**中南大学**

数据结构试验报告

题 目 栈、队列与递归算法设计

学生姓名 张子洋

学 号 8208221223

指导老师 郑瑾

学 院 计算机学院

专业班级 计算机与通信类2212班

完成时间 2023.5

指导老师评定 签名

1.栈和队列操作的实现（验证性实验）

1. 需求分析

要求：

(1) 建立一个顺序栈。

(2) 建立一个循环顺序队列。

(3) 分别实现栈和队列的基本操作

1. 概要设计
2. 为了实现程序功能，需要定义栈和队列的抽象数据类型。

ADT Stack {

数据对象：

D＝{ ai | ai ∈ElemSet, i=1,2,...,n, n≥0 }

数据关系：

R1＝{ <ai-1, ai >| ai-1, ai∈D, i=2,...,n }

约定an 端为栈顶，a1 端为栈底。

typedef struct{

selemtype \*base; //尾指针

selemtype \*top; //头指针

int stacksize; //栈长

}sqstack; //栈结构

基本操作：

InitStack(&S)

操作结果：构造一个空栈S

DestroyStack (&S)

初始条件：栈S已存在

操作结果：销毁栈S

ClearStack (&S)

初始条件：栈S已存在

操作结果：将栈置为空栈

StackEmpty(&S)  
 初始条件：栈S已存在

操作结果：若S为空栈则返回TRUE，否则返回FALSE

StackLength(&S)

初始条件：栈S已存在

操作结果：返回S中数据元素的个数，即栈的长度

GetTop(&S, &e)

初始条件：栈S已存在

操作结果：用e返回S的栈顶元素

Push(&S, e)

初始条件：栈S已存在  
 操作结果：插入元素e为新的栈顶元素

Pop(&S, &e)  
 初始条件：栈S已存在且非空

操作结果：删除S的栈顶元素，并用e返回其值

StackTraverse(&S, visit())

初始条件：栈S已存在且非空

操作结果：从栈底到栈顶依次对S的每个数据元素调用函数visit()，一旦visit()失败，则操作失败

}ADT Stack

ADT Queue {

数据对象：

D＝{ai | ai∈ElemSet, i=1,2,...,n, n≥0}

数据关系：

R1＝{ <a i-1, ai > | ai-1, ai ∈D, i=2,...,n}

约定其中a1 端为队列头，an 端为队列尾

typedef struct{

qelemtype \*base;

int front;

int rear;

}sqqueue; //队列结构

基本操作：

InitQueue(&Q)  
 操作结果：构造一个空队列Q。  
 DestroyQueue(&Q)  
 初始条件：队列Q已存在。  
 操作结果：队列Q被销毁，不再存在。  
 QueueEmpty(Q)  
 初始条件：队列Q已存在。  
 操作结果：若Q为空队列，则返回TRUE，否则返回FALSE。

QueueLength(Q)  
 初始条件：队列Q已存在。  
 操作结果：返回Q的元素个数，即队列的长度。

GetHead(Q,&e)  
 初始条件：Q为非空队列。  
 操作结果：用e返回Q的队头元素。

ClearQueue(&Q)  
 初始条件：队列Q已存在。  
 操作结果：将Q清为空队列。   
 EnQueue(&Q, e)  
 初始条件：队列Q已存在。  
 操作结果：插入元素e为Q的新的队尾元素

DeQueue(&Q, &e)  
 初始条件：Q为非空队列。  
 操作结果：删除Q的队头元素，并用e返回其值。

QueueTraverse(Q, visit())

初始条件：队列Q存在且非空

操作结果：从栈底到栈顶依次对S的每个数据元素调用函数visit()，一旦visit()失败，则操作失败

} ADT Queue

1. 程序栈操作的实现包含11个函数：

void InitStack(SqStack \*&s)

void DestroyStack(SqStack \*&s)

bool StackEmpty(SqStack \*&s)

bool Push(SqStack \*&s, int &e)

bool Pop(SqStack \*&s, int &e)

bool GetTop(SqStack \*s, int &e)

int StackLength(SqStack \*s)

1. 程序队列操作的实现包含11个函数：

void InitQueue(SqQueue \*&q)

void DestroyQueue(SqQueue \*&q)

bool QueueEmpty(SqQueue \*q)

bool enQueue(SqQueue \*&q, int e)

bool deQueue(SqQueue \*&q, int &e)

int QueueLength(SqQueue \*q)

return len;

