**数据结构第一次上机实验报告**

学号：161730126 姓名：李双玖

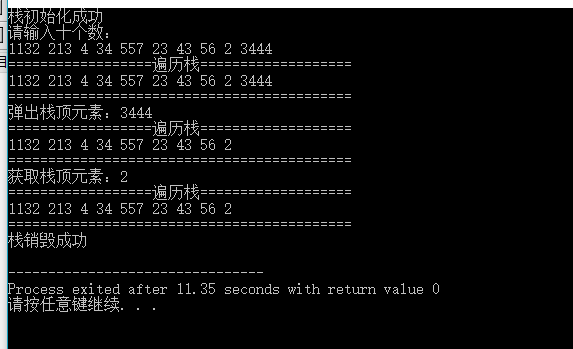
1. 调试成功程序及说明

**1、**

**题目：编程实现书P45 ADT Stack 基本操作9个，用顺序存储结构实现；**

算法思想：动态顺序数组形式进行储存。

运行结果：



结果分析：成功。

源程序：

1. #include<iostream>
2. #include<stdlib.h>
4. #define STACK\_INIT\_SIZE 100
5. #define STACKINCREMENT 10
7. **using** **namespace** std;
9. **typedef** **struct** sqStack {
10. **int** stackSize;
11. **int** \*base;
12. **int** \*top;
13. }sqStack;
15. **class** SqStack {
17. **public**:
18. **bool** InitStack(sqStack &sqStack) {
19. sqStack.base = (**int**\*)malloc(**sizeof**(**int**)\*STACK\_INIT\_SIZE);
20. **if**(!sqStack.base) **return** **false**;
21. sqStack.top = sqStack.base;
22. sqStack.stackSize = STACK\_INIT\_SIZE;
23. **return** **true**;
24. }
26. **void** DestroyStack(sqStack &sqStack) {
27. **if**(sqStack.base) free(sqStack.base);
28. }
30. **void** ClearStack(sqStack &sqStack) {
31. **if**(sqStack.base) sqStack.top = sqStack.base;
32. }
34. **bool** StackEmpty(sqStack sqStack) {
35. **if**(!sqStack.base) **return** **true**;
36. **if**(sqStack.base == sqStack.top-1) **return** **true**;
37. **return** **false**;
38. }
40. **int** StackLength(sqStack sqStack) {
41. **if**(!sqStack.base) **return** 0;
42. **return** sqStack.top - sqStack.base;
43. }
45. **bool** GetTop(sqStack sqStack, **int** &e) {
46. **if**(!sqStack.base) **return** **false**;
47. e = \*(sqStack.top-1);
48. **return** **true**;
49. }
51. **bool** Push(sqStack &sqStack, **int** e) {
52. **if**(!sqStack.base) **return** **false**;
53. **if**(StackLength(sqStack) == sqStack.stackSize-1) {
54. sqStack.base = (**int**\*)realloc(sqStack.base, sqStack.stackSize+STACKINCREMENT);
55. sqStack.top = sqStack.base + sqStack.stackSize - 1;
56. sqStack.stackSize += STACKINCREMENT;
57. }
58. \*sqStack.top = e;
59. sqStack.top++;
60. **return** **true**;
61. }
63. **bool** Pop(sqStack &sqStack, **int** &e) {
64. **if**(!sqStack.base || !StackLength(sqStack)) **return** **false**;
65. sqStack.top--;
66. e = \*sqStack.top;
67. **return** **true**;
69. }
71. **void** StackTraverse(sqStack sqStack) {
72. printf("==================遍历栈===================\n");
73. **int** \*p = sqStack.base;
74. **while**(p!=sqStack.top) {
75. printf("%d ", \*p);
76. p++;
77. }
78. printf("\n===========================================\n");
79. }
80. };

83. **int** main() {
84. SqStack S;
85. sqStack s;
86. **if**(S.InitStack(s))
87. printf("栈初始化成功\n");
88. printf("请输入十个数：\n");
89. **int** num = 0;
90. **for**(**int** i=0; i<10; i++) {
91. scanf("%d", &num);
92. S.Push(s, num);
93. }
94. S.StackTraverse(s);
96. **int** temp = 0;
97. S.Pop(s, temp);
98. printf("弹出栈顶元素：%d\n", temp);
99. S.StackTraverse(s);
101. S.GetTop(s, temp);
102. printf("获取栈顶元素：%d\n", temp);
103. S.StackTraverse(s);
105. S.DestroyStack(s);
106. printf("栈销毁成功\n");
107. }

