《数据结构与算法》实验报告——栈的操作和应用

吉林大学数学学院2020级计算1班 姓名 陈文宇 学号10200115

日期 2023/5/28

一、实验题目

编写栈的基本操作函数，形成头文件后利用栈的基本操作完成数制转换、括号匹配、背包问题求解、Ackman函数的递归求解。

二、算法简述

数制转换算法：将整数取余，余数入栈，更新整数(N=[N/8]）。

括号匹配算法：”(, [, {“入栈，遇到”}, ], )”，若与栈顶元素匹配，栈顶元素出栈，若否，则括号输入规则失败。

背包问题算法：第j个元素开始，遇到第j+i件物品，判断剩余空间，若为正若则入栈，剩余空间减少，若为负，i=i+1重新判断，直至剩余空间为0，输出这组解，若始终剩余空间不为零，则j=j+1;

三、程序代码

|  |
| --- |
| Stack\_Basic\_Operations.cpp |
| //陈文宇  //10200115  #include<iostream>  using namespace std;  const int STACKINIT\_SIZE=100;  const int STACKINCREMENT=10;  const bool TRUE=1;  const bool FALSE=0;  typedef char ElemType;  //顺序栈  typedef struct {  ElemType \*elem;  int top; //栈顶指针  int stacksize; //当前分配的最大容量  int incrementsize; //约定的增补空间量  }SqStack;  //链式栈  typedef struct LNode{ //typedef是用来设置结构体别名，使用过后 定义结构体可以 LNode p ;LNode \*p;有简化的效果  ElemType date;  struct LNode \*next; //结构体自引用-next作为指针 指向 结构体;  }LNode,\*LinkList; // LinkList 是通过结构体声明的指针变量（结构体指针），相当于LNode \*LinkList; LinkList是指向该种结构体的指针  //LNode本身就是一个结构体，但是它的命名是一个指针类型。  typedef LinkList LinkStack;  //函数声明  void InitStack\_Sq(SqStack &S,int maxsize,int incresize);  void DestroyStack\_Sq(SqStack &S);  void ClearStack\_Sq(SqStack &S);  bool StackEmpty\_Sq(SqStack S);  void Push\_Sq(SqStack &S,ElemType e);  void GetTop\_Sq(SqStack S,ElemType &e);  bool Pop\_Sq(SqStack &S,ElemType &e);  void Incrementsize\_Sq(SqStack &S);  void Stack\_Traverse\_Sq(SqStack S);  int main(){    //顺序栈的操作验证  SqStack S;  ElemType e;  //初始化  InitStack\_Sq(S,100,10);    //入栈  Push\_Sq(S,'a');  Push\_Sq(S,'b');  Push\_Sq(S,'c');  Push\_Sq(S,'d');  Stack\_Traverse\_Sq(S);  //出栈  Pop\_Sq(S,e);  Stack\_Traverse\_Sq(S);  //链栈的验证  }  //顺序栈初始化  void InitStack\_Sq(SqStack &S,int maxsize,int incresize){  S.elem = new ElemType[maxsize];  S.top=-1;  S.stacksize=maxsize;  S.incrementsize=incresize;  } //InitStack\_Sq  //顺序栈的销毁  void DestroyStack\_Sq(SqStack &S){  delete[] S.elem;  S.stacksize=0;  S.top=-1;  }  //顺序栈的清空  void ClearStack\_Sq(SqStack &S){  int i;  ElemType e;  for(i=S.top;i>=0;i--) Pop\_Sq(S,e);  }  //判断顺序栈是否为空栈  bool StackEmpty\_Sq(SqStack S){  if(S.top==-1) return TRUE;  else return FALSE;  }  //顺序栈的插入  void Push\_Sq(SqStack &S,ElemType e){  if(S.top+1==S.stacksize) Incrementsize\_Sq(S);//空间扩容  S.elem[++S.top]=e;  }//Push\_Sq  //获取栈顶元素，并用 e 承接  void GetTop\_Sq(SqStack S,ElemType &e){  e=S.elem[S.top];  }  //删除栈顶元素，并用 e 承接  bool Pop\_Sq(SqStack &S,ElemType &e){  if(S.top==-1) return FALSE;  e=S.elem[S.top--];    return TRUE;  }  //栈扩容  void Incrementsize\_Sq(SqStack &S){  ElemType A[S.top+S.incrementsize+1];  ElemType \*B;  for(int i=0;i<=S.top;i++) A[i]=S.elem[i];  S.stacksize=S.top+S.incrementsize+1;  B=S.elem;  S.elem=A;  free(B);  }  //顺序栈输出  void Stack\_Traverse\_Sq(SqStack S){  int i;  ElemType e;  if(S.top==-1) printf("空栈");  for(i=S.top;i>=0;i--){  Pop\_Sq(S,e);  cout<<e<<endl;  }  }  //链栈的基本操作  //初始化  void InitStack\_L(LinkStack &S){  S=NULL;  }//InitStack\_L  //元素入栈  void Push\_L(LinkStack &S,ElemType e){  LNode \*p;  p=new LNode;  p->date=e;  p->next=S;  S=p;    }//Push\_L  //栈顶元素出栈  bool Pop\_L(LinkStack &S,ElemType &e){  if(S==NULL) return FALSE;  else{  LNode \*p;  p=S;  e=p->date;  S=p->next;  delete p;  }  } |
| 数字转换问题.cpp |
| //陈文宇  //10200115  #include"Stack\_headerfile.h"  int main(){    //顺序栈的操作验证  SqStack S;  ElemType N;  cout<<"十进制为:";  cin>>N;    //初始化  InitStack\_Sq(S,100,10);    //入栈  while(N!=0){  Push\_Sq(S,N%8);  N=N/8;  }  //输出    cout<<"八进制为:";  Stack\_Traverse\_Sq(S);  } |
| 背包问题.cpp |