}

1. 详细设计

栈操作的实现.c 详细源代码如下：

1. //栈的各种操作实现
2. # include <stdio.h>
3. # include <stdlib.h>
4. # include <string.h>
5. # define MaxSize 100
6. typedef struct {
7. int data[MaxSize];
8. int top;
9. } SqStack;
10. void InitStack(SqStack \*&s) {
11. s = (SqStack \*)malloc(sizeof(SqStack));
12. s->top = -1;
13. }
14. void DestroyStack(SqStack \*&s) {
15. free(s);
16. }
17. bool StackEmpty(SqStack \*&s) {
18. return (s->top == -1);
19. }
20. bool Push(SqStack \*&s, int &e) {
21. if (s->top == MaxSize - 1)
22. return false;
23. s->top++;
24. s->data[s->top] = e;
25. return true;
26. }
27. bool Pop(SqStack \*&s, int &e) {
28. if (s->top == -1)
29. return false;
30. e = s->data[s->top];
31. s->top--;
32. return true;
33. }
34. bool GetTop(SqStack \*s, int &e) {
35. if (s->top == -1)
36. return false;
37. e = s->data[s->top];
38. return true;
39. }
40. int StackLength(SqStack \*s) {
41. return s->top + 1;
42. }
43. int main() {
44. SqStack \*s;
45. InitStack(s);
46. int i, k;
47. if (StackEmpty(s))
48. printf("栈空\n");
49. else
50. printf("栈不空\n");
51. printf("请输入十个数用于测试：\n");
52. for (int j = 0; j < 10; j++) {
53. scanf("%d", &i);
54. Push(s, i);
55. }
56. i = 0;
57. if (StackEmpty(s))
58. printf("此时栈空\n");
59. else
60. printf("此时栈不空\n");
61. if (GetTop(s, k))
62. printf("栈顶元素：%d\n", k);
63. else
64. printf("出现错误\n");
65. int n;
66. int len = StackLength(s);
67. printf("栈长：%d", len);
68. printf("\n需要出栈元素个数（小于10）：");
69. scanf("%d", &n);
70. int g;
71. for (int j = 1; j <= n; j++) {
72. Pop(s, g);
73. }
74. printf("此时栈为：");
75. for (int i = s->top ; i >= 0; i--) {
76. printf("%d ", s->data[i]);
77. }
78. DestroyStack(s);
79. return 0;

}

//队列

# include <stdio.h>

# include <stdlib.h>

# define MaxSize 1000

typedef struct {

int data[MaxSize];

int front, rear;

} SqQueue;

void InitQueue(SqQueue \*&q) {

q = (SqQueue \*)malloc(sizeof(SqQueue));

q->front = q->rear = 0;

}

void DestroyQueue(SqQueue \*&q) {

free(q);

}

bool QueueEmpty(SqQueue \*q) {

return (q->front == q->rear);

}

bool enQueue(SqQueue \*&q, int e) {

if ((q->rear + 1) % MaxSize == q->front)

return false;

q->rear++;

q->data[q->rear] = e;

return true;

}

bool deQueue(SqQueue \*&q, int &e) {

if (q->front == q->rear)

return false;

q->front++;

e = q->data[q->front];

return true;

}

int QueueLength(SqQueue \*q) {

int len = 0, \*p;

for (int p = q->front; p != q->rear; p++) {

len++;

}

return len;

}

/\*1 2 3 4 5 6 7 8 9 10\*/

int main() {

SqQueue \*s;

InitQueue(s);

int i, k;

if (QueueEmpty(s))

printf("队空\n");

else

printf("队不空\n");

printf("请输入十个数用于测试：\n");

for (int j = 0; j < 10; j++) {

scanf("%d", &i);

enQueue(s, i);

}

i = 0;

if (QueueEmpty(s))

printf("此时队空\n");

else

printf("此时队不空\n");

int n;

int len = QueueLength(s);

printf("队长：%d", len);

printf("\n需要出队元素个数（小于10）：");

scanf("%d", &n);

int g;

for (int j = 1; j <= n; j++) {

deQueue(s, g);

