**《程序设计专题》实验报告1**

**大程序开发基础：函数与程序结构、递归**

## 1．实验目的

通过实验：

1）掌握大程序中的函数与程序结构设计思想方法；

2）掌握递归的应用；

3）走迷宫综合程序设计方法。

## 实验内容

**阅读课件，阅读教材第10章内容。**

1. 完成本报告中的补充习题
2. 多文件实现走迷宫综合小程序设计
3. 打包本实验报告(转为pdf)、源代码，压缩为.zip格式，上传至学在浙大。上传的压缩文件命名为学号\_姓名\_HW1.zip。

## 3. 实验记录解答

**3.1 补充习题**

**解答：**

**1、**a=98, b=11, c=32766

**2、**19

**3、**

a[n]=’\0’

int x, char a[]

t[x % 16]

n + 1

**=======================================**

**补充题：**

1. 以下程序的输出结果是\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_。

#include <stdio.h>

int g=96;

void gf(void)

{ g++;

}

int df(void)

{

int g=10;

g++;

return g;

}

int \* sf(void)

{

static int g=20;

g++;

return &g;

}

void main()

{

int a, b, c, \*p;

gf(); gf(); a=g;

b=df(); b=df();

p=sf(); \*p=32765; p=sf(); c=\*p;

printf("a=%d, b=%d, c=%d", a, b, c);

}

2. 假定运行以下程序时输入

3+4\*5-3/2=

则程序的输出结果是\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_。

#include <stdio.h>

int calc(void)

{ int x; char p;

scanf("%d %c", &x, &p);

if(p == '=')

return x;

switch(p)

{

case '+': return x + calc();

case '-': return x - calc();

case '\*': return x \* calc();

case '/': return x / calc();

}

}

void main()

{

printf("%d", calc());

}

3. hex()是一个递归函数，该函数的功能是通过递归调用把一个长整数转化成十六进制字符串并返回所得十六进制数的位数。main()输入长整数x，调用hex()对它进行转化（转化结果存放在字符数组a中），最后输出结果。例如：输入13，则返回n=1输出D；输入100，则返回n=2输出64；输入65535，则返回n=4输出FFFF。

#include <stdio.h>

void main()

{

char a[9];

int n;

unsigned long int x;

scanf("%lu", &x);

n = hex(x, a);

(1) ;

puts(a);

}

int hex( (2) )

{

char t[]="0123456789ABCDEF";

int n;

if(x<16)

{

a[0] = t[x];

return 1;

}

n = hex(x/16, a);

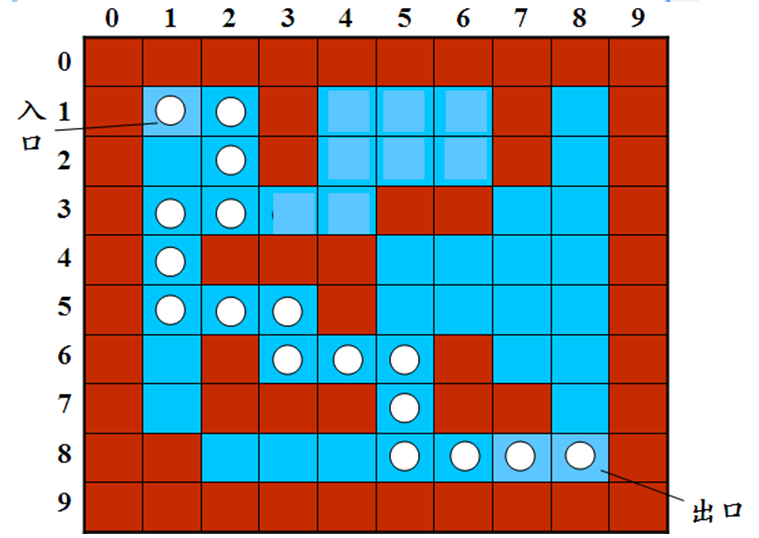
a[n] = (3) ;

return (4) ;

}

**3.2走迷宫综合小程序设计**

**用递归求解迷宫问题**

****

* **求迷宫路径算法的基本思想：**
  + **若当前位置“可通”，则纳入路径，继续前进；**
  + **若当前位置“不可通”，则后退，换方向（按东南西北的顺序）继续探索；**
  + **若四周“均无通路”，则将当前位置从路径中删除出去**

