là đoạn code của redo_stack và redo_push:

```

void redo_stack(T*& aData, Stack<T>*& undo, Stack<T>*& redo) {

```

```

int sizeStack = *((int*)origin_addr(redo));

if (sizeStack == 0 || aData == nullptr) {
    system("cls");
    return;
}

if (redo[sizeStack - 1].history == delete_action) {
    delete_pos(aData, redo[sizeStack - 1].pos);
    undo_push(redo[sizeStack - 1], undo);
    redo = resize_arr(redo, sizeStack - 1);
}
else {
    inset_pos_val(aData, redo[sizeStack - 1].pos,
redo[sizeStack - 1].val);
    undo_push(redo[sizeStack - 1], undo);
    redo = resize_arr(redo, sizeStack - 1);
}
system("cls");
}

template <class T>
void redo_push(const Stack<T>& undo, Stack<T>*& redo) {
    int sizeStack = *((int*)origin_addr(redo));

    redo = resize_arr(redo, sizeStack + 1);

    redo[sizeStack] = undo;
}

```

2.5. Save (lưu dãy số vào thư mục output.txt trên 1 dòng).

Chức năng save được thực hiện ở dòng 137 đến 158. Trong suốt quá trình chương trình chạy, mảng sẽ được lưu vào aData, khi người dùng thực hiện chức năng Save thì mảng sẽ được ghi vào file output.txt. Dưới đây là đoạn mã thực hiện chức năng Save:

```

template <class T>

```

```

void save_file_data(T*& aData) {
    if (!aData) return;

    int n = *((int*)origin_addr<T>(aData));

    std::fstream fout;
    fout.open("resources/output.txt", std::ios::trunc |
std::ios::out);

    if (!fout.is_open()) {
        std::cout << "Fail to open file!\n";
        return;
    }
    for (int i = 0; i < n; i++) {
        fout << aData[i];

        if (i != (n - 1)) {
            fout << " ";
        }
    }
    system("cls");
}

```

Mảng được lưu như file input.txt không có gì đặc biệt, chú ý dòng 144 có chế độ std::ios::trunc với mục đích là xóa toàn bộ dữ liệu cũ đã tồn tại trong file output.txt.

2.6.Reset (Khởi tạo lại phiên làm việc).

Tại file main.cpp, chức năng reset được thực hiện bằng cách tạo 1 vòng lặp do while với biến reset kiểm tra chức năng reset, lồng trong là vòng lặp while với biến is_running kiểm tra xem người dùng có muốn chạy chương trình tiếp không. Cả 2 biến này mang kiểu dữ liệu bool và reset được gán giá trị false còn is_running được gán giá trị true.

Nếu người dùng chọn chức năng Reset thì biến reset sẽ được gán giá trị true và biến is_running được gán giá trị false. Việc này có nghĩa là vòng lặp chức năng sẽ bị dừng lại và vòng lặp reset sẽ được thực hiện. Lúc này, chương trình sẽ được thực hiện lại toàn bộ. các biến reset và is_running sẽ được gán lại giá trị như lúc ban đầu. Minh họa ở đoạn code dưới đây:

```

int main() {

```

```

    bool reset = false;

    do
    {
        reset = false;
        bool is_running = true;
        //...
        while (is_running) {
            switch (choice) {
                //Các lệnh và chức năng khác được thực hiện
trên này

                case 7 :

                    reset = true;
                    is_running = false;
                    system("cls");
                    std::cout << "\t Reset is done!\n";
                    break;
                    //...
            }
        }
    } while (reset);

    return 0;
}

```

2.7.Quit (Kết thúc chương trình).

Chức năng Quit được tạo bằng việc sử dụng giá trị của biến `is_running` đã được trình bày ở chức năng Reset. Bằng việc gán cho `is_running` giá trị `false`, vòng lặp `while` sẽ được dừng và chương trình sẽ kết thúc.

2.8.Các hàm hỗ trợ cho chương trình:

2.8.1.Hàm `current_arr`.

Thực chất hàm này chính là hàm in ra mảng đang có khi thực hiện xong các lệnh và chức năng:

```

template <class T>
void current_arr(T*& aData) {

```

```

    if (!aData) return;

    int n = *((int*)origin_addr<T>(aData));

    std::cout << "\n\t Current array: ";

    for (int i = 0; i < n; i++) {
        std::cout << aData[i];

        if (i != (n - 1)) {
            std::cout << " ";
        }
    }

    std::cout << "\n";
}

```

2.8.2.Hàm menu.