}

printf("此时队为：");

for (int u = s->front + 1; u <= s->rear; u++) {

printf("%d ", s->data[u]);

}

DestroyQueue(s);

return 0;

}

4.调试分析

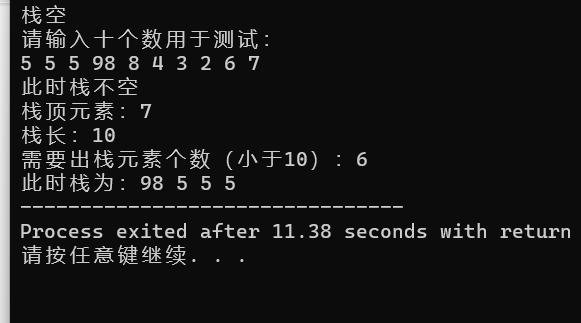
1. 采用IDE中自带的调试功能进行调试，手动添加断点和查看程序。
2. 对设计和编码的讨论和分析。该程序实现了顺序栈的操作。分析程序代码的质量，主要从以下几个方面考虑。 

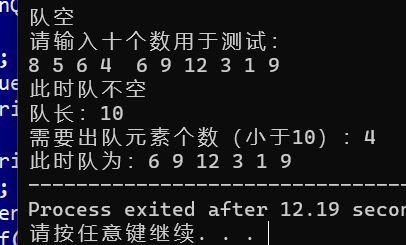
* 正确性。在一定的数据范围内，该程序能实现所需功能，所以正确性是没有问题的。
* 健壮性。在一定的数据输入范围内，该程序能较好的实现链表的操作。但是如果输入数 据非法，该程序还是可能会产生一些预想不到的输出结构，或是不做任何处理。所以， 该程序的健壮性有待进一步的提高。要综合考虑一些情况，当输入有误时，应返回一个 表示错误的值，并中止程序的执行，以便在更高的抽象层次上进行处理。

1. 使用说明

按照屏幕提示，输入数个int型数字，按下ENTER键后，根据屏幕提示进行输入，即可得到想要的结果。

1. 测试程序的运行结果





1. 心得体会

本次实验中，我自己动手打了栈和队列的基本操作，对于栈和队列的概念有了更深的理解，对于栈和队列的基本操作如今也能熟练的掌握。而这些代码非常基础，也确实比较简单。

附录：源程序文件清单

各程序源代码文件随本实验报告电子版一起打包，存放在文件夹栈和队列操作的实现中。

文件清单如下：

栈操作的实现.c…………………………………………栈操作的实现

队列操作的实现.c………………………………………队列操作的实现

2.括号匹配的检验（设计性实验）

1. 需求分析

问题描述 假设表达式中允许有两种括号,即圆括号和方括号,其嵌套的顺序随意,即(()[ ])或[([ ] [ ])]等均 为正确格式,[(])或(((]均为错误格式.检验括号是否匹配的方法可用“期待的紧迫程度”这个概念来 描述.例如,考虑下列的括号序列：

[ ( [ ] [ ] ) ] 1 2 3 4 5 6 7 8

当计算机接受了第1个括号“[”以后,它期待着与其匹配的第8个括号“]”的出现,然而等 来的却是第2个括号,此时第1个括号“[”只能暂时靠边,而迫切等待与第2个括号相匹配的第7个 括号“]”的出现.类似地,因只等来了第3个括号“[”,此时,其期待的紧迫程度比第2个括号更 紧迫,则第2个括号只能靠边,让位于第3个括号,显然第3个括号的期待紧迫程度高于第2个括号, 而第2个括号的期待紧迫程度高于第1个括号.在接受了第4个括号“]”之后,第3个括号的期待得 到了满足,消解之后,第2个括号的期待匹配就成了最急迫的任务了,依次类推.可见这个处理过 程正好和栈的特点相吻合.

基本要求 读入圆括号和方括号的任意序列,输出“匹配”或“此串括号匹配不合法”.

测试数据 输入([ ]()),结果“匹配”. 输入[ [( )],结果“此串括号匹配不合法”.

实现提示 设置一个栈,每读入一个括号,若是左括号,则作为一个新的更急迫的期待压入栈中；若是右括 号,并且与当前栈顶的左括号相匹配,则将当前栈顶的左括号退出,继续读取下一个括号,如果读入 的右括号与当前栈顶的左括号不匹配,则属于不合法的情况.在初始和结束时,栈应该是空的.

