**要求代码和实验报告规范，在算法思想中：对实验涉及的数据结构进行有效设计和分析；对算法进行分析并给出时间、空间复杂度的结论；清晰表达实验思路、出现的问题及解决方法。**

一、调试成功程序及说明

题目：

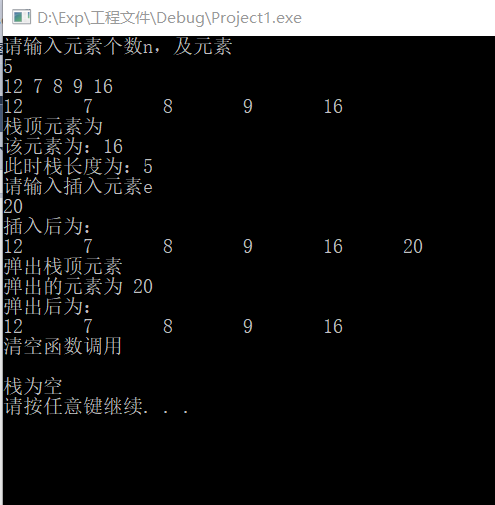
编程实现书P45 ADT Stack 基本操作9个，用顺序存储结构实现；

算法思想：

建立栈，顺序存储结构，动态分配空间。

在实现上基本与顺序表类似。

运行结果：



结果分析：

完成测试功能。

附源程序:

// ADT Stack 基本操作

// 顺序实现 (C++结构体)

#include <iostream>

using namespace std;

//函数结果状态代码

#define TRUE 1

#define FALSE 0

#define OK 1

#define ERROR 0

#define INFEASIBLE -1

//Status 为函数类型，值为函数结果状态代码

typedef int Status;

//栈定义

#define STACK\_INIT\_SIZE 100 //栈存储空间 初始分配量

#define STACKINCREMENT 10 //栈存储空间 分配增量

struct SqStack

{

int \*top;

int \*base;

int stacksize;

};

//十个 基本函数实现：

//构造(初始化)空栈

Status InitStack(SqStack &S)

{

S.base = new int[STACK\_INIT\_SIZE];

if (!S.base)

exit(OVERFLOW);

S.top = S.base;

S.stacksize = STACK\_INIT\_SIZE;

return OK;

}

//销毁栈S

Status DestroyStack(SqStack &S)

{

if (!S.base)

return ERROR;

delete []S.base;

S.top = NULL;

S.stacksize = -1;

return OK;

}

//将栈置为空栈

Status ClearStack(SqStack &S)

{

if (!S.base)

return ERROR;

S.top = S.base;

return OK;

}

//栈为空返回TRUE，否则返回FALSE

Status StackEmpty(SqStack S)

{

if (!S.base)

return ERROR;

if (S.top - S.base == 0)

return TRUE;

else

return FALSE;

}

//返回栈的长度

int StackLength(SqStack S)

{

if (!S.base)

return ERROR;

return S.top - S.base;

}

//若栈不空，用e返回S的栈顶元素，并返回OK；否则返回FALSE

Status GetTop(SqStack S, int &e)

{

if (S.top - S.base == 0)

return FALSE;

e = \*(S.top - 1);

return OK;

}

//插入e为新的栈顶元素

Status Push(SqStack &S, int e)

{

if (S.top - S.base == S.stacksize)

{

int \*newbase;

newbase = new int[S.stacksize + STACKINCREMENT];

if (!newbase)

exit(OVERFLOW);

for (int i = 0;i < S.stacksize;i++)

newbase[i] = S.base[i];

S.base = newbase;

S.top = S.base + S.stacksize;

S.stacksize += STACKINCREMENT;

}

\*S.top = e;

S.top++;

return OK;

}

//若栈不空，则删除S的栈顶元素，用e返回其值，并返回OK；否则返回FALSE

Status Pop(SqStack &S, int &e)

{

if (S.top - S.base == 0)

return ERROR;

e = \*(S.top - 1);

S.top--;

return OK;

}

//遍历栈,从栈底到栈顶

Status StackTraverse(SqStack S)

{

for (int i = 0; i < S.top - S.base; i++)

cout << S.base[i] << '\t';

cout << endl;

return OK;

}

//创建栈

Status CreatStack(SqStack &S, int n)

