

# 初始项目结构

首先，根据项目要求创建一个符合要求的项目结构。

从要求中，大致要以下几个模块：

- 数据结构 data\_structures
- 排序 sorting
- 可视化 visualization
- 其他工具类，如随机数生成器 utils

根据上述模块，可以创建以下目录结构：

```
.
├── Makefile
├── README.md
├── build
├── doc
│   └── report.md
├── include
│   ├── data_structures
│   ├── sorting
│   ├── utils
│   └── visualization
├── lib
├── src
│   ├── data_structures
│   ├── main.cpp
│   ├── sorting
│   ├── utils
│   └── visualization
└── test
```

## 数据结构与排序

### 数据结构

首先从数据结构开始，在 include/data\_structures 目录下创建 array.h 文件，定义数据结构的接口。

作为一个排序算法，数组是最不可少的一个数据结构，因此首先定义数组的接口。

```
#ifndef ARRAY_H
#define ARRAY_H

class Array {
public:
    Array(int size);
    ~Array();

    int getSize() const;
    int get(int index) const;
    void set(int index, int value);

private:
    int* data;
    int size;
};

#endif // ARRAY_H
```

在上面的代码中，定义了一个 Array 类，包含了数组的基本操作，如获取数组大小、获取数组元素、设置数组元素等。

之后需要实现这个类，因此在 src/data\_structures 目录下创建 array.cpp 文件，实现 Array 类的接口。

由于本次项目不考虑泛型，因此数组的元素类型固定为 int。

本项目较为复杂，为了避免命名空间冲突，不使用 using namespace std;，而是使用 std:: 前缀。

```

#include "array.h"
#include <stdexcept>

Array::Array(int size) : size(size) {
    data = new int[size];
}

Array::~Array() {
    delete[] data;
}

int Array::getSize() const {
    return size;
}

int Array::get(int index) const {
    if (index < 0 || index >= size) {
        throw std::out_of_range("Index out of range");
    }
    return data[index];
}

void Array::set(int index, int value) {
    if (index < 0 || index >= size) {
        throw std::out_of_range("Index out of range");
    }
    data[index] = value;
}

```

在上面的代码中，实现了 Array 类的构造函数、析构函数、获取数组大小、获取数组元素、设置数组元素等接口。

## 排序

接下来实现排序算法，我希望首先实现一个冒泡排序。

在 include/sorting 目录下创建 bubble\_sort.h 文件，定义冒泡排序的接口。

然后在 src/sorting 目录下创建 bubble\_sort.cpp 文件，实现冒泡排序的接口。

过程和上面的数据结构类似，这里不再赘述。

冒牌排序的实现如下：

```
void bubbleSort(Array& array) {
    int size = array.getSize();
    for (int i = 0; i < size - 1; ++i) {
        for (int j = 0; j < size - i - 1; ++j) {
            if (array.get(j) > array.get(j + 1)) {
                int temp = array.get(j);
                array.set(j, array.get(j + 1));
                array.set(j + 1, temp);
                visualize(array);
            }
        }
    }
}
```

接下来是实现随机数填充数组的函数，这个函数在 `utils` 模块中。

随机数的生成使用 `rand()` 函数，种子使用 `time(nullptr)` 函数。

在 `utils` 目录下创建 `random_generator` 文件，定义随机数生成函数。

头文件和此前模块类似，源文件的核心实现如下：

```
void generateRandomArray(Array& array, int minValue, int maxValue) {
    std::srand(std::time(nullptr));
    for (int i = 0; i < array.getSize(); ++i) {
        array.set(i, minValue + std::rand() % (maxValue - minValue + 1));
    }
}
```

## 可视化

接下来需要完成可视化部分，但此时我想要先对此前的内容进行测试，先写一个简单的 `std::cout` 来显示的可视化。在 `visualization` 目录下创建 `visualize` 头文件和源文件。