**2、**

**题目：编程实现书P59 ADT Queue 基本操作9个，用链式存储结构实现；**

算法思想：用带头结点的链表形式进行储存。队头指针指向头结点，队尾指针指向最后一个结点。

运行结果：



结果分析：成功。

源程序：

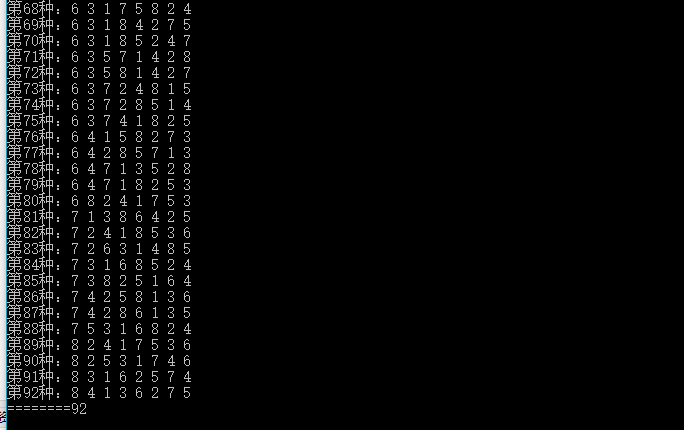
1. #include <stdio.h>
2. #include <stdlib.h>
4. **typedef** **struct** QNode{
5. **int** data;
6. **struct** QNode \*next;
7. }\*QueuePtr;
9. **typedef** **struct**{
10. QueuePtr front;
11. QueuePtr rear;
12. }LinkQueue;
14. **void** InitQueue(LinkQueue &Q) {
15. Q.front = (QNode\*)malloc(**sizeof**(QNode));
16. Q.rear = Q.front;
17. }
19. **void** ClearQueue(LinkQueue &Q) {
20. QNode \*p = Q.front->next, \*temp;
21. **while**(p) {
22. temp = p;
23. p = p->next;
24. free(temp);
25. }
26. Q.rear = Q.front;
27. }
29. **void** DestoryQueue(LinkQueue &Q) {
30. ClearQueue(Q);
31. free(Q.front);
32. }
34. **bool** QueueEmpty(LinkQueue Q) {
35. **if**(Q.front == Q.rear) **return** **true**;
36. **else** **return** **false**;
37. }
39. **int** QueueLength(LinkQueue Q) {
40. QNode \*p = Q.front->next;
41. **int** account = 0;
42. **while**(p) {
43. account++;
44. p = p->next;
45. }
46. **return** account;
47. }
49. **bool** GetHead(LinkQueue Q,**int** &e) {
50. QNode \*p = Q.front->next;
51. **if**(p) {
52. e = p->data;
53. **return** **true**;
54. }
55. **else** **return** **false**;
56. }
58. **void** EnQueue(LinkQueue &Q,**int** e) {
59. QNode \*temp = (QNode\*)malloc(**sizeof**(QNode));
60. temp->data = e;
61. temp->next = NULL;
62. Q.rear->next = temp;
63. Q.rear = temp;
64. }
66. **bool** DeQueue(LinkQueue &Q,**int** &e) {
67. QNode \*p = Q.front->next;
68. Q.front->next = p->next;
69. e = p->data;
70. free(p);
71. }
73. **void** QueueTraverse(LinkQueue Q) {
74. printf("=================遍历队列==================\n");
75. QNode \*p = Q.front->next;
76. **while**(p) {
77. printf("%d ", p->data);
78. p = p->next;
79. }
80. printf("\n===========================================\n");
81. }
83. **int** main() {
84. LinkQueue Q;
85. InitQueue(Q);
86. printf("队列初始化成功\n");
88. **for**(**int** i=0; i<10; i++) {
89. EnQueue(Q, i);
90. }
91. QueueTraverse(Q);
93. **int** temp = 0;
94. DeQueue(Q, temp);
95. printf("删去队头元素\n");
96. QueueTraverse(Q);
98. GetHead(Q, temp);
99. printf("队列头元素为：%d\n", temp);
101. EnQueue(Q, temp);
102. printf("在队尾加入%d\n", temp);
103. QueueTraverse(Q);
105. DestoryQueue(Q);
106. printf("队列销毁成功\n");
107. }

