/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

顺序栈类SeqStack的类定义

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

const int StackSize = 10; //10是示例性的数据，根据实际问题具体定义

template <typename DataType> //定义模板类SeqStack

class SeqStack

{

public:

SeqStack( ); //构造函数，初始化一个空栈

~SeqStack( ); //析构函数

void Push( DataType x ); //入栈操作，将元素x入栈

DataType Pop( ); //出栈操作，将栈顶元素弹出

DataType GetTop( ); //取栈顶元素（并不删除）

int Empty( ); //判断栈是否为空

private:

DataType data[StackSize]; //存放栈元素的数组

int top; //游标，栈顶指针，为栈顶元素在数组中的下标

};

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

顺序栈类SeqStack的构造函数和析构函数

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

template <typename DataType>

SeqStack<DataType> :: SeqStack()

{

top = -1;

}

template <typename DataType>

SeqStack<DataType> :: ~SeqStack()

{

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

顺序栈类SeqStack的入栈操作

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

template <typename DataType>

void SeqStack<DataType> :: Push(DataType x)

{

if (top == StackSize - 1) throw "上溢";

data[++top] = x;

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

顺序栈类SeqStack的出栈操作

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

template <typename DataType>

DataType SeqStack<DataType> :: Pop( )

{

DataType x;

if (top == -1) throw "下溢";

x = data[top--];

return x;

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

顺序栈类SeqStack的判空操作

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

template <typename DataType>

int SeqStack<DataType> :: Empty()

{

if(top == -1)

return 1;

else

return 0;

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

顺序栈类SeqStack的取栈顶操作

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

template <typename DataType>

DataType SeqStack<DataType> :: GetTop( )

{

if(top == -1)

throw "下溢异常";

else

return data[top];

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

对应教材3.2.2节，顺序栈类SeqStack的使用范例

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

#include <iostream>

using namespace std;

const int StackSize = 10; //10是示例性的数据，根据实际问题具体定义

template <typename DataType> //定义模板类SeqStack

class SeqStack

{

public:

SeqStack( ); //构造函数，初始化一个空栈

~SeqStack( ); //析构函数

void Push( DataType x ); //入栈操作，将元素x入栈

DataType Pop( ); //出栈操作，将栈顶元素弹出

DataType GetTop( ); //取栈顶元素（并不删除）

int Empty( ); //判断栈是否为空

private:

DataType data[StackSize]; //存放栈元素的数组

int top; //游标，栈顶指针，为栈顶元素在数组中的下标

};

template <typename DataType>

SeqStack<DataType> :: SeqStack()

{

top = -1;

}

template <typename DataType>

SeqStack<DataType> :: ~SeqStack()

{

}

template <typename DataType>

int SeqStack<DataType> :: Empty()

{

if(top == -1)

return 1;

else

return 0;

}

template <typename DataType>

DataType SeqStack<DataType> :: GetTop( )

{

if(top == -1)

throw "下溢异常";

else

return data[top];

}

template <typename DataType>

void SeqStack<DataType> :: Push(DataType x)

{

if (top == StackSize - 1) throw "上溢";

data[++top] = x;

}

template <typename DataType>

DataType SeqStack<DataType> :: Pop( )

{

DataType x;

if (top == -1) throw "下溢";

x = data[top--];

return x;

}

int main( )

{

int x;

SeqStack<int> S{ }; //定义顺序栈变量S

cout << "对15和10执行入栈操作，";

S.Push(15); S.Push(10);

cout << "当前栈顶元素为：" << S.GetTop( ) << endl; //输出栈顶元素10

try

{

x = S.Pop( );

cout << "执行一次出栈操作，删除元素" << x << endl; //输出出栈元素10

} catch(char\* str){

cout << str << endl;

}

try

{

cout << "请输入待入栈元素：";

cin >> x;

S.Push(x);

} catch(char\* str){

cout << str << endl;

}

if (S.Empty( ) == 1) cout << "栈为空" << endl;

else cout << "栈非空" << endl; //栈有2个元素，输出栈非空

return 0;

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

链栈类LinkStack的类定义

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

template <typename DataType>

struct Node

{

DataType data; //数据域

Node<DataType> \*next; //指针域

};

template <typename DataType>

class LinkStack

{

public:

LinkStack( ); //构造函数，初始化一个空链栈

~LinkStack( ); //析构函数，释放链栈各结点的存储空间

void Push(DataType x); //入栈操作，将元素x入栈

DataType Pop( ); //出栈操作，将栈顶元素出栈

DataType GetTop( ); //取栈顶元素（并不删除）

int Empty( ); //判空操作，判断链栈是否为空栈

private:

Node<DataType> \*top; //栈顶指针即链栈的头指针

};

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

链栈类LinkStack的构造函数

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

template <typename DataType>

LinkStack<DataType> :: LinkStack()

{

top = new Node<DataType>; //生成头结点

top->next = nullptr; //头结点的指针域置空

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

链栈类LinkStack的析构函数

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

template <typename DataType>

LinkStack<DataType> :: ~LinkStack()

{

Node<DataType> \*q = nullptr;

while (top != nullptr) //释放链栈的每一个结点的存储空间

{

q = top; //暂存被释放结点

top = top->next; // top指向被释放结点的下一个结点

delete q;

}

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

链栈类LinkStack的入栈操作

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

template <typename DataType>

void LinkStack<DataType> :: Push(DataType x)

{

Node<DataType> \*s = nullptr;

s = new Node<DataType>; s->data = x; //申请结点s数据域为x

s->next = top; top = s; //将结点s插在栈顶

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

链栈类LinkStack的出栈操作

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

template <typename DataType>

DataType LinkStack<DataType> :: Pop( )

{

Node<DataType> \*p = nullptr;

DataType x;

if (top == nullptr) throw "下溢";

x = top->data; p = top; //暂存栈顶元素

top = top->next; //将栈顶结点摘链

delete p;

return x;

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

链栈类LinkStack的取栈顶操作

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

template <typename DataType>

DataType LinkStack<DataType> :: GetTop()

{

if(top == nullptr)

throw "下溢异常";

else

return top->data;

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

链栈类LinkStack的判空操作

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

template <typename DataType>

int LinkStack<DataType> :: Empty()

{

if(top == nullptr)

throw "下溢异常";

else

return 0;

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

对应教材3.2.3节，链栈类LinkStack的使用范例

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

#include <iostream>

using namespace std;

template <typename DataType>

struct Node

{

DataType data; //数据域

Node<DataType> \*next; //指针域

};

template <typename DataType>

class LinkStack

{

public:

LinkStack( ); //构造函数，初始化一个空链栈

~LinkStack( ); //析构函数，释放链栈各结点的存储空间

void Push(DataType x); //入栈操作，将元素x入栈

DataType Pop( ); //出栈操作，将栈顶元素出栈

DataType GetTop( ); //取栈顶元素（并不删除）

int Empty( ); //判空操作，判断链栈是否为空栈

private:

Node<DataType> \*top; //栈顶指针即链栈的头指针

};

template <typename DataType>

LinkStack<DataType> :: LinkStack()

{

top = new Node<DataType>; //生成头结点

top->next = nullptr; //头结点的指针域置空

}

template <typename DataType>

LinkStack<DataType> :: ~LinkStack()

{

Node<DataType> \*q = nullptr;

while (top != nullptr) //释放链栈的每一个结点的存储空间

{

q = top; //暂存被释放结点

top = top->next; // top指向被释放结点的下一个结点

delete q;

}

}

template <typename DataType>

DataType LinkStack<DataType> :: GetTop()

{

if(top == nullptr)

throw "下溢异常";

else

return top->data;

}

template <typename DataType>

void LinkStack<DataType> :: Push(DataType x)

{

Node<DataType> \*s = nullptr;

s = new Node<DataType>; s->data = x; //申请结点s数据域为x

s->next = top; top = s; //将结点s插在栈顶

}

template <typename DataType>

DataType LinkStack<DataType> :: Pop( )

{

Node<DataType> \*p = nullptr;

DataType x;

if (top == nullptr) throw "下溢";

x = top->data; p = top; //暂存栈顶元素

top = top->next; //将栈顶结点摘链

delete p;

return x;

}

template <typename DataType>

int LinkStack<DataType> :: Empty()

{

if(top == nullptr)

throw "下溢异常";

else

return 0;

}

int main( )

{

int x;

LinkStack<int> S{ }; //定义链栈变量S

cout << "对15和10执行入栈操作，";

S.Push(15); S.Push(10);

cout << "当前栈顶元素为：" << S.GetTop( ) << endl; //输出栈顶元素10

try

{

x = S.Pop( );

cout << "执行一次出栈操作，删除元素：" << x << endl; //输出出栈元素10

} catch(char\* str){

cout << str << endl;

}

try

{

cout << "请输入待插入元素：";

cin >> x;

S.Push(x);

} catch(char\* str){

cout << str << endl;

}

if (S.Empty( ) == 1) cout << "栈为空" << endl;

else cout << "栈非空" << endl; //栈有2个元素，输出栈非空

return 0;

}