{

int e;

InitStack(S);

for (int i = 0; i < n; i++)

{

cin >> e;

Push(S, e);

}

return OK;

}

int main()

{

SqStack S;

int n;

//1.创建 2.遍历 3.初始化函数 检验

cout << "请输入元素个数n，及元素\n";

cin >> n;

CreatStack(S, n);

StackTraverse(S);

int e;

//4.获取元素函数 检验

cout << "栈顶元素为\n";

GetTop(S, e);

cout << "该元素为：" << e << endl;

//5.长度

cout << "此时栈长度为：" << StackLength(S) << endl;

//6.插入 7.弹出

cout << "请输入插入元素e\n";

cin >> e;

Push(S, e);

cout << "插入后为：" << endl;

StackTraverse(S);

cout << "弹出栈顶元素\n";

Pop(S, e);

cout << "弹出的元素为 " << e << endl;

cout << "弹出后为：" << endl;

StackTraverse(S);

//8.清空 9.是否为空 10.销毁

//清空栈

cout << "清空函数调用\n";

ClearStack(S);

StackTraverse(S);

if (StackEmpty(S))

cout << "栈为空" << endl;

DestroyStack(S);

system("pause");

return 0;

}

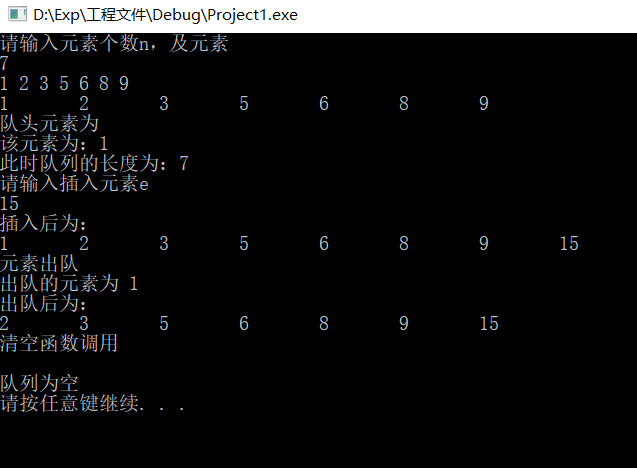
题目：

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：

建立链式队列，以线性链表为基础，实现先进先出的功能。

运行结果：



结果分析：

完成测试功能。

附源程序：

// ADT Queue 基本操作

// 链式实现 (C++结构体)

#include <iostream>

using namespace std;

//函数结果状态代码

#define TRUE 1

#define FALSE 0

#define OK 1

#define ERROR 0

#define INFEASIBLE -1

//Status 为函数类型，值为函数结果状态代码

typedef int Status;

//链式队列定义，单链队列

struct QNode

{

int data;

QNode \*next;

};

struct LinkQueue

{

QNode \*front;

QNode \*rear;

};

typedef QNode \*QueuePtr;

//创建值为e的队列结点

Status MakeQNode(QueuePtr &s, int e)

{

s = new QNode;

if (!s)

exit(OVERFLOW);

s->data = e;

s->next = NULL;

return OK;

}

//基本函数实现 十个

//构造一个空队列Q

Status InitQueue(LinkQueue &Q)

{

Q.front = new QNode;

if (!Q.front)

exit(OVERFLOW);

Q.rear = Q.front;

return OK;

}

//销毁队列Q，Q不存在

Status DestroyQueue(LinkQueue &Q)

{

QueuePtr p = Q.front;

QueuePtr q = p->next;

while (q)

{

delete p;

p = q;

q = q->next;

}

delete p;

Q.front = NULL;

Q.rear = NULL;

return OK;

}

//将Q清为空队列

Status ClearQueue(LinkQueue &Q)

{

QueuePtr p, q;

p = Q.front->next;

while (p)

{

q = p->next;

delete p;

p = q;

}

Q.front->next = NULL;

Q.rear = Q.front;

return OK;

}

//若队列为空，则返回TRUE，否则返回FALSE

Status QueueEmpty(LinkQueue Q)

{

if (Q.front == Q.rear)

return TRUE;

else

return FALSE;

}

//返回队列长度

int QueueLength(LinkQueue Q)

{

QueuePtr p = Q.front;

int i = 0;

while (p != Q.rear)