可视化需要获取到数组的大小和数组的元素，因此 `visualize` 函数需要一个 `Array` 对象作为参数。

在排序算法的恰当时机调用 `visualize` 函数，即可实现可视化。根据冒泡排序的实现，可以在交换元素时调用 `visualize` 函数。

目前还没有引入 ncurses 库，因此只能使用 `std::cout` 来进行简单的可视化。  
简易可视化的代码如下：

```
void visualize(const Array& array) {
    system("clear");
    for (int i = 0; i < array.getSize(); ++i) {
        std::cout << array.get(i) << "\t";
        for (int j = 0; j < array.get(i); ++j) {
            std::cout << "*";
        }
        std::cout << std::endl;
    }
    std::this_thread::sleep_for(std::chrono::milliseconds(100));
}
```

## Makefile

现在，我们几乎有了一个完整的项目，但是还不能简单地运行起来，我们需要一个 Makefile 来编译项目。

模块分别使用静态库、动态库编译技术（即有的模块为静态编译、有的模块为动态编译）

根据上述要求，我们既需要静态库，也需要动态库。我决定将数据结构、排序和工具类编译为静态库，可视化模块可能在将来使用 ncurses 库，因此编译为动态库。

## MakeFile 常量定义

首先定义一些常量，如编译器、编译选项、目录等。

```

# Compiler and flags
CXX = g++
CXXFLAGS = -Wall -Wextra -std=c++17 $(shell find $(INCLUDE_DIR) -type d -exec echo -I{})

# Directories
SRC_DIR = src
BUILD_DIR = build
LIB_DIR = lib
INCLUDE_DIR = include
TEST_DIR = test

# Libraries
STATIC_LIBS = $(LIB_DIR)/libdata_structures.a $(LIB_DIR)/libsorting.a $(LIB_DIR)/libuti
DYNAMIC_LIBS = $(LIB_DIR)/libvisualization.so

# Source files
SRC_FILES = $(wildcard $(SRC_DIR)/**/*.cpp) $(SRC_DIR)/main.cpp

# Object files
OBJ_FILES = $(patsubst $(SRC_DIR)/%.cpp, $(BUILD_DIR)/%.o, $(SRC_FILES))

# Target
TARGET = $(BUILD_DIR)/visualAlgo

```

## 目标文件编译

接下来编译目标文件，将目标文件放在 build 目录下。

```

$(BUILD_DIR)/%.o: $(SRC_DIR)/%.cpp
    mkdir -p $(dir $@)
    $(CXX) $(CXXFLAGS) -fPIC -c -o $@ $<

```

## 编译动态库

接下来编译动态库，动态库只有可视化模块一个，因此只需要编译一个动态库即可。将编译出的动态库放在 lib 目录下

```

# Build dynamic library
$(LIB_DIR)/libvisualization.so: $(BUILD_DIR)/visualization/visualize.o
    $(CXX) -shared -o $@ $^

```

## 编译静态库

接下来编译静态库，共有三个静态库，分别是数据结构、排序和工具类。将编译出的静态库放在 lib 目录下。

```
# Build static libraries
$(LIB_DIR)/libdata_structures.a: $(BUILD_DIR)/data_structures/array.o
    ar rcs $@ $^

$(LIB_DIR)/libsorting.a: $(BUILD_DIR)/sorting/bubble_sort.o
    ar rcs $@ $^

$(LIB_DIR)/libutils.a: $(BUILD_DIR)/utils/random_generator.o
    ar rcs $@ $^
```

## make install

现在我们有动态库文件和静态库文件，我们需要将这些文件安装到系统目录中，以便程序使用。根据要求，我们需要将静态库和动态库安装到 /usr/local/lib 目录下，并根据特点指定权限。

```
# Install libraries
install: $(STATIC_LIBS) $(DYNAMIC_LIBS)
    install -m 644 $(STATIC_LIBS) /usr/local/lib/
    install -m 755 $(DYNAMIC_LIBS) /usr/local/lib/
    ldconfig
```

## 可执行文件

现在我们有所有的组件，我们需要将这些组件链接起来，生成可执行文件。

```
# Build target executable
$(TARGET): $(OBJ_FILES) $(STATIC_LIBS) $(DYNAMIC_LIBS)
    $(CXX) $(CXXFLAGS) -o $@ $(OBJ_FILES) $(STATIC_LIBS)
```

