Lab 6 - 复数排序

Task 0 - Warm up

本Task使用代码文件`task0.c`。

0.1 - 定义复数

首先，我们需要利用结构体定义“复数"。

```
//! try to define complex number with structure
//! using double type
typedef struct Complex
{
    double real;
    double imag;
} complex;
```

0.2 - 结构体的存储对齐问题

```
struct Complex2
{
    float real;
    double imag;
    char x; // 复数标识 c
} ;
struct Complex3
{
    char x; // 复数标识 c
    float real;
    double imag;
} ;
struct Complex4
{
    float real;
    char x; // 复数标识 c
    double imag;
};
```

任务：使用`sizeof`打印这三个`struct`的大小，解释原因。

注意，本次实验代码将会使用到`math.h`中的`sqrt`函数，因此需要在编译时使用`-lm`参数链接数学库（如果使用`gcc`编译器），或是在`CMakeLists.txt`中添加`target_link_libraries(some_target m)`。

若使用gcc编译器，可以使用如下命令编译（将文件名替换为实际的文件名）：

```
gcc some_code.c -o output_file -lm
```

Task 1 - 复数的加法

本Task使用代码文件`task1.c`。

输入两个复数，输出两个复数的和（存储至传入的指针`result`所指向的“复数”变量）。

任务：完成函数`void addNumbers(complex c1, complex c2, complex *result);`。输入x+yi中的x和y，验证函数的正确性。

一些典型的测试用例如：

- 1+0i和2+0i
- 0+3i和0+5i
- 3+4i和5+6i

Task 2 - 复数的乘法

本Task使用代码文件`task2.c`。

输入两个复数，输出两个复数的乘积。

任务：完成函数`void multiNumbers(complex c1, complex c2, complex *result);`。输入x+yi中的x和y，验证函数的正确性。

Task 3 - 顺序输出复数数组

本Task使用代码文件`task3_4.c`。

任务：完成函数`void ivec_print_complex(complex arr[], int n)`，顺序输出复数数组，用于后续检查复数排序算法的输出。

实现方式可以参考输出整数数组的代码：

```c
//! 顺序输出整数数组的函数，可用于检查整数排序输出
void ivec_print_p1(int *x, int n)
{
    int i;
    for (i = 0; i < n; i++)
    {
        printf("%d \t", *x++);
    }
    printf("\n");
}
```

Task 4 - 生成随机复数

本Task使用代码文件`task3_4.c`。

任务：完成函数`int randnum(complex arr[], int dim, int range)`，生成dim个随机复数，实部和虚部的范围均为range。

注意：此函数可能在短时间内被多次调用。如果在此函数内部设置随机种子，可能导致多次调用使用相同的随机种子重置随机数生成器，从而生成相同的随机数。因此，需要在此函数外部一次性地设置随机种子。

实现方式可参考生成随机整数数组的代码：

```c
// 生成随机整数数组的函数
// int dim      // 生成多大的数组
// int range    // 数组的元素大小从1到range
// 产生长度为dim，数值在［1-range］区间的随机数
int randnum(int *arr, int dim, int range)
{
    for (int i = 0; i < dim; i ++)
    {
        *arr = rand() % range + 1;  //产生1-range的随机数
    }
    return 0;
}
```

Task 5 - 排序算法（复数版）

本Task使用代码文件`task5.c`。

任务：修改上次Lab整数排序中的某一种排序算法，使其对复数按模（$\sqrt{x^2+y^2}$）进行排序。（此处将以插入排序为例，你可以选择其他排序算法，若选择了其他排序算法，整数排序的比较对象请根据实际情况选择）

注：`sqrt`函数需要用到数学包：`math.h`。编译时请不要忘记链接数学库。

实现方式可以参考整数插入排序的代码：

每次調用都算modulus，還是先算好再排序？
後者

```c
//! 整数插入排序
void insertion_sort(int arr[], int len)
{
    int i,j,key;
    for (i=1;i<len;i++)
    {
        key = arr[i];
        j=i-1;
        while((j>=0) && (arr[j]>key))
        {
            arr[j+1] = arr[j];
            j--;
        }
        arr[j+1] = key;
    }
}
```

Homework: 测试排序算法

作业:

1. 在单个代码文件`lab6.c`中整合Lab 5中的整数插入排序算法与Lab 6中的复数插入排序算法（以及相关工具函数）。
2. 在main函数中，执行并比较Lab 5中整数插入排序与本次Lab 6中复数插入排序的运行时间：针对10/100/1000/10000维整数与复数排序各重复100次，分别求平均执行时间，并将其打印输出。
3. 将代码输出结果记录在实验报告中，并画图对比，分析整数、复数两种算法在执行效率上的区别。

提交文件:

- `lab6.c`：包含整数插入排序、复数插入排序、相关工具函数，以及执行两种算法不同维度多次计算并输出每种情况平均运行时间的main函数。
- `report.pdf`：实验报告，包含代码运行输出（不同情况的执行时间）、图表对比，以及两种算法的执行效率分析。