概要设计

1. 为了实现程序功能，需要定义栈和队列的抽象数据类型。

ADT Stack {

数据对象：

D＝{ ai | ai ∈ElemSet, i=1,2,...,n, n≥0 }

数据关系：

R1＝{ <ai-1, ai >| ai-1, ai∈D, i=2,...,n }

约定an 端为栈顶，a1 端为栈底。

typedef struct{

selemtype \*base; //尾指针

selemtype \*top; //头指针

int stacksize; //栈长

}sqstack; //栈结构

基本操作：

InitStack(&S)

操作结果：构造一个空栈S

DestroyStack (&S)

初始条件：栈S已存在

操作结果：销毁栈S

ClearStack (&S)

初始条件：栈S已存在

操作结果：将栈置为空栈

StackEmpty(&S)  
 初始条件：栈S已存在

操作结果：若S为空栈则返回TRUE，否则返回FALSE

StackLength(&S)

初始条件：栈S已存在

操作结果：返回S中数据元素的个数，即栈的长度

GetTop(&S, &e)

初始条件：栈S已存在

操作结果：用e返回S的栈顶元素

Push(&S, e)

初始条件：栈S已存在  
 操作结果：插入元素e为新的栈顶元素

Pop(&S, &e)  
 初始条件：栈S已存在且非空

操作结果：删除S的栈顶元素，并用e返回其值

StackTraverse(&S, visit())

初始条件：栈S已存在且非空

操作结果：从栈底到栈顶依次对S的每个数据元素调用函数visit()，一旦visit()失败，则操作失败

}ADT Stack

本程序包含函数：

void InitStack(SqStack \*s)

bool StackEmpty(SqStack \*s)

bool Push(SqStack \*s, char e)

bool Pop(SqStack \*s, char \*e)

bool March(SqStack \*s, char \*str)

1. 详细设计
2. #include <stdio.h>
3. #include <stdlib.h>
4. #define MaxSize 100
5. typedef struct {
6. char data[MaxSize];
7. int top;
8. } SqStack;
9. void InitStack(SqStack \*s) {
10. s->top = -1;
11. }
12. bool StackEmpty(SqStack \*s) {
13. return (s->top == -1);
14. }
15. bool Push(SqStack \*s, char e) {
16. if (s->top == MaxSize - 1)
17. return false;
18. s->top++;
19. s->data[s->top] = e;
20. return true;
21. }
22. bool Pop(SqStack \*s, char \*e) {
23. if (s->top == -1)
24. return false;
25. \*e = s->data[s->top];
26. s->top--;
27. return true;
28. }
29. bool March(SqStack \*s, char \*str) {
30. InitStack(s);
31. char e;
32. int i = 0;
33. while (str[i] != '\0') {
34. switch (str[i]) {
35. case '(':
36. Push(s, '(');
37. break;
38. case '{':
39. Push(s, '{');
40. break;
41. case '[':
42. Push(s, '[');
43. break;
44. case ')':
45. Pop(s, &e);
46. if (e != '(')
47. return 0;
48. break;
49. case '}':
50. Pop(s, &e);
51. if (e != '{')
52. return 0;
53. break;
54. case ']':
55. Pop(s, &e);
56. if (e != '[')
57. return 0;
58. break;
59. default:
60. break;
61. }
62. i++;
63. }
64. int h = StackEmpty(s);
65. if (h == 0)
66. return 0;
67. else
68. return 1;
69. }
70. int main() {
71. SqStack s;
72. char str[MaxSize];
73. printf("请输入你要收入的字符串：");
74. scanf("%s", str);
75. bool h = March(&s, str);
76. if (h == 0)
77. printf("括号不匹配");
78. else
79. printf("括号匹配");
80. return 0;

}

4.调试分析

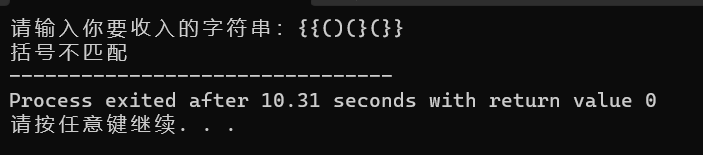
1. 采用IDE中自带的调试功能进行调试，手动添加断点和查看程序。
2. 对设计和编码的讨论和分析。该程序实现了城市链表的操作。分析程序代码的质量，主要从以下几个方面考虑。 