我们把库文件安装到了标准路径，这样，在编译时就可以直接链接这些库文件，而不需要指定路径。

## make 目标

现在已经完成了所有的编译工作，我们需要定义一些 make 目标，来简化编译过程。

如 all、clean、run 等。

在 Makefile 的适当位置添加以下代码：

```

all: $(TARGET)

run: $(TARGET) install
    $(TARGET)

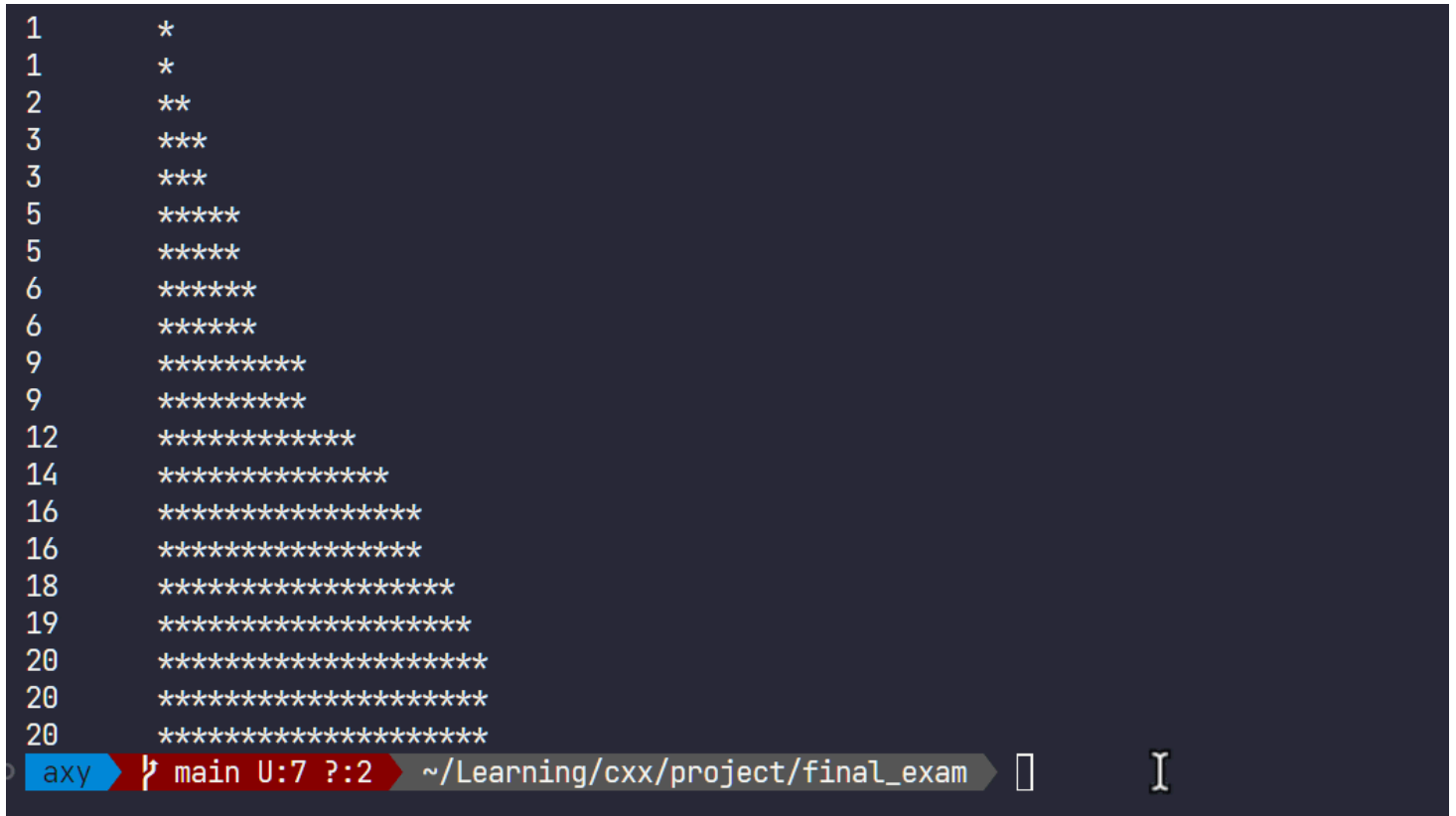
uninstall:
    rm -f /usr/local/lib/libdata_structures.a
    rm -f /usr/local/lib/libsorting.a
    rm -f /usr/local/lib/libutils.a
    rm -f /usr/local/lib/libvisualization.so
    ldconfig

clean:
    rm -rf $(BUILD_DIR) $(LIB_DIR)/*.a $(LIB_DIR)/*.so