**设定当前位置的初值为入口位置；**

**do{**

**if (当前位置可通)**

**{**

**将当前位置插入栈顶； 　　　　　　// 纳入路径**

**若该位置是出口位置，则算法结束；**

**// 此时栈中存放的是一条从入口位置到出口位置的路径**

**否则切换当前位置的东邻方块为新的当前位置；**

**}**

**else**

**{**

**若栈不空且栈顶位置尚有其他方向未被探索，**

**则设定新的当前位置为: 沿顺时针方向旋转找到的栈顶位置的下一相邻块；**

**若栈不空但栈顶位置的四周均不可通，**

**则{ 删去栈顶位置； 　　　　　// 从路径中删去该通道块**

**若栈不空，则重新测试新的栈顶位置，**

**直至找到一个可通的相邻块或出栈至栈空；**

**}**

**}**

**} while (栈不空)；**

**要求：采用多文件的设计方法实现，给出详细的文件模块设计与实现过程。比如可以分为以下四个模块：**

* **main：调用其他模块完成输入计算和输出；**
* **init模块：初始化迷宫（采用算法随机初始化或者预设一个迷宫）;**
* **probe模块：探索迷宫的出路，要求用递归实现，管理路径数据；**
* **print模块：输出迷宫，输出逃出迷宫的路径。print模块可以选择在控制台中用可见字符打印出迷宫的形状和最终的解决路径。可以选择简单一点，仅仅打印出行进路径，但实验报告中应给出原始迷宫图，程序运行结果截图。**

**文件模块设计内容包括介绍每个模块完成的功能、维护的数据结构（如果有），实现过程描述各个模块对数据的操作（算法），算法可以用伪代码形式（如上利用栈的代码段）来体现，或用语言来描述。**

**源代码中应当有注释，以方便它人阅读和理解。注意代码规范，例如缩进、变量命名、函数命名等。**

**解答：**

**3.2.1 文件模块设计**

**main.c**：主程序，调用其他模块，完成程序功能。

**maze.h**：定义了全局使用的数据结构maze\_t、tile\_t。其中maze\_t中的tile是一个动态分配的二维数组，用于保存迷宫数据；tile\_t定义了一些迷宫格子的字符常量，包括空地、围墙、路径、起点和终点，便于更改和移植。

**init.h**：定义了模块内部使用的迷宫实际数据块buf、用于Kruskal算法的并查集fat、用于Kruskal算法的边表edges。这些数据结构都是动态分配空间的，减少浪费。

**init.c**：主要实现了迷宫数据的分配和迷宫的生成，利用了最小生成树的Kruskal算法完成。此外还实现了跨平台优美显示围墙字符的代码页初始化（用于Windows），以及fallback功能。

**print.c**：实现了迷宫的输出，在走迷宫前后各运行一次。

**probe.h**：定义了模块内部使用的位移数组dx、dy，便于走迷宫的实现。

**probe.c**：实现了确定迷宫起点终点，找到迷宫的路径。

**Makefile**：使用Make构建程序。

**clean.bat**：在Windows下清理二进制文件。

**3.2.2 实现过程**

**main.c**：调用其他模块，依次完成代码页初始化、询问用户迷宫大小、建立图、用Kruskal算法生成迷宫、确定迷宫起点终点、输出迷宫、探索迷宫、输出迷宫路径。

**init.c**：

* **mazeAlloc**先分配迷宫数据块，再分配、计算二维数组tile的头指针，完成动态分配二维数组。
* **getSize**询问用户迷宫大小，调整行列数不小于5，使迷宫有实际意义，并调用mazeAlloc分配。
* **createGraph**初始化并查集，分配边表，在点集间连边（详见代码注释），并把边表随机改变顺序。
* **Kruskal**执行该算法，利用并查集快速判断一条边的两个点是否位于同一棵树中，若不是则加入生成树，并把两个点对应的树合并。
* **getfat**是并查集核心算法，递归寻找一个点的祖先，并执行路径压缩以大幅提高效率。
* **randn**随机生成[0, n)间的整数，解决了Windows下rand最大不超过32767的问题，而\*nix下则为231-1，且能跨平台使用。
* **initGraph**在Windows下初始化代码页为UTF-8，并实现了fallback。