**3、**

**题目：利用栈操作实现八皇后问题求解。**

算法思想：用函数实现栈操作。因为函数本身是一个栈帧，函数的调用与递归就是栈的操作。定义一个一维数组num用来存放八皇后的位置，num[i]=j表示位置在第i行第j列。定义check(int)来判断当前摆放的八皇后是否满足条件。定义函数fun(int)来实现函数栈操作，当i==8即达到第九行时函数返回，在函数利用for循环在1<=num[i]<=8的范围中使num[i]++，若满足check条件则fun(i+1)，for循环结束后函数return

运行结果：



结果分析：成功。

源程序：

1. #include<stdio.h>
3. **int** num[8];
4. **int** flag = 0;
6. **void** show() {
7. flag++;
8. printf("第%d种：", flag);
9. **for**(**int** i=0; i<8; i++) printf("%d ", num[i]);
10. printf("\n");
11. }
13. **bool** check(**int** length) {
14. **for** (**int** i = 0; i < length-1; i++) {
15. **for** (**int** j = i+1; j < length; j++) {
16. **if** (num[i] == num[j] || num[j]-num[i] == j-i || num[i]-num[j] == j-i) {
17. **return** **false**;
18. }
19. }
20. }
21. **return** **true**;
22. }
24. **void** fun(**int** i) {
25. **if**(i == 8) {
26. show();
27. **return**;
28. }
29. **for**(num[i]=1; num[i]<=8; num[i]++) {
30. **if**(check(i+1)) fun(i+1);
31. }
32. **return**;
33. }
35. **int** main() {
36. fun(0);
37. printf("========%d\n", flag);
38. }

**4、**

**题目：在某图形操作系统中,有 N 个窗口,每个窗口都是一个两边与坐标轴分别平行的矩形区域。窗口的边界上的点也属于该窗口。窗口之间有层次的区别,在多于一个窗口重叠的区域里,只会显示位于顶层的窗口里的内容。  
　　当你点击屏幕上一个点的时候,你就选择了处于被点击位置的最顶层窗口,并且这个窗口就会被移到所有窗口的最顶层,而剩余的窗口的层次顺序不变。如果你点击的位置不属于任何窗口,则系统会忽略你这次点击。  
　　现在我们希望你写一个程序模拟点击窗口的过程。**

**输入格式**

**输入的第一行有两个正整数,即 N 和 M。(1 ≤ N ≤ 10,1 ≤ M ≤ 10)  
　　接下来 N 行按照从最下层到最顶层的顺序给出 N 个窗口的位置。 每行包含四个非负整数 x1, y1, x2, y2,表示该窗口的一对顶点坐标分别为 (x1, y1) 和 (x2, y2)。保证 x1 < x2, y1 < y2。  
　　接下来 M 行每行包含两个非负整数 x, y,表示一次鼠标点击的坐标。  
　　题目中涉及到的所有点和矩形的顶点的 x, y 坐标分别不超过 2559 和　　1439。**