{

p = p->next;

i++;

}

return i;

}

//若队列不为空，则用e返回Q的队头元素，并返回OK；否则返回FALSE

Status GetHead(LinkQueue Q, int &e)

{

if (Q.front == Q.rear)

return FALSE;

QueuePtr head;

head = Q.front->next;

e = head->data;

return OK;

}

//插入元素e为Q的新的队尾元素

Status EnQueue(LinkQueue &Q, int e)

{

QueuePtr s;

MakeQNode(s, e);

Q.rear->next = s;

Q.rear = s;

return OK;

}

//若队列不空，则删除Q的队头元素，并用e返回其值，并返回OK，否则返回FALSE

Status DeQueue(LinkQueue &Q, int &e)

{

if (Q.front == Q.rear)

return FALSE;

QueuePtr p;

p = Q.front->next;

Q.front->next = p->next;

e = p->data;

delete p;

if (!Q.rear)

Q.rear = Q.front;

return OK;

}

//遍历输出队列

Status QueueTraverse(LinkQueue Q)

{

QueuePtr p;

p = Q.front->next;

while (p)

{

cout << p->data << '\t';

p = p->next;

}

cout << endl;

return OK;

}

//初始化并创建队列

Status CreatQueue(LinkQueue &Q, int n)

{

InitQueue(Q);

int e;

for (int i = 0;i < n;i++)

{

cin >> e;

EnQueue(Q, e);

}

return OK;

}

int main()

{

LinkQueue Q;

int n;

//1.创建 2.遍历 3.初始化函数 检验

cout << "请输入元素个数n，及元素\n";

cin >> n;

CreatQueue(Q, n);

QueueTraverse(Q);

int e;

//4.获取元素函数 检验

cout << "队头元素为\n";

GetHead(Q, e);

cout << "该元素为：" << e << endl;

//5.长度

cout << "此时队列的长度为：" << QueueLength(Q) << endl;

//6.入队 7.出队

cout << "请输入插入元素e\n";

cin >> e;

EnQueue(Q, e);

cout << "插入后为：" << endl;

QueueTraverse(Q);

cout << "元素出队\n";

DeQueue(Q, e);

cout << "出队的元素为 " << e << endl;

cout << "出队后为：" << endl;

QueueTraverse(Q);

//8.清空 9.是否为空 10.销毁

//清空队列

cout << "清空函数调用\n";

ClearQueue(Q);

QueueTraverse(Q);

if (QueueEmpty(Q))

cout << "队列为空" << endl;

DestroyQueue(Q);

system("pause");

return 0;

}

题目：

设某传感器，每秒不间断采集数据（整数0-1000），要求任何时刻只保留最后N个数据（初始值为0），并输出当前最大值，设N=10，试利用循环队列编写算法。

算法思想：

建立循环队列，设置出列条件。每次入列前加入判断条件——当队列长度大于等于N时先出列再入列，不足N时直接入列。

for (int i = 0; i < n; i++)

{

//只保留最后 N 个数据

if (Q.length==N)

{

DeQueue\_Sq(Q, e);

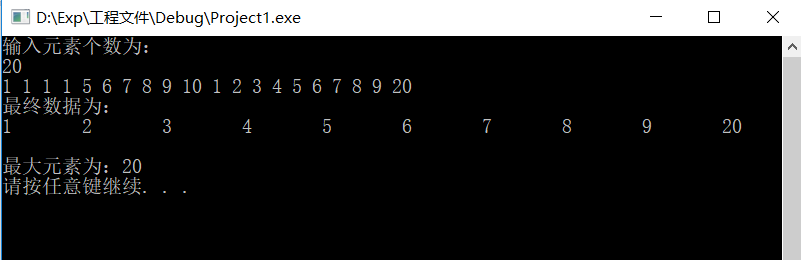
}

cin >> e;

EnQueue\_Sq(Q, e);

}

运行结果：



结果分析：

手动输入20个数，保留了最后10个。

附源程序:

Queue\_Sq.h

//SqQueue 循环队列实现

//(C++结构体)

#include <iostream>

using namespace std;

//函数结果状态代码

#define TRUE 1

#define FALSE 0

#define OK 1

#define ERROR 0

#define INFEASIBLE -1

//Status 为函数类型，值为函数结果状态代码

typedef int Status;