```

## 测试

执行 `make run` ( 可能需要 `sudo` ) 即可编译并运行程序。



```

1      *
1      *
2      **
3      ***
3      ***
5      *****
5      *****
6      *****
6      *****
9      *********
9      *********
12     ***********
14     *************
16     *************
16     *************
18     *************
19     *************
20     *************
20     *************
20     *************

```

从上述 gif 动图 ( pdf 版本图片不可动 , 请查看 1.gif 或 doc/report.md 报告 ) 可以看到 , 程序成功运行 , 实现了冒泡排序的可视化。但是由于没有引入 `ncurses` 库 , 因此只能使用 `std::cout` 来进行简单的可视化。



# ncurses 库支持的可视化

## 安装 ncurses

ncurses 库安装比较简单，我从源码编译、构建和安装 ncurses 库。

```
wget https://invisible-island.net/datafiles/release/ncurses.tar.gz
tar -xzvf ncurses.tar.gz
cd ncurses-6.3/
./configure
make
sudo make install
```

## 安装 Google Test

接下来，由于需要对 ncurses 库进行测试和学习，我引入了 Google Test 测试框架，因此需要安装 Google Test。

```
git clone https://github.com/google/googletest.git
cd googletest
mkdir build
cd build/
cmake ..
make
sudo make install
```

安装好 Google Test 后，需要对 Makefile 进行修改，以便能够通过 `make test` 来运行测试。我在 Makefile 中添加了这些内容，如下所示：

```

# Test source files
TEST_SRC_FILES = $(wildcard $(TEST_DIR)/*.cpp)

TEST_TARGET = $(BUILD_DIR)/runTests

$(TEST_TARGET): $(TEST_OBJ_FILES)
    $(CXX) -o $@ $^ $(LDFLAGS) -lgtest -lgtest_main -pthread

$(BUILD_DIR)/%.o: $(TEST_DIR)/%.cpp
    mkdir -p $(dir $@)
    $(CXX) $(CXXFLAGS) -c -o $@ $<

test: $(TEST_TARGET)
    $(TEST_TARGET)

```

## 测试 ncurses 库

处理完成相关依赖后，即可编写测试代码。先后编写了两个测试代码，一个测试 gtest 是否成功运行，一个测试 ncurses 库是否成功运行。

下面是测试 ncurses 库的基本使用，如下所示：

```

TEST(NcursesTest, Initialization) {
    initscr();
    printf("Hello World !!!");
    refresh();
    getch();
    endwin();
}

```

执行 `make test` 即可运行测试，测试结果如下：

```

[=====] Running 2 tests from 2 test suites.
[-----] Global test environment set-up.
[-----] 1 test from GTest
[ RUN      ] GTest.Initialization
[      OK  ] GTest.Initialization (0 ms)
[-----] 1 test from GTest (0 ms total)

[-----] 1 test from NcursesTest
[ RUN      ] NcursesTest.Initialization

```

通过上述测试，我们可以看到 ncurses 库成功运行，可以进行下一步的开发。

## ncurses 可视化

### menu

目标是支持多种排序算法的可视化，首先需要提供一个选择界面， showMenu

```

int showMenu( char*choices[], int n_choices) {
    ITEM **items;
    MENU *menu;
    int i;
    WINDOW *menu_win;

    clear();
    bkgd(COLOR_PAIR(1));

    items = (ITEM **)calloc(n_choices + 1, sizeof(ITEM *));
    for (i = 0; i < n_choices; ++i)
        items[i] = new_item(choices[i], "");

    menu = new_menu(items);

    // Create menu window
    menu_win = newwin(n_choices + 2, 40, 1, 4);
    box(menu_win, 0, 0);
    wbkgd(menu_win, COLOR_PAIR(1));
    keypad(menu_win, TRUE);

    set_menu_back(menu, COLOR_PAIR(1));
    set_menu_fore(menu, COLOR_PAIR(1) | A_REVERSE);

    set_menu_win(menu, menu_win);
    set_menu_sub(menu, derwin(menu_win, n_choices, 38, 1, 1));

    set_menu_mark(menu, "> ");

    mvprintw(LINES - 3, 3, "Select the sorting algorithm:");

    refresh();

    post_menu(menu);
    wrefresh(menu_win);

    int ch;
    while ((ch = wgetch(menu_win)) != '\n') {
        switch (ch) {
            case KEY_DOWN:
                menu_driver(menu, REQ_DOWN_ITEM);
                break;