**问题分析：这个问题可以用链式线性表来实现。**

**输出格式：输出包括 M 行,每一行表示一次鼠标点击的结果。如果该次鼠标点击选择了一个窗口,则输出这个窗口的编号(窗口按照输入中的顺序从 1 编号到 N);如果没有,则输出"IGNORED"(不含双引号)。**

**样例输入**

**3 4  
0 0 4 4  
1 1 5 5  
2 2 6 6  
1 1  
0 0  
4 4  
0 5**

**样例输出**

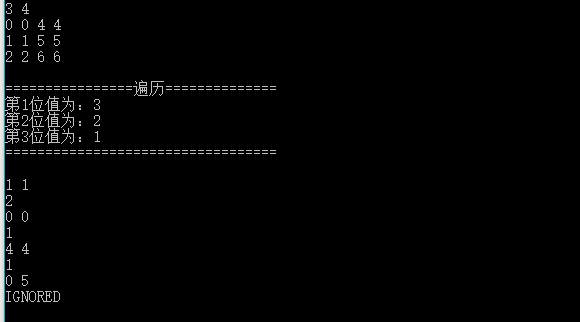
**2  
1  
1  
IGNORED**

**样例说明**

**第一次点击的位置同时属于第 1 和第 2 个窗口,但是由于第 2 个窗口在上面,它被选择并且被置于顶层。  
　　第二次点击的位置只属于第 1 个窗口,因此该次点击选择了此窗口并将其置于顶层。现在的三个窗口的层次关系与初始状态恰好相反了。  
　　第三次点击的位置同时属于三个窗口的范围,但是由于现在第 1 个窗口处于顶层,它被选择。  
　　最后点击的 (0, 5) 不属于任何窗口。**

算法思想：用带头结点的链表结构进行储存。每次将点击窗口插到第一个位置。

运行结果：



结果分析：成功。

源程序：

1. #include<stdio.h>
2. #include<stdlib.h>
4. **typedef** **struct** LNode {
5. **int** data;
6. **int** x1, y1, x2, y2;
7. LNode \*next;
8. }\*LinkList;
10. //遍历函数
11. **void** ListTraverse(LinkList list) {
12. printf("\n================遍历==============\n");
13. LNode \*p = list->next;
14. **int** i = 1;
15. **while**(p) {
16. printf("第%d位值为：%d\n", i, p->data);
17. p = p->next;
18. i++;
19. }
20. printf("==================================\n\n");
21. };
23. **int** main() {
24. **int** n, m;
25. scanf("%d%d", &n, &m);
26. LinkList list = (LNode\*)malloc(**sizeof**(LNode));
27. list->next = NULL;
29. LNode \*temp;
30. **for**(**int** i=0; i<n; i++) {
31. temp = (LNode\*)malloc(**sizeof**(LNode));
32. scanf("%d%d%d%d", &temp->x1, &temp->y1, &temp->x2, &temp->y2);
33. temp->data = i+1;
34. temp->next = list->next;
35. list->next = temp;
36. }
37. ListTraverse(list);
39. **int** x, y;
40. LNode \*prior, \*p;
41. **int** flag;
42. **for**(**int** i=0; i<m; i++) {
43. scanf("%d%d", &x, &y);
44. p = list->next;
45. prior = list;
46. flag = 0;
47. **while**(p) {
48. **if**(p->x1<=x && p->x2>=x && p->y1<=y && p->y2>=y) {
49. // printf("%d %d %d %d\n", p->x1, p->y1, p->x2, p->y2);
50. flag = p->data;
51. prior->next = p->next;
52. p->next = list->next;
53. list->next = p;
54. **break**;
55. }
56. **else** {
57. prior = p;
58. p = prior->next;
59. }
60. }
61. **if**(flag) printf("%d\n", flag);
62. **else** printf("IGNORED\n");
63. }
65. p = list;
66. **while**(p) {
67. temp = p;
68. p = p->next;
69. free(temp);
70. }
71. }