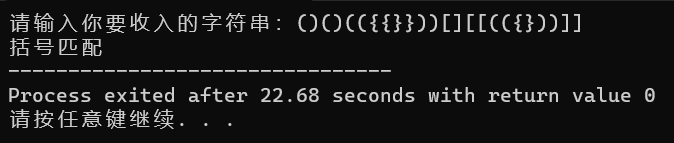
* 正确性。在一定的数据范围内，该程序能实现所需功能，所以正确性是没有问题的。
* 健壮性。在一定的数据输入范围内，该程序能较好的实现链表的操作。但是如果输入数 据非法，该程序还是可能会产生一些预想不到的输出结构，或是不做任何处理。所以， 该程序的健壮性有待进一步的提高。要综合考虑一些情况，当输入有误时，应返回一个 表示错误的值，并中止程序的执行，以便在更高的抽象层次上进行处理。

5.使用说明

按照屏幕提示，输入一个非负int型数字，输出对应八进制int型数字；输入0则退出程序。

6.测试程序的运行结果





1. 心得体会

本次实验中，我运用了验证性实验中栈的基本操作作为头文件，用栈为数据结构实现了数制转换这一功能，学会并理解了头文件的含义，也对栈的运用有了更好的理解。

附录：源程序文件清单

各程序源代码文件随本实验报告电子版一起打包，存放在文件夹数制转换问题中。

文件清单如下：括号匹配.c

3.停车场管理（综合性实验）

1. 需求分析

需求：

设停车场内是一个可停放n辆汽车的狭长区域，且只有一个大门可供汽车进出。在停车场内，汽车按到达时间的先后顺序，依次由北向南排列（假设大门在最南端，最先到达的那辆车停放在车场的 最北端）。若车场内已停满n辆汽车，则后来的汽车只能在门外的便道上等候。一旦有车开走，则排在 便道上的第一辆车即可开入。当停车场内某辆车要离开时，在它之后开入的车辆必须先退出车场为它 让路，待该车开出大门外，其他车再按原次序返回停车场。每辆停放在车场的车离开停车场时，必须 按其停留时间的长短交费。试为停车场编制按上述要求进行管理的模拟程序。

基本要求： 以顺序栈模拟停车场，以链队列模拟便道，按照从终端读入的输入数据序列进行模拟管理。每一组输入数据包括3项：是“到达”还是“离去”、汽车牌照号码及“到达”或“离去”的时刻。对每一 组输入数据进行操作后的输出数据为：如果是到达的车辆，则输出其在停车场内或便道上的停车位置； 如果是离去的车辆，则输出其在停车场内的停留时间和应交的费用（在便道上停留的时间不收费）。 栈以顺序结构实现，队列以链表实现。

分析：

测试数据 设n＝2,输入数据为(A, 1, 5),(A, 2, 10),(D, 1, 15),(A, 3, 20),(A, 4, 25),(A, 5, 30),(D, 2, 35), (D, 4, 40)和(E, 0, 0).每一组输入数据包括3个数据项：汽车“到达”或“离去”信息,汽车牌照号码 及到达或离去的时刻,其中,“A”表示到达,“D”表示离去,“E”表示输入结束.

基本要求 以顺序栈模拟停车场,以链队列模拟便道,按照从终端读入的输入数据序列进行模拟管理.每一 组输入数据包括3项：是“到达”还是“离去”,汽车牌照号码及“到达”或“离去”的时刻.对每一 组输入数据进行操作后的输出数据为：如果是到达的车辆,则输出其在停车场内或便道上的停车位置； 如果是离去的车辆,则输出其在停车场内的停留时间和应交的费用(在便道上停留的时间不收费). 栈以顺序结构实现,队列以链表实现.

实现提示 需另设一个栈,临时停放为给要离去的汽车让路而从停车场退出来的汽车,也用顺序存储结构实 现.输入数据按到达或离去的时刻有序.栈中每个元素表示一辆汽车,包含两个数据项：汽车的牌照 号码和进入停车场的时刻.