```

```

        case KEY_UP:
            menu_driver(menu, REQ_UP_ITEM);
            break;
    }
    wrefresh(menu_win);
}

ITEM *cur_item = current_item(menu);
int selected_index = item_index(cur_item);

unpost_menu(menu);
free_menu(menu);
for (i = 0; i < n_choices; ++i)
    free_item(items[i]);
delwin(menu_win);

return selected_index;
}

```

在这个函数中，提供一个选择界面，用户可以通过上下键选择排序算法，按回车键确认选择。下面这张动图展示了这个选择界面（pdf文件无法展示动图）。

```

> Bubble Sort
  Selection Sort
  Insertion Sort
  Quick Sort
  Merge Sort
  Merge Sort
  Merge Sort
  Exit

```



Select the sorting algorithm:

## Array

下一步是把数组内容显示到屏幕上，通过 `displayArray` 函数实现。

```

void displayArray(Array& arr, int delay) {

    int height, width;
    getmaxyx(stdscr, height, width);

    int sub_height = arr.getSize() * 2 + 1;
    int sub_width = 70;
    int start_y = (height - sub_height) / 2;
    int start_x = (width - sub_width) / 2;

    WINDOW *subwin = newwin(sub_height, sub_width, start_y, start_x);
    wbkgd(subwin, COLOR_PAIR(3));
    box(subwin, 0, 0);

    wattr_on(subwin, COLOR_PAIR(3), NULL);

    for (int i = 0; i < arr.getSize(); ++i) {
        mvwprintw(subwin, i * 2 + 1, 1, "%d", arr.get(i));
        for (int j = 0; j < arr.get(i); ++j) {
            mvwaddch(subwin, i * 2 + 1, j + 5, '\u2598' );
            // mvwaddstr
        }
    }

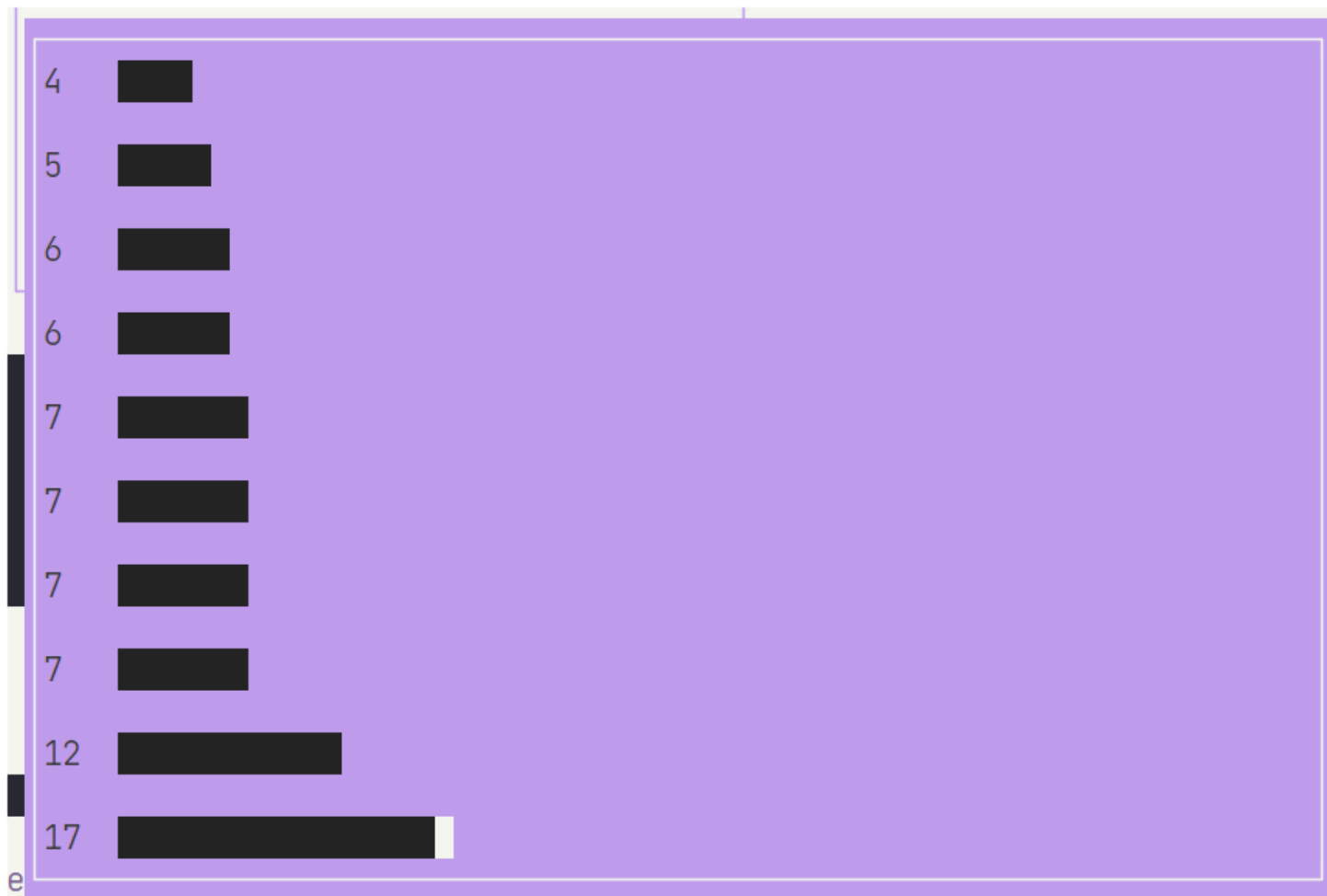
    wattr_off(subwin, COLOR_PAIR(3), NULL);

    refresh();
    wrefresh(subwin);

    std::this_thread::sleep_for(std::chrono::milliseconds(delay));
}