//队列定义

#define QUEUE\_INIT\_SIZE 100

#define QUEUEINCREMENT 10

struct SqQueue

{

int \*base;

int \*front;

int \*rear;

int length;

int queuesize;

};

//十个基本函数实现

//构造一个空队列Q

Status InitQueue\_Sq(SqQueue &Q)

{

Q.base = new int[QUEUE\_INIT\_SIZE];

if (!Q.base)

return OVERFLOW;

Q.front = Q.base;

Q.rear = Q.front;

Q.length = 0;

Q.queuesize = QUEUE\_INIT\_SIZE;

return OK;

}

//销毁队列Q

Status DestroyQueue\_Sq(SqQueue &Q)

{

//如果队列基地址不存在，可认为队列 Q 已经销毁

if (!Q.base)

return OK;

delete Q.base;

Q.front = NULL;

Q.rear = NULL;

Q.length = -1;

Q.queuesize = -1;

Q.base = NULL;

return OK;

}

//将Q清为空队列

Status ClearQueue\_Sq(SqQueue &Q)

{

Q.front = Q.rear;

Q.length = 0;

return OK;

}

//若队列为空，则返回TRUE，否则返回FALSE

Status QueueEmpty\_Sq(SqQueue Q)

{

if (Q.length == 0)

return TRUE;

else

return FALSE;

}

//返回队列长度

int QueueLength\_Sq(SqQueue Q)

{

return Q.length;

}

//若队列不为空，则用e返回Q的队头元素，并返回OK；否则返回FALSE

Status GetHead\_Sq(SqQueue Q, int &e)

{

if (Q.length == 0)

{

return ERROR;

}

else

{

e = \*Q.front;

return OK;

}

}

//插入元素e为Q的新的队尾元素

Status EnQueue\_Sq(SqQueue &Q, int e)

{

//判断队列是否已满，若满则申请新的空间

if (Q.length == Q.queuesize)

{

//申请空间

int \*newbase;

newbase = new int[Q.length + QUEUEINCREMENT];

if (!newbase)

return OVERFLOW;

//复制元素

int i;

for (i = 0; i < Q.queuesize + Q.base - Q.front; i++)

{

newbase[i] = Q.front[i];

}

for (int j = 0; j < Q.rear - Q.base; j++)

{

newbase[i + j] = Q.base[j];

}

//销毁原空间

delete Q.base;

//修改指向,修改元素值

Q.base = newbase;

Q.front = Q.base;

Q.rear = Q.front + Q.length;

Q.queuesize += QUEUEINCREMENT;

}

//插入元素

\*Q.rear = e;

//循环利用队列空间

if (Q.rear - Q.base + 1 == Q.queuesize)

{

Q.rear = Q.base;

}

else

{

Q.rear++;

}

//修改队列元素值

Q.length++;

return OK;

}

//若队列不空，则删除Q的队头元素，并用e返回其值，并返回OK，否则返回FALSE

Status DeQueue\_Sq(SqQueue &Q, int &e)

{

if (Q.length == 0)

{

return FALSE;

}

//取元素，并删除

e = \*Q.front;

//循环利用队列空间

if (Q.front - Q.base + 1 == Q.queuesize)

{

Q.front = Q.base;

}

else

{

Q.front++;

}

//修改队列元素参数

Q.length--;

return OK;

}

//遍历输出队列

Status QueueTraverse\_Sq(SqQueue Q)

{

//队列为空时，

if (Q.length == 0)

{

cout << "队列为空" << endl;

return OK;

}

//队列不为空时

if (Q.front - Q.rear >= 0)

{

for (int i = 0; i < Q.queuesize + Q.base - Q.front; i++)

{

cout << Q.front[i] << '\t';

}

for (int i = 0; i < Q.rear - Q.base; i++)

{

cout << Q.base[i] << '\t';

}

cout << endl;

}

else

{

for (int i = 0; i < Q.length; i++)

{

cout << Q.front[i] << '\t';

}

cout << endl;

}

return OK;

}

//初始化并创建队列

Status CreatQueue\_Sq(SqQueue &Q, int n)

{

int e;

//初始化队列

InitQueue\_Sq(Q);