1. 概要设计

为了实现程序功能，需要定义栈和队列的抽象数据类型。

ADT Stack {

数据对象：

D＝{ ai | ai ∈ElemSet, i=1,2,...,n, n≥0 }

数据关系：

R1＝{ <ai-1, ai >| ai-1, ai∈D, i=2,...,n }

约定an 端为栈顶，a1 端为栈底。

typedef struct{

selemtype \*base; //尾指针

selemtype \*top; //头指针

int stacksize; //栈长

}sqstack; //栈结构

基本操作：

InitStack(&S)

操作结果：构造一个空栈S

DestroyStack (&S)

初始条件：栈S已存在

操作结果：销毁栈S

ClearStack (&S)

初始条件：栈S已存在

操作结果：将栈置为空栈

StackEmpty(&S)  
 初始条件：栈S已存在

操作结果：若S为空栈则返回TRUE，否则返回FALSE

StackLength(&S)

初始条件：栈S已存在

操作结果：返回S中数据元素的个数，即栈的长度

GetTop(&S, &e)

初始条件：栈S已存在

操作结果：用e返回S的栈顶元素

Push(&S, e)

初始条件：栈S已存在  
 操作结果：插入元素e为新的栈顶元素

Pop(&S, &e)  
 初始条件：栈S已存在且非空

操作结果：删除S的栈顶元素，并用e返回其值

StackTraverse(&S, visit())

初始条件：栈S已存在且非空

操作结果：从栈底到栈顶依次对S的每个数据元素调用函数visit()，一旦visit()失败，则操作失败

}ADT Stack

ADT Queue {

数据对象：

D＝{ai | ai∈ElemSet, i=1,2,...,n, n≥0}

数据关系：

R1＝{ <a i-1, ai > | ai-1, ai ∈D, i=2,...,n}

约定其中a1 端为队列头，an 端为队列尾

typedef struct{

qelemtype \*base;

int front;

int rear;

}sqqueue; //队列结构

基本操作：

InitQueue(&Q)  
 操作结果：构造一个空队列Q。  
 DestroyQueue(&Q)  
 初始条件：队列Q已存在。  
 操作结果：队列Q被销毁，不再存在。  
 QueueEmpty(Q)  
 初始条件：队列Q已存在。  
 操作结果：若Q为空队列，则返回TRUE，否则返回FALSE。

QueueLength(Q)  
 初始条件：队列Q已存在。  
 操作结果：返回Q的元素个数，即队列的长度。

GetHead(Q,&e)  
 初始条件：Q为非空队列。  
 操作结果：用e返回Q的队头元素。

ClearQueue(&Q)  
 初始条件：队列Q已存在。  
 操作结果：将Q清为空队列。   
 EnQueue(&Q, e)  
 初始条件：队列Q已存在。  
 操作结果：插入元素e为Q的新的队尾元素

DeQueue(&Q, &e)  
 初始条件：Q为非空队列。  
 操作结果：删除Q的队头元素，并用e返回其值。

QueueTraverse(Q, visit())

初始条件：队列Q存在且非空

操作结果：从栈底到栈顶依次对S的每个数据元素调用函数visit()，一旦visit()失败，则操作失败

} ADT Queue

本程序包含函数：

* //初始化栈
* void Stackinit(Stack\* s,int size);
* //入栈
* int Stackpush(Stack\* s, Carinfo c)
* //出栈
* int Stackpop(Stack\* s, Carinfo\* e)
* //判断是否为空
* int Stackempty(Stack s)
* //判断栈是否满
* int StackFull(Stack s)
* Queue Queueinit()
* //入队
* int Queuepush(Queue q, Carinfo c)
* //出队
* int Queuepop(Queue q, Carinfo\* e)
* //判断是否为空
* int Queueempty(Queue q)
* //计算停车费xqls
* double pay(int h1, int h2, double p)