```

这个函数将会创建一个新的窗口，为它添加一个美丽的边框，然后将数组内容显示到屏幕上，并在每一次变化后延迟一段时间，以便观察。下面这张动图展示了这个函数的效果（pdf文件无法展示动图）。



## 更多排序算法

这一部分就很简单了，只需要实现更多的排序算法，然后在 main 函数中添加选项即可。

在这里，我选择了五种常见的排序算法

```
char* choices[] = {  
    "Bubble Sort",  
    "Selection Sort",  
    "Insertion Sort",  
    "Quick Sort",  
    "Merge Sort",  
    "Exit"  
};
```

这是一个十分灵活的设计，只需要在 choices 数组中添加排序算法的名字，然后在 main 函数中添加对应的处理即可。十分便于之后的维护。

之后再在 main 函数中添加对应的处理即可，如下所示：

```
while (1) {
    Array arr(array);
    selected = showMenu(choices, sizeof(choices) / sizeof(char*));

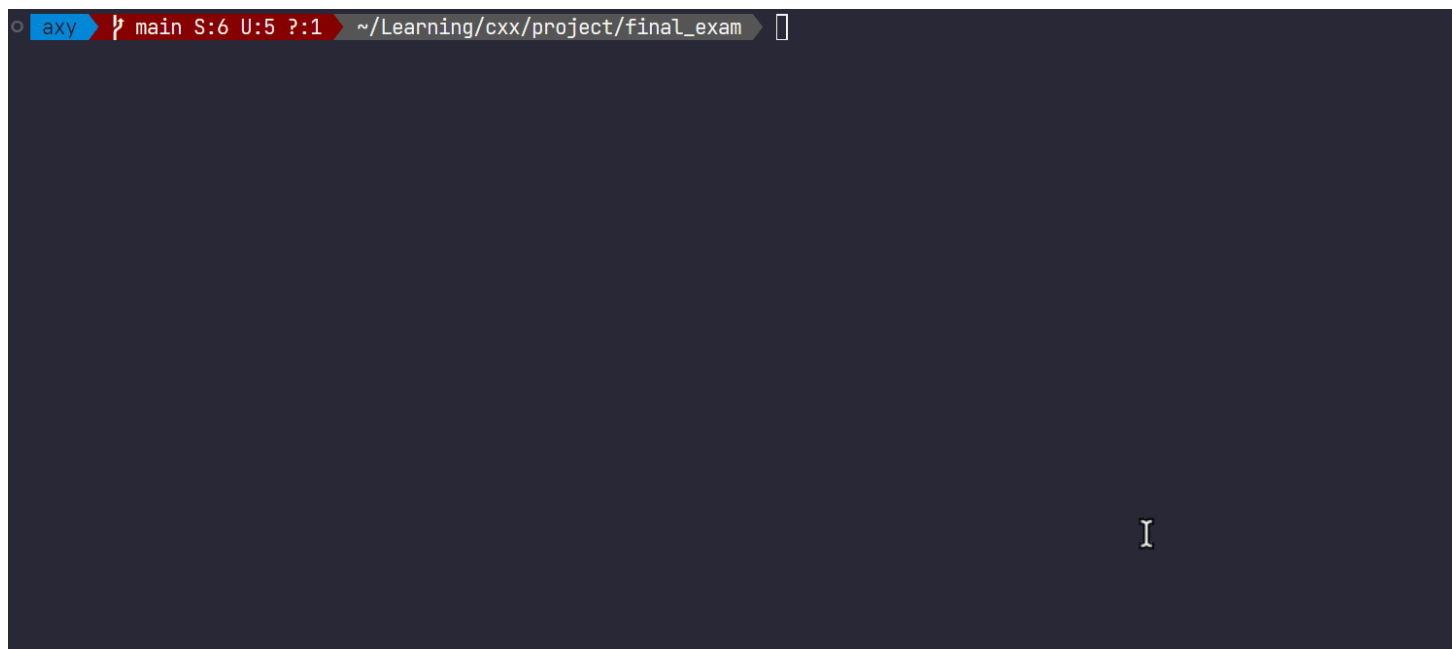
    switch (selected) {
    case 0:
        bubbleSort(arr);
        break;
    case 1:
        selectionSort(arr);
        break;
    case 2:
        insertionSort(arr);
        break;
    case 3:
        quickSort(arr);
        break;
    case 4:
        mergeSort(arr);
        break;
    default:
        endUi();
        return 0;
        break;
    }
    sleep(1);
}
```

## 总结

现在代码已经完成了，整体使用了 ncurses 库来实现可视化，使用了 Google Test 来进行单元测试，使用了 Makefile 来编译项目，并且同时支持静态库（数据结构、排序算法和工具类）和动态库（可视化）。