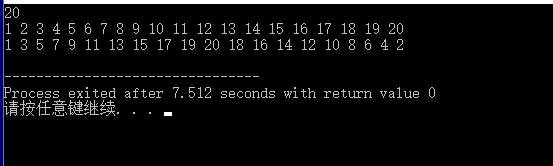
**5、**

**题目：**



算法思想：先用一个指针last指向链表尾结点，一个n记录链表长度。然后遍历1到n-1位置的结点，如果i%2==0，则将该结点插到last指针的下一个。

运行结果：



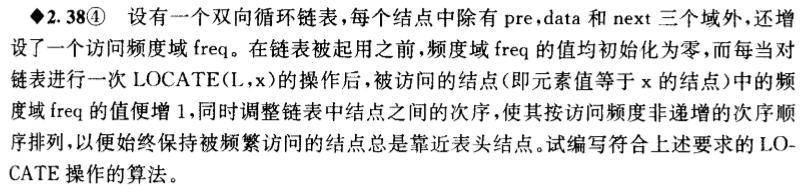
结果分析：成功。

源程序：

1. #include<stdio.h>
2. #include<stdlib.h>
4. **typedef** **struct** DLNode {
5. **int** data;
6. DLNode \*pre, \*next;
7. }\*DLinkList;
9. **int** main() {
10. DLinkList dList = (DLNode\*)malloc(**sizeof**(DLNode));
11. dList->pre = dList->next = NULL;
12. dList->data = 0;
14. **int** n = 0;
15. scanf("%d", &n);
17. DLNode \*temp, \*p = dList;
18. **for**(**int** i=0; i<n; i++) {
19. temp = (DLNode\*)malloc(**sizeof**(DLNode));
20. temp->data = i+1;
21. temp->pre = p;
22. p->next = temp;
23. p = temp;
24. }
25. p->next = NULL;
26. DLNode \*last = p;
28. p = dList->next;
29. **while**(p) {
30. printf("%d ", p->data);
31. p = p->next;
32. }
33. printf("\n");
35. DLNode \*pre = dList;
36. p = dList->next;
37. **int** i = 1;
38. **while**(i<n) {
39. **if**(i%2 == 0) {
40. pre->next = p->next;
41. **if**(p->next) p->next->pre = pre;
43. p->next = last->next;
44. last->next = p;
45. p->pre = last;
46. **if**(last->next) last->next->pre = p;
47. }
48. **else** pre = p;
49. p = pre->next;
50. i++;
51. }
53. p = dList->next;
54. **while**(p) {
55. printf("%d ", p->data);
56. p = p->next;
57. }
58. printf("\n");
60. p = dList;
61. **while**(p) {
62. temp = p;
63. p = p->next;
64. free(p);
65. }
66. }

**6、**

**题目：**



算法思想：在locate函数中，遍历链表直到p->data==x，然后将p结点依照顺序插到链表中的相应位置。

运行结果：



结果分析：成功。

源程序：

1. #include<stdio.h>
2. #include<stdlib.h>
4. **typedef** **struct** DLNode {
5. **int** data, freq;
6. DLNode \*pre, \*next;
7. }\*DLinkList;
9. **bool** locate(DLinkList dList, **int** num) {
10. DLNode \*pre = dList, \*p = dList->next;
11. **while**(p&&p->data!=num) {
12. pre = p;
13. p = pre->next;
14. }
15. **if**(p) {
16. p->freq++;
17. pre->next = p->next;
18. **if**(p->next) p->next->pre = pre;
19. DLNode \*temp = p;
21. pre = dList, p = dList->next;
22. **while**(p&&p->freq>temp->freq) {
23. pre = p;
24. p = pre->next;
25. }
26. pre->next = temp;
27. temp->next = p;
29. p->pre = temp;
30. temp->pre = pre;
32. **return** **true**;
33. }
34. **else** **return** **false**;
35. }
37. **int** main() {
38. DLinkList dList = (DLNode\*)malloc(**sizeof**(DLNode));
39. dList->pre = dList->next = NULL;
40. dList->data = 0;
42. **int** n = 0;
43. printf("请输入链表长度：");
44. scanf("%d", &n);
46. DLNode \*temp, \*p = dList;
47. **for**(**int** i=0; i<n; i++) {
48. temp = (DLNode\*)malloc(**sizeof**(DLNode));
49. temp->data = i+1;
50. temp->freq = 0;
51. temp->pre = p;
52. p->next = temp;
53. p = temp;
54. }
55. p->next = NULL;
56. DLNode \*last = p;
58. p = dList->next;
59. **while**(p) {
60. printf("data=%d, freq=%d  ", p->data, p->freq);
61. p = p->next;
62. }
63. printf("\n");

66. **int** i = 0, num = 0;
67. printf("请输入依次十个点击的序号：\n");
68. **while**(i<10) {
69. scanf("%d", &num);
70. **if**(!locate(dList, num)) printf("不存在这个数：%d\n", num);
71. i++;
73. p = dList->next;
74. **while**(p) {
75. printf("data=%d, freq=%d  ", p->data, p->freq);
76. p = p->next;
77. }
78. printf("\n");
79. }

82. p = dList;
83. **while**(p) {
84. temp = p;
85. p = p->next;
86. free(p);
87. }
88. }