**print.c**：实现了逐行输出迷宫数据，并能在未fallback情况下将围墙字符“#”显示为“▌”，更加美观。

**probe.c**：

* **findStartEnd**在第2行和倒数第2行找到迷宫的起点终点。
* **probe**递归探索迷宫，每次向4个方向探索新格子，若到达终点则完成探索，若新格子是空地则标记为路径，走过去递归探索，若不能到达终点（死路）则回溯，将新格子恢复成空地，继续探索。

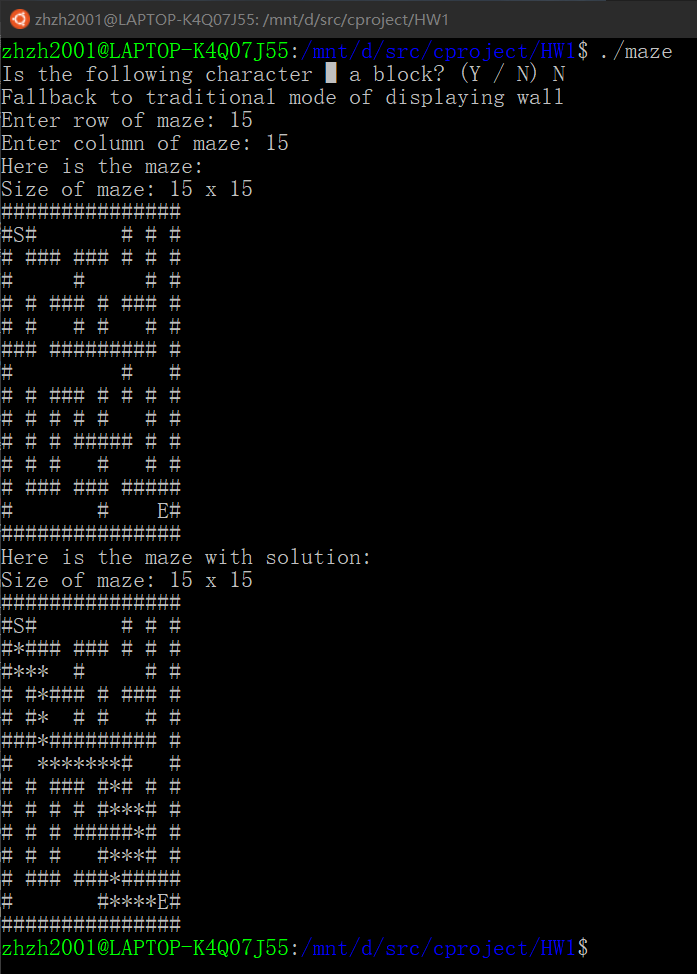
**3.2.3 运行结果展示**

****

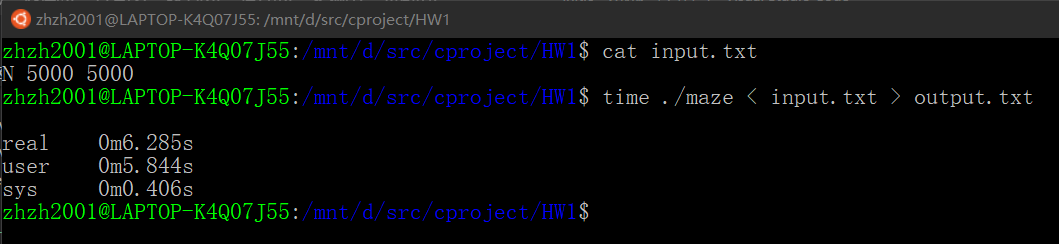
Windows下编译运行，迷宫大小15 x 15，使用UTF-8显示。



WSL（可视为Linux）下编译运行，默认使用UTF-8编码。



WSL下运行，使用fallback。



本程序能处理超大的迷宫，例如生成、探索5000 x 5000的迷宫，开启O2优化，使用fallback，用时如图所示。效率还有待提高。