//输入元素，然后插入队列

for (int i = 0; i < n; i++)

{

cin >> e;

EnQueue\_Sq(Q, e);

}

return OK;

}

CPP

//SqQueue 循环队列实现

//(C++结构体)

#include "Queue\_Sq.h"

//保留数据的个数为 N

#define N 10

//实现获得队列元素最大值

int QueueMaxElem(SqQueue Q)

{

int n = Q.length;

int e;

int max = Q.front[0];

for (int i = 0; i < n; i++)

{

DeQueue\_Sq(Q, e);

if (max<e)

{

max = e;

}

}

return max;

}

//主函数

int main()

{

SqQueue Q;

InitQueue\_Sq(Q);

//队列建立

int n,e=0;

cout << "输入元素个数为：\n";

cin >> n;

for (int i = 0; i < n; i++)

{

//只保留最后 N 个数据

if (Q.length==N)

{

DeQueue\_Sq(Q, e);

}

cin >> e;

EnQueue\_Sq(Q, e);

}

cout << "最终数据为：\n";

QueueTraverse\_Sq(Q);

//得到队列元素最大值

int max;

max = QueueMaxElem(Q);

cout << "最大元素为：" << max << endl;

//空间释放

DestroyQueue\_Sq(Q);

system("pause");

return 0;

}

题目：

CSP题目，窗口点击。

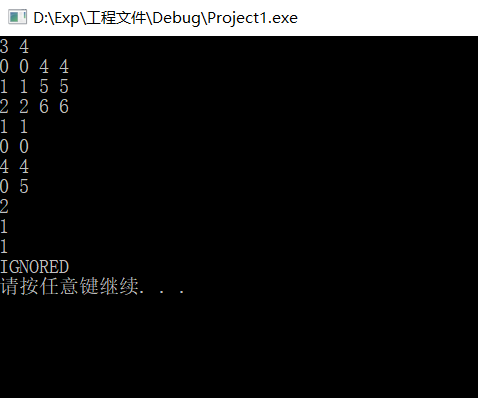
算法思想：

链式结构存储窗口信息。

将每个窗口的信息存放在链表中的一个结点中，针对每条指令依次遍历链表寻找符合条件的窗口结点，当找到被点击到的窗口时删除该结点，插入头结点后的位置。

对链表进行遍历寻找的次数等于指令的个数。

运行结果：



结果分析：

输出结果正确。

附源程序：

//csp 窗口点击问题

#include <iostream>

using namespace std;

struct LNode

{

int x1, y1;

int x2, y2;

int name;

LNode \*next;

};

typedef LNode \*Link;

struct LinkList

{

Link head;

};

void MakeNode(Link &s)

{

s = new LNode;

}

void InsertFirst(LinkList &L, Link s)

{

s->next = L.head->next;

L.head->next = s;

return;

}

void ListDelete(LinkList &L, Link s)

{

Link p = L.head;

while (p->next != s)

p = p->next;

p->next = s->next;

return;

}

struct operation

{

int x, y;

};

int IsInList(operation click, LinkList L, Link &p)

{

p = L.head->next;

while (p)

{

if (p->x1 <= click.x && click.x <= p->x2 && p->y1 <= click.y && click.y <= p->y2)

return p->name;

p = p->next;

}

return 0;

}

int main()

{

LinkList L;

L.head = new LNode;

L.head->next = NULL;

int N, M;

Link s;

cin >> N >> M;

operation \*click;

click = new operation[M];

for (int i = 0; i < N; i++)

{

MakeNode(s);

cin >> s->x1 >> s->y1 >> s->x2 >> s->y2;

s->name = i + 1;

InsertFirst(L, s);

}

for (int i = 0; i < M; i++)

{

cin >> click[i].x >> click[i].y;

}

int name;

Link window;

for (int i = 0; i < M; i++)

{

name = IsInList(click[i], L, window);

if (name)

{

cout << name << '\n';

ListDelete(L, window);

InsertFirst(L, window);

}

else

cout << "IGNORED" << '\n';

}

system("pause");

return 0;

}

二、未调试成功程序及说明

此次程序调试成功。

三、小结

内容：

抽象数据类型：

顺序栈，链式队列，循环队列实现。

应用：

循环队列的应用，链式存储结构的应用。