1. 详细设计

停车场管理.c：

1. #define \_CRT\_SECURE\_NO\_WARNINGS 1
2. #define Maxpart 2
3. #include <stdio.h>
4. #include <stdlib.h>
5. #include <string.h>
6. //定义车辆信息
7. typedef struct \_carinfo
8. {
9. char id[10]; //车牌号
10. int hourCome; //到达时间，单位时，为了简便，这里只处理同一天内的时间
11. int hourLeave; //离开时间，单位时
12. }Carinfo;
13. /\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*定义顺序栈，栈，先进后出xqls\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/
14. typedef struct \_Stack
15. {
16. Carinfo\* pcar;
17. int maxlen;
18. int front;
19. }Stack;
20. //初始化栈
21. void Stackinit(Stack\* s,int size)
22. {
23. s->maxlen = size;
24. s->pcar = (Carinfo\*)malloc(size \* sizeof(Carinfo));
25. s->front = 0;
26. }
27. //入栈
28. int Stackpush(Stack\* s, Carinfo c)
29. {
30. if (s->front >= s->maxlen)
31. return 0;
32. else
33. {
34. s->pcar[s->front] = c;
35. s->front++;
36. return s->front;
37. }
38. }
39. //出栈
40. int Stackpop(Stack\* s, Carinfo\* e)
41. {
42. if (s->front <= 0)
43. return 0;
44. else
45. {
46. s->front--;
47. \*e = s->pcar[s->front];
48. return 1;
49. }
50. }
51. //判断是否为空
52. int Stackempty(Stack s)
53. {
54. if (s.front <= 0)
55. return 1;
56. else
57. return 0;
58. }
59. //判断栈是否满
60. int StackFull(Stack s)
61. {
62. if (s.front == s.maxlen)
63. return 1;
64. else
65. return 0;
66. }
67. /\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*定义链式队列，先进先出xqls\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/
68. typedef struct \_node
69. {
70. Carinfo car;
71. struct \_node\* next;
72. }QueueNode,\*Queue;
73. //初始化队列
74. Queue Queueinit()
75. {
76. Queue head = (Queue)malloc(sizeof(QueueNode));
77. head->next = 0;
78. return head;
79. }
80. //入队
81. int Queuepush(Queue q, Carinfo c)
82. {
83. Queue p = q;
84. QueueNode\* t = (QueueNode\*)malloc(sizeof(QueueNode));
85. int pos = 1;
86. t->car = c;
87. t->next = 0;
88. while (p->next)
89. {
90. p = p->next;
91. pos++;
92. }
93. p->next = t;
94. return pos;
95. }
96. //出队
97. int Queuepop(Queue q, Carinfo\* e)
98. {
99. Queue p = q->next;
100. if (q->next == 0)
101. return 0;
102. else
103. {
104. \*e = p->car;
105. q->next = p->next;
106. free(p);
107. p = 0;
108. return 1;
109. }
110. }
111. //判断是否为空
112. int Queueempty(Queue q)
113. {
114. if (q->next == 0)
115. return 1;
116. else
117. return 0;
118. }
119. //计算停车费xqls
120. double pay(int h1, int h2, double p)
121. {
122. double out = (h2 - h1) \* p;
123. return out;
124. }
126. int main()
127. {
128. Stack parking; //停车场栈 xqls
129. Stack tmp; //临时栈 xqls
130. Queue waiting; //等候队列 xqls
131. //int n; //停车场最多停放的车辆数目
132. char op; //操作
133. int h,h1;
134. int pos = 0;
135. char id[10] = { 0 };
136. double pay\_perhour = 2; //每小时的停车费
137. /\*printf("请输入停车场最大的停车数量n：");
138. scanf("%d", &n);\*/
139. //初始化
140. Stackinit(&parking,Maxpart);
141. Stackinit(&tmp,Maxpart);
142. waiting = Queueinit();
144. while (1)
145. {
146. printf("请输入操作：A.到达；D.离开；E.退出程序 : ");
147. scanf(" %c", &op);
148. if (op == 'D'&& Stackempty(parking))
149. {
150. printf("停车场目前暂无车辆\n");
151. continue;
152. }
153. printf("请输入车牌号：");
154. scanf("%s", id);
155. //到达
156. if (op == 'A')
157. {
158. printf("请输入到达时间：");
159. scanf("%d", &h);
160. Carinfo cc;
161. strcpy(cc.id, id);
162. cc.hourCome = h;
163. if (StackFull(parking))
164. {
165. pos = Queuepush(waiting, cc);
166. printf("便道位置：%d\n", pos);
167. }
168. else
169. {
170. pos = Stackpush(&parking, cc);
171. printf("停车场位置：%d\n", pos);
172. }
173. }
174. else if (op == 'D')
175. {
176. while (1)
177. {
178. //该车之后的先出栈，入临时栈
179. Carinfo cc;
180. Stackpop(&parking, &cc);
181. if (strcmp(cc.id, id) == 0)
182. {
183. printf("请输入离开时间：");
184. scanf("%d", &cc.hourLeave);
185. h1=cc.hourLeave;
186. printf("缴费：%.1f\n 停留时间：%d\n",pay(cc.hourCome, cc.hourLeave, pay\_perhour),cc.hourLeave-cc.hourCome);
187. break;
188. }
189. else
190. {
191. Stackpush(&tmp, cc);
192. printf(" %s入临时栈\n", cc.id);
193. }
194. }
195. //临时栈车辆返回
196. while (!Stackempty(tmp))
197. {
198. Carinfo cc;
199. Stackpop(&tmp, &cc);
200. Stackpush(&parking, cc);
201. printf(" %s返回停车场\n", cc.id);
202. }
203. //等待队列入栈
204. if (!Queueempty(waiting))
205. {
206. Carinfo cc;
207. Queuepop(waiting, &cc);
208. cc.hourCome=h1;
209. pos = Stackpush(&parking, cc);
210. printf("%s驶入停车场，位置：%d\n", cc.id,pos);
211. }
213. }
214. else if (op == 'E'){
215. scanf("%d");
216. break;
217. }
218. }
219. return 0;
220. }