现在，可以通过 make 来编译项目，通过 make run 来运行项目( sudo )，通过 make test 来运行测试。运行效果如下：





整个项目的结构清晰，易于维护，十分灵活。整体项目结构如下：

```
.
├─ Makefile
├─ README.md
├─ build
│   ├── data_structures
│   │   └─ array.o
│   ├── main.o
│   ├── sorting
│   │   ├── bubble_sort.o
│   │   └─ sort.o
│   ├── utils
│   │   └─ random_generator.o
│   ├── visual_algo
│   └─ visualization
│       ├── ui.o
│       └─ visualize.o
├─ doc
│   ├── image
│   │   ├── 1.gif
│   │   ├── 2.gif
│   │   └─ 3.png
│   └─ report.md
├─ include
│   ├── data_structures
│   │   └─ array.h
│   ├── sorting
│   │   └─ sort.h
│   ├── utils
│   │   └─ random_generator.h
│   └─ visualization
│       ├── ui.h
│       └─ visualize.h
├─ lib
│   ├── libdata_structures.a
│   ├── libsorting.a
│   ├── libutils.a
│   └─ libvisualization.so
├─ src
│   ├── data_structures
│   │   └─ array.cpp
│   ├── main.cpp
│   ├── sorting
│   │   └─ sort.cpp
```

```
|   ├── utils
|   |   └── random_generator.cpp
|   └── visualization
|       ├── ui.cpp
|       └── visualize.cpp
└── test
    ├── gtest_test.cpp
    ├── main_test.cpp
    ├── ncurses_test.cpp
    └── ui_test.cpp
```