Feladat

Olvassuk be a standard inputról érkező számokat, és írjuk ki őket fordított sorrendben a standard outputra!

Megoldás

A feladat megoldásához készítünk egy egész számokat tartalmazó verem-típust. A standard inputról beolvasott számokat belerakjuk egy verembe, majd a beolvasás után a vermet kiürítjük standard outputra. A vermet egy egyirányú, fejelem nélküli láncolt listával fogjuk ábrázolni.

Megoldás C++-ban

Verem-típus

Header fájl

A verem-típust egy osztállyal valósítjuk meg. Az osztály definícióját a header fájlban (**`stack.h`**) helyezzük el. Az osztály publikus interfésze a szokásos verem-műveleteket tartalmazza:

```
#ifndef STACK_H
#define STACK_H

class Stack{
  public:
    enum Exceptions{EMPTYSTACK};

    Stack();
    ~Stack();

    void Push(int e);
    int  Pop();
    int  Top();
    bool Empty();
```

Az **`Exceptions`** felsorolt típus azokat az értékeket tartalmazza, amelyeket a verem-osztály hibás felhasználás esetén kivételként dob. Jelen esetben a verem csak akkor dob kivételt, ha egy üres veremből ki akarunk venni egy érték, vagy egy üres verem tetején levő elemet akarjuk használni. (**`Pop`** és **`Top`** műveletek)

Az osztály privát részében deklaráljuk a copy konstruktort és az értékadás operátort: így letiltjuk a verem-objektumok érték szerinti paraméterátadását és az értékadását. Ezt a listaelem-típus definíciója követi. A verem reprezentációja a legelső listaelemre mutató **head** pointer.

```
    private:
      Stack(const Stack&);
      Stack& operator=(const Stack&);

      struct Node{
          int  val;
          Node *next;
          Node(const int& e, Node *n) : val(e), next(n){}
      };

      Node *head;
};

#endif
```

Implementációs fájl

Az osztály implementációs fálja (**stack.cpp**) tartalmazza a műveletek megvalósítását. A konstruktor egy üres vermet, azaz egy nulla hosszúságú láncolt listát inicializál, a destruktor felszabadítja a vermet ábrázoló listaelemeket:

```
#include "stack.h"
#define nil 0

Stack::Stack(): head(nil){}

Stack::~Stack()
{
    Node *p;
    while(head!=nil){
        p = head;
        head = head->next;
        delete p;
    }
}
```

A **nil** a sehova sem mutató cím, amelyet a fájl elején a **#define nil 0** sor definiál.

A **Push** művelet létrehoz egy új listaelemet, és befűzi azt a lista elejére:

```
void Stack::Push(int e)
{
    head = new Node(e,head);
}
```

A **Pop** művelet üres lista esetén **EMPTYSTACK** kivételt dob, egyébként kifűzi a lista legelső elemét, majd felszabadítja azt:

```
int Stack::Pop()
{
    if(head==nil) throw EMPTYSTACK;
    int e = head->val;
    Node *p = head;
    head = head->next;
    delete p;
    return e;
}
```

A **Top** művelet üres lista esetén **EMPTYSTACK** kivételt dob, egyébként visszaadja a lista legelső elemében tárolt értéket:

```
int Stack::Top()
{
    if(head==nil) throw EMPTYSTACK;
    return head->val;
}
```

Az **Empty** művelet üres lista esetén add igaz értéket, egyébként hamis értékkel tér vissza:

```
bool Stack::Empty()
{
    return head==nil;
}
```

A főprogram

A főprogramban létrehozunk egy verem-objektumot, majd a standard inputról érkező számokat belerakjuk:

```
#include <iostream>
#include <string>
#include "stack.h"
using namespace std;

int main()
{
    Stack s;
    int i;
    while(cin >> i){
        s.Push(i);
    }
    while(!s.Empty()){
        cout << s.Pop() << endl;
    }
  return 0;
}
```

Egy újabb feladat

Olvassuk be a standard inputról érkező szavakat, és írjuk ki őket fordított sorrendben a standard outputra.

Megoldás

Ez a feladat analóg az előzővel. A főprogram mindössze abban tér el, hogy nem egészeket tartalmazó vermet, hanem karakterláncokat tartalmazó vermet használ. A verem-osztályból egy osztálysablon készítünk, amelyet a verem elemeit adó értékek típusával lehet paraméterezni. Ekkor a főprogramban elegendő megjelölni, hogy éppen milyen elemi típusra épülő verem-objektumot akarunk használni.

Megoldás C++-ban

A főprogram

A főprogramban egy **`string`** típussal paraméterezett osztályt használunk a verem-objektum definiálására:

```
#include <iostream>
#include <string>
#include "stack.h"

using namespace std;

int main()
{
    Stack<string> s;

    string i;
    cin >> i;
    while(i.compare("q")){
        s.Push(i);
        cin >> i;
    }

    while(!s.Empty()){
        cout << s.Pop() << endl;
    }

  char ch;
  cin >> ch;
  return 0;
}
```

A szavak beolvasása természetesen némileg eltér az előző feladat megoldásában látott számok beolvasásától.