Như tên của hàm, hàm này dùng để in ra menu các lệnh và chức năng của chương trình:

```

void menu() {
    std::cout << "\n\t
    _____MENU_____\n"
    << "\t\t1. Delete element at position          |\n"
    << "\t\t2. Insert element at position with value  |\n"
    << "\t\t3. Undo last operation                    |\n"
    << "\t\t4. Redo last undone operation              |\n"
    << "\t\t5. Save array to file                      |\n"
    << "\t\t6. Reset workspace                        |\n"
    << "\t\t7. Quit                                    |\n"
    <<
    "\t|_____|\n";
    std::cout << "\t Please input: ";
}

```

2.8.3. Hàm size_file_data.

Hàm này sẽ trả về số lượng phần tử có trong mảng được lưu trữ trong file input.txt:

```
int size_file_data() {  
  
    int n = 0;  
    std::fstream fin;  
    fin.open("resources/input.txt", std::ios::in);  
  
    if (!fin.is_open()) {  
        std::cout << "Fail to open file!\n";  
        return n;  
    }  
  
    while (!fin.eof()) {  
        int temp;  
        fin >> temp;  
        n++;  
    }  
    return n;  
}
```

2.8.4. Hàm input_checking.

Hàm này sẽ kiểm tra xem giá trị nhập vào có đúng là số không, nếu như người dùng nhập chữ hay ký tự đặc biệt vào thì hàm sẽ báo lỗi:

```
void input_checking(int& num) {  
    bool check;  
  
    do {  
        check = false;  
        std::cin >> num;  
        if (std::cin.fail()) {  
            std::cin.clear();  
            std::cout << "\t Invalid input! Input again: ";  
            check = true;  
        }  
        std::cin.ignore();  
    } while (check);  
}
```



```
    } while (check);
}
```

2.8.5.Hàm input_pos và input_pos_and_val.

Hai hàm này sẽ kiểm tra giá trị hợp lệ của pos (vị trí) và val (giá trị). Nếu pos thuộc vị trí nằm ngoài mảng hoặc giá trị val là âm thì hàm sẽ báo lỗi và yêu cầu người dùng nhập lại:

```
void input_pos(const int& n,int& pos) {

    bool check = false;

    do {
        if (check) {
            std::cout << "\t Invalid position! Input again\n";
        }
        std::cout << "\t Please input position you want to delete:
";
        input_checking(pos);
        check = true;
    } while (pos < 1 || pos > n);

    pos--;
}

void input_pos_and_val(const int& n, int& pos, int& val) {

    bool check = false;

    do {
        if (check) {
            std::cout << "\t Invalid position! Input again\n";
        }
        std::cout << "\t Please input position you want to insert:
";
        input_checking(pos);
        std::cout << "\t Please input value you want to insert: ";
    }
```

```

        input_checking(val);
        check = true;
    } while (pos < 1 || pos > n);

    pos--;
}

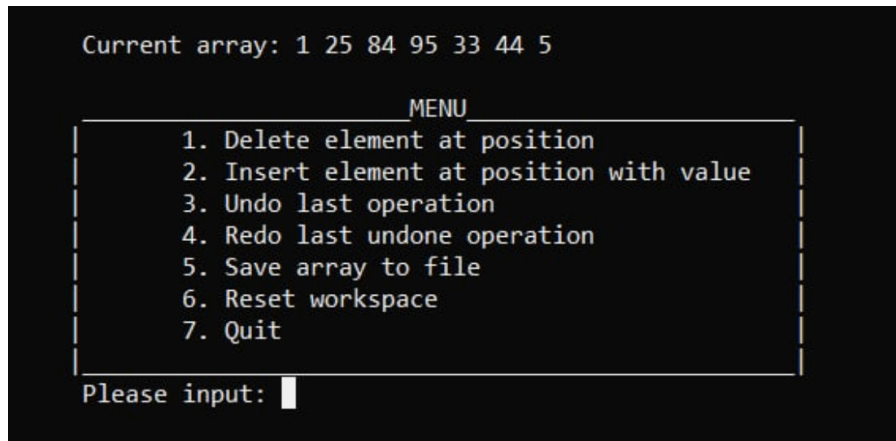
```

3. Các bước thực hiện của chương trình.

- Bước 1: Có rất nhiều cách để chạy chương trình này, đối với phần mềm VS2022 hay các phần mềm khác có hỗ trợ compiler thì việc đơn giản chỉ là Run chương trình. Còn đối với việc chạy chương trình trên terminal, điều này yêu cầu người dùng phải chạy chương trình bằng tay, điều đầu tiên cần làm là truy cập vào đúng thư mục của chương trình, sau đó mở terminal và thực hiện dòng code:

```
g++ -o main main.cpp function.cpp
```

- Bước 2: Khi chương trình được chạy trên màn hình compiler sẽ hiện ra menu, việc đơn giản là chọn chức năng theo từng số tương ứng trên menu.



```

Current array: 1 25 84 95 33 44 5

      MENU
      -----
      1. Delete element at position
      2. Insert element at position with value
      3. Undo last operation
      4. Redo last undone operation
      5. Save array to file
      6. Reset workspace
      7. Quit

Please input: 

```