4.调试分析

1. 采用IDE中自带的调试功能进行调试，手动添加断点和查看程序。
2. 对设计和编码的讨论和分析。该程序实现了城市链表的操作。分析程序代码的质量，主要从以下几个方面考虑。 

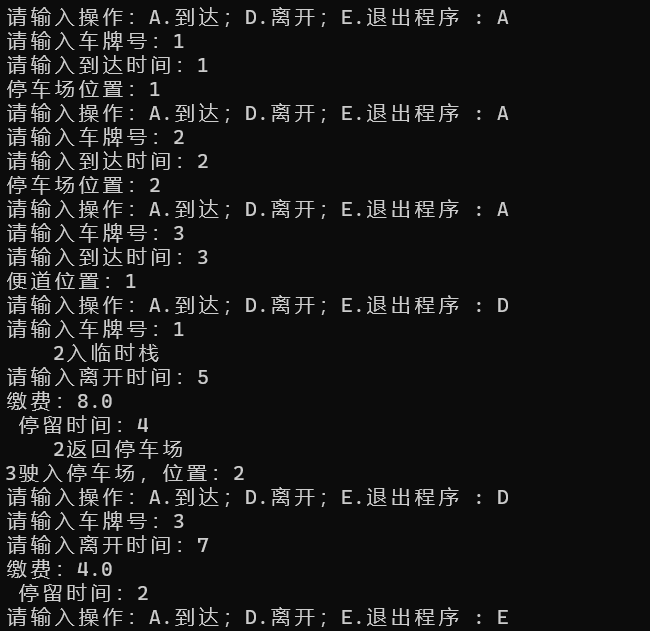
* 正确性。在一定的数据范围内，该程序能实现所需功能，所以正确性是没有问题的。
* 健壮性。在一定的数据输入范围内，该程序能较好的实现链表的操作。但是如果输入数 据非法，该程序还是可能会产生一些预想不到的输出结构，或是不做任何处理。所以， 该程序的健壮性有待进一步的提高。要综合考虑一些情况，当输入有误时，应返回一个 表示错误的值，并中止程序的执行，以便在更高的抽象层次上进行处理。

1. 使用说明

按照屏幕提示，将数据放入对应文件中，按下ENTER键，结果放在文件“停车场输出结果.txt”中。

数据格式：(A, 1, 5)，第一个代表性质，第二个代表车牌号，第三个代表时间。

1. 测试程序的运行结果



1. 心得体会

本次实验中，我利用栈的基本操作和链队列的基本操作模拟了停车场，对于栈和队列这两种数据结构有了更深的理解和运用。

附录：源程序文件清单

各程序源代码文件随本实验报告电子版一起打包，存放在文件夹停车场管理中。

文件清单如下：

停车场管理.c…………………………………………停车场的模拟