| //陈文宇  //10200115  #include"Stack\_headerfile.h"  void knapsack(int\* w,int T,int n);  int main(){    //顺序栈的操作验证  int T=10;  int w[6]={1,8,4,3,5,2};//背包  printf("物品大小：");  for (int i=0; i<6; i++)  printf("%d ", \*(w + i));  printf("\n");  cout<<"全部解在背包的位序"<<endl;  knapsack(w,T,6);  }  void knapsack(int\* w,int T,int n){  int k=0;  SqStack S;    //初始化  InitStack\_Sq(S,100,10);  while(!StackEmpty\_Sq(S) || k<n){  while(T>0 && k<n){  if(T-w[k]>=0){  Push\_Sq(S,k);  T-=w[k];  }  k++;  }//while 第k件可选，则入栈    if(T==0) {  Stack\_Traverse\_Sq(S);  printf("\n");  }// if 如果为解，则输出解    //栈顶元素出栈，栈顶元素为最后一个可选元素的位序  //故从第k+1元素开始再次筛选  //如果后面没有解存在，则再次将栈顶元素出栈  Pop\_Sq(S,k);  T+=w[k];  k++;  }//while  } |
| 括号匹配检验.cpp |
| //陈文宇  //10200115  #include"Stack\_headerfile.h"  bool matching(char exp[]);  int main(){    char w[10]={'(','(','[',']','{','}',')',')','#'};  char x[10]={'(','(','[',')','{','}',')',')','#'};    printf("%s\n",w);  cout<<matching(w)<<endl;    printf("%s\n",x);  cout<<matching(x)<<endl;  }  bool matching(char exp[]){  int state=1;  ElemType e;  SqStack S;  char ch;    InitStack\_Sq(S,100,10);  ch=\*exp++;    while(ch!='#' && state){  switch(ch){  case'(':  case'[':  case'{':{  Push\_Sq(S,ch);  break;  }  case')':{  GetTop\_Sq(S,e);  if(!StackEmpty\_Sq(S) && e=='(')  Pop\_Sq(S,e);  else state=0;  break;  }  case']':{  GetTop\_Sq(S,e);  if(!StackEmpty\_Sq(S) && e=='[')  Pop\_Sq(S,e);  else state=0;  break;  }  case'}':{  GetTop\_Sq(S,e);  if(!StackEmpty\_Sq(S) && e=='{')  Pop\_Sq(S,e);  else state=0;  break;  }    }//switch  ch=\*exp++;  }//while  if(state && StackEmpty\_Sq(S)) return TRUE;  else return FALSE;  } //matching |
| Ackerman函数.cpp |
| //10200115  //陈文宇  #include<iostream>  using namespace std;  const int STACKINIT\_SIZE=100;  const int STACKINCREMENT=10;  const bool TRUE=1;  const bool FALSE=0;  typedef struct ElemType{  int nval;  int xval;  int yval;  }ElemType;  typedef struct SqStack{  ElemType \*elem;  int top; //栈顶指针  int stacksize; //当前分配的最大容量  int incrementsize; //约定的增补空间量  }SqStack;  //顺序栈初始化  void InitStack\_Sq(SqStack &S,int maxsize,int incresize){    S.elem = new ElemType[maxsize];    S.top=-1;    S.stacksize=maxsize;    S.incrementsize=incresize;  } //InitStack\_Sq  //栈扩容  void Incrementsize\_Sq(SqStack &S){  ElemType A[S.top+S.incrementsize+1];  ElemType \*B;  for(int i=0;i<=S.top;i++) A[i]=S.elem[i];    S.stacksize=S.top+S.incrementsize+1;    B=S.elem;  S.elem=A;  free(B);  }  //判断顺序栈是否为空栈  bool StackEmpty\_Sq(SqStack S){  if(S.top==-1) return TRUE;  else return FALSE;  }  //获取栈顶元素，并用 e 承接  void GetTop\_Sq(SqStack S,ElemType &e){  e=S.elem[S.top];  }  //删除栈顶元素，并用 e 承接  bool Pop\_Sq(SqStack &S,ElemType &e){  if(S.top==-1) return FALSE;  e=S.elem[S.top--];    return TRUE;  }  //顺序栈的插入  void Push\_Sq(SqStack &S,ElemType e){  if(S.top+1==S.stacksize) Incrementsize\_Sq(S);//空间扩容  S.elem[++S.top]=e;  }//Push\_Sq  int value(int n,int x,int y){  if(n==0) return x+1;  else switch(n){  case 1:return x;  case 2:return 0;  case 3:return 1;  default :return 2;  }//switch  }//value  int Ackerman(int n,int x,int y){  ElemType e;  SqStack S;  int u;    InitStack\_Sq(S,100,10);  e.nval=n; e.xval=x; e.yval=y; Push\_Sq(S,e);    while(!StackEmpty\_Sq(S)){    GetTop\_Sq(S,e);  while(e.nval!=0 && e.yval!=0){  e.yval--;  Push\_Sq(S,e);  }//while    Pop\_Sq(S,e);  u=value(e.nval,e.xval,e.yval);    if(!StackEmpty\_Sq(S)){  Pop\_Sq(S,e);  e.nval--; e.yval=e.xval; e.xval=u;  Push\_Sq(S,e);  }//if  }//while    return u;  }//Ackerman  int main(){  cout<<Ackerman(3,2,1)<<endl;  } |

四、实验结果