Verem-típus osztálysablonja

Nézzük meg, hogyan lehet a verem-típus az elemi típustól függetlenül definiálni:

```
template <class Element>
class Stack{
  public:
    enum Exceptions{EMPTYSTACK};

    Stack();
    ~Stack();
    Stack(const Stack& s);
    Stack& operator=(const Stack& s);

    void Push(const Element& e);
    Element Pop();
    Element Top();
    bool Empty();

  private:
    struct Node{
        Element val;
        Node *next;
        Node(const Element& e, Node *n) : val(e), next(n){};
    };
    Node *head;
};
```

A fenti kódrészben a **`template <class Element>`** sor jelzi, hogy a verem-osztály definíciójában szereplő **`Element`** osztály egy később megjelölendő típus. Definiálni fogjuk a copy konstruktort és az értékadás oparátort is.
Az osztály-definíción kívül ezzel a sorral vezetjük be az egyes típusműveletek megvalósítását is. Itt az osztálynevére történő hivatkozásokat mindenhol **`Stack<Element>`**-re cseréljük ki:

```
template <class Element>
Stack<Element>::Stack(): head(nil){}

template <class Element>
Stack<Element>::~Stack()
{
    Node *p;
    while(head!=nil){
        p = head;
        head = head->next;
        delete p;
    }
}
```

A **Push** műveletnél a bejövő paraméter érték szerinti átadás helyett referencia szerinti átadásra módosítottuk, hogy majd ne kelljen meghívni az **Element** helyébe kerülő osztály copy konstruktorát:

```
template <class Element>
void Stack<Element>::Push(const Element& e)
{
    head = new Node(e,head);
}

template <class Element>
Element Stack<Element>::Pop()
{
    if(head==nil) throw EMPTYSTACK;
    Element e = head->val;
    Node *p = head;
    head = head->next;
    delete p;
    return e;
}

template <class Element>
Element Stack<Element>::Top()
{
    if(head==nil) throw EMPTYSTACK;
    return head->val;
}


template <class Element>
bool Stack<Element>::Empty()
{
    return head==nil;
}
```

A copy konstruktor üres inicializáló verem-objektum esetén egy üres láncolt listát, egyébként egy új láncolt listát épít fel. Ez utóbbi ugyanazokat az értékeket, ugyanolyan sorrendben tartalmazza, mint az inicializáló verem láncolt listája:

```
template <class Element>
Stack<Element>::Stack(const Stack<Element>& s)
{
    if(s.head==nil) head = nil;
    else {
        head = new Node(s.head->val,nil);
        Node *q = head;
        Node *p = s.head->next;
        while(p!=nil){
            q->next = new Node(p->val,nil);
            q = q->next;
            p = p->next;
        }
    }
}
```

Az értékadás operátor – feltéve, hogy az értékadás két oldalán található verem-objektumok nem azonosak – az alapobjektum által foglalt listaelemek felszabadításából (lásd destruktor), majd az inicializáló objektum alapján történő felépítéséből (lásd konstruktor) áll. Mindez kiegészül a megfelelő visszatérési érték megadásával:

```
template <class Element>
Stack<Element>& Stack<Element>::operator=
                      (const Stack<Element>& s)
{
    if(this==&s) return *this;

    Node *p;
    while(head!=nil){
        p = head;
        head = head->next;
        delete p;
    }

    if(s.head==nil) head = nil;
    else {
        head = new Node(s.head->val,nil);
        Node *q = head;
        Node *p = s.head->next;
        while(p!=nil){
            q->next = new Node(p->val,nil);
            q = q->next;
            p = p->next;
        }
    }

    return *this;
}
```

Az így definiált verem-típus nem teljesen független az elemi típustól, mert annak lefordításához több helyen is (például **`Top`** művelet visszatérési értékénél vagy a **`Node`** típus konstruktorában) szükség van az elemi típus publikus copy konstruktorára. Ez azonban csak a példányosításnál derül ki. Ezért **az osztálysablon műveleteinek megvalósítását is a header fájlban helyezzük el**.

Osztálysablon által dobott kivétel elkapása

Mivel egy osztálysablonból több osztályt is lehet példányosítani, a különböző példányokból dobott kivételeket meg kell különböztetni:

```
Stack<int>    si;
Stack<char>   sa;
Stack<string> ss;

si.Push(4);
sa.Push('w');
ss.Push("alma");

try{
    cout << si.Pop;
    cout << sa.Pop;
    cout << ss.Pop;
}catch(Stack<int>::Exceptions e){
    if(e==Stack<int>::EMPTYSTACK){
        cout << "üres verem" << endl;
}catch(Stack<sting>::Exceptions e){
    if(e==Stack<string>::EMPTYSTACK){
        cout << "üres verem" << endl;
}
```