程序设计专题大作业2

**二元n次多项式**

组号 6

1号组长：学号 3180102688 姓名 刘政泽 联系电话 18923232336

2号组员：学号 3180101346 姓名 卢晨韵 联系电话 18358865269

3号组员：学号 3160101437 姓名 方佳豪 联系电话 18606668402

## 目录

[1 主要数据结构说明 3](#_Toc4161763)

[1.1 文件组织结构 3](#_Toc4161764)

[1.2 主要数据类型 3](#_Toc4161765)

[1.3 主要数据结构 4](#_Toc4161766)

[1.4 数据结构示例说明 5](#_Toc4161767)

[2 算法设计 5](#_Toc4161768)

[2.1 主程序 5](#_Toc4161769)

[2.2 生成器 5](#_Toc4161770)

[2.3 搜索 7](#_Toc4161771)

## 1 主要数据结构说明

在本程序中，应课程要求及代码规范要求，我们重定义了基本的数据类型。本程序制作的迷宫为的简单迷宫。通过对比，最后我们选择了的二元矩阵来储存迷宫。

### 1.1 文件组织结构

|  |  |
| --- | --- |
| 文件名 | 用途 |
| Labyrinth | 整个工程的主函数 |
| Labyrinth\_Base | 定义工程所需的数据类型和通用函数 |
| Labyrinth\_Generator | 该模块用于迷宫的随机生成 |
| Labyrinth\_Search | 该模块用于寻找迷宫的通路 |
| Labyrinth\_Render | 该模块用于可视化迷宫及迷宫通路路径 |

我们的工程一共分为五个子程序，每个程序都有相应的.h头文件。其中Labyrinth为我们的主程序，在这个程序我们调用各个模块来实现功能。

### 1.2 主要数据类型

|  |  |  |
| --- | --- | --- |
| 类型名称 | 数据类型 | 用途 |
| block\_t | enum | 表征迷宫中点的状态，  其中Vacancy代表此点为通路，Barrier代表此点为障碍 |
| bool\_t | enum | 逻辑数据类型，False代表假，True代表真 |
| direction\_t | enum | 代表迷宫中前进的方向，  Right代表右，Down代表下，Left代表左，Up代表上 |
| int32\_t | int | 代表int |
| char\_t | char | 代表char |
| maze\_t | struct | Maze\_t结构体记录了迷宫的尺寸、形状和查找出路时的访问记录 |
| position | struct | 结构体含x, y，代表坐标 |

### 1.3 主要数据结构

|  |  |  |  |
| --- | --- | --- | --- |
| 结构名称 | 结构大小 | 结构类型 | 用途 |
| maze\_t.map | size\*size | block\_t | 记录生成的迷宫，maze\_t.map[i][j]==Vacancy则表示第i行第j列的点是通路，反之则为障碍 |
| maze\_t.vis | size\*size | bool\_t | 用于记录寻找出路时已访问过的通路，若maze\_t.vis[i][j]==True,则代表第i行第j列的点已访问过，反之则为未访问 |
| maze\_t. path | size\*size | bool\_t | 记录出路，若maze\_t. path [i][j]==True，则代表最后的出路包含第i行第j列的点，反之则为不含 |
| position.x/  position.y | 1 | int32\_t | 记录position坐标的行值和列值，其中x代表行值，y代表列值 |
| father | size\*size | int32\_t | 记录并查集中迷宫各点联通的根节点，其中每个点有一个ID,其ID为迷宫的行坐标\*size+列坐标，即x\*size+y.如在一个9\*9的迷宫中，father[2\*9+1]=father[19]=1,意为第二行第一列的点与ID为1也就是第一行第一列的点是相通的。 |

### 1.4 数据结构示例说明

如图为一个尺寸17\*17的迷宫。其中第一、十七行，第一、十七列作为迷宫外围均为Barrier，因而不储存在迷宫的数据结构中，实际迷宫的尺寸为15\*15，将储存在maze\_t.map[15][15]中，图中灰色的方块代表障碍，黑色的部分代表通路，如其中maze\_t.map[1][1]代表图中的红点起点，因为是通路，所以类型为Vacancy。如maze\_t.map[1][2]即代表红色方块相邻右边的方块，其颜色是灰色，因而其类型为Barrier。

## 2 算法设计

### 2.1 模块简介

程序由三个模块组成：**交互**模块（Interact）、**识别**模块(Identify）和**计算**模块（Render）。其中识别模块负责将字符串形式的多项式转换为链表形式并储存；计算模块，顾名思义，对传入两个多项式进行相应的运算。最后，交互模块负责实现控制台界面，接受用户输入的多项式，并调用识别模块和计算模块进行处理，输出多项式。

### 2.2 识别模块（Identify）

识别模块接收一个字符串参数，并返回转换后的链表头节点。对于接收的字符串形式多项式，我们通过while循环，以项（term）为单位进行处理。对于某一项，重复调用ReadNum函数，依次读取该项系数、x的幂次和y的幂次。然后调用Creat函数。Creat函数的作用是将系数和幂次等数据存放到新申请的结点里，并将该结点插入链表头节点后。

由于多项式输入的格式不规范问题，在进行单项处理时，要考虑到许多特殊情况，如x^2y^2（系数隐去）、3x^2y（幂次隐去）和2y（无x）等。出现上述特殊情况时，我们分别将系数设为1，y的幂次设为1，x的幂次设为0，以追求结点储存格式的一致，从而提高程序的兼容性。

在最终返回链表头节点前，识别模块还将调用arrange函数，完成对链表的排序。arrange函数的实现逻辑比较简单：将链表的结点信息依次复制到临时数组中，调用qsort函数完成对临时数组的排序后，再顺次对链表结点信息完成替换。

### 2.3 计算模块 (Calculate)

#### 加法ADD:

加法函数ADD接收（以链表头节点形式）两个多项式，并以相同形式返回两个多项式的和。由于识别模块在最后对多项式进行了按项排序，因此对两个有序多项式的加法，可以类比归并排序中合并两个有序数组的过程。具体过程如下：

1 将a, b分别指向两个多项式的首项

2 当a, b均不为空时，执行第3步，否则跳到第5步

3 调用Powercmp函数比较a, b幂次的大小。优先比较总幂次，其次是x的幂次，最后为y的幂次。在此比较关系下，若a, b不相等，则将较大者链接到代表运算结果的多项式尾部，并使其指向原属多项式的下一项；若a, b幂次完全相同且相加不为零，调用Creat函数合并同类项。

4 回到第2步

5 将a, b中不为空者链接到结果多项式尾部。

#### 减法 SUBSTRACT:

A – B = A + ( -B )。因此加法实现后，减法就变得十分简单。只需要对减去的多项式预先进行系数取反（这个过程由coreverse实现），再调用加法即可。

#### 乘法 MULTIPLY:

多项式满足乘法分配律，因此我们让第二个多项式中的每一项分别与第一个多项式相乘（product函数）。相乘后多项式依然是有序的，所以我们只需要再次利用ADD函数，将这些结果依次相加，即可得到两个多项式的乘积。

### 2.4 交互模块 (Interact)

首先，我们仿照计算器的交互逻辑，对题目所给的交互方式进行了如下的改进。

1 删除了“输入多项式”这一指令。输入运算符后，紧接着用户自然会输入多项式，不需要事先输入这条多余的指令

2 运算完成后自动输出结果，进一步简化操作

3 加入了清零指令

改进后交互界面支持的操作如下：

1-清除多项式

2-多项式加（+）

3-多项式减（-）

4-多项式乘（\*）

5-多项式输出（手动）

0-退出

像计算器一样，程序中始终储存着当前运算的结果，等待与下一个输入的多项式进行运算。为了继续提高兼容性，若用户未输入操作符就输入多项式，程序默认用户用该多项式覆盖程序中储存的“当前值”。

我们用while循环搭配switch-case语句实现这个交互系统。循环开始前，将多项式置为0，操作符置为2（+），并提示用户输入第一个多项式。循环内部先处理用户输入的多项式并输出，然后读入下一个操作符。循环的终止条件是操作符为0。