Objektum elvű alkalmazások fejlesztése

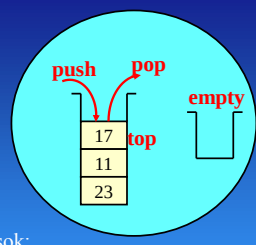
Verem típus osztály-sablonja

Készítette: Gregorics Tibor

Feladat

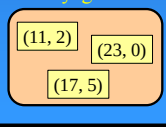
Készítsünk olyan újra-felhasználható kódot, amellyel vermet lehet létrehozni és használni. A példányosításnál tetszőlegesen választhassuk meg a verem elemeinek típusát, valamint a verem konkrét reprezentációját, amely lehet statikus (tömbös azaz aritmetikai), illetve dinamikus (egyirányú, fejelem nélküli láncolt listás) ábrázolás! Különítsük el a verem interfészét a reprezentációjától!

Verem-típus specifikáció

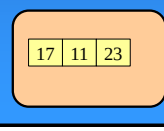


Absztrakt ábrázolások:

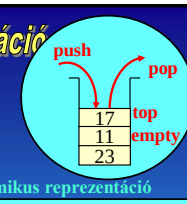
Időbélyeges elemek halmaza:



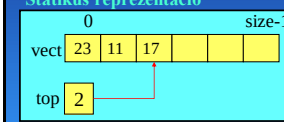
Sorozat:



Kétféle konkrét reprezentáció

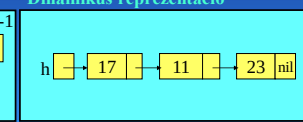


Statikus reprezentáció



1. Rögzített a lefoglalt memória mérete (felesleges vagy kevés)
2. Tömbnyi szabad memória kell
3. Adatelemek elérési ideje konstans (A top illetve a vect[top] elérésére van szükség.)

Dinamikus reprezentáció



1. Valódi mérethez igazodik a lefoglalt memória (de a címeket is tárolja)
2. Sok kisméretű memória szelet kell
3. Adatelemek elérési ideje általában lineáris, de például az első elem elérési ideje konstans.

Osztály-sablon példányosítás

Stack<Item> osztály-sablon példányosításai ahol **Item** az elemi-típus paramétere

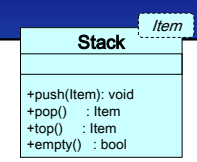
```
Stack<int>     si1;        Stack<int> si2(20);
Stack<string>  ss;
Stack<Test*>   st;

si1.push(4);
ss.push("alma");
Kup *o = new Kup(2.0, 4.5);
st.push(o);
```

verem maximális mérete statikus reprezentációhoz

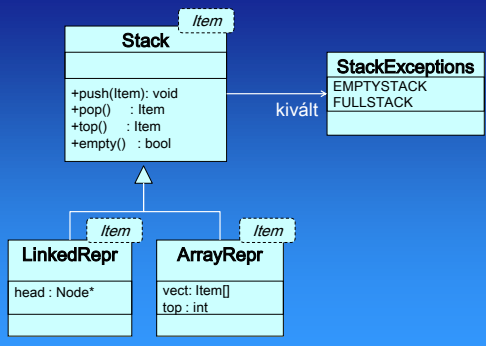
main.cpp

Verem-típus osztály-sablonjának publikus része

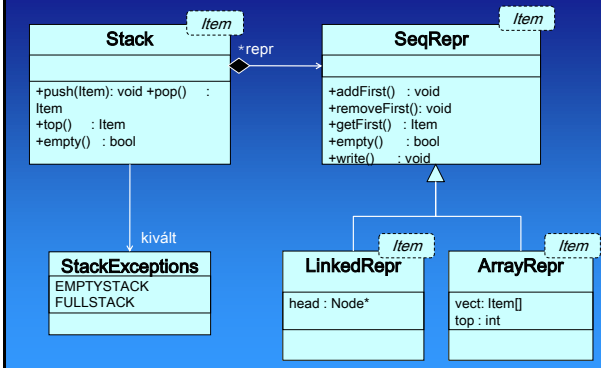
```
template <typename Item>
class Stack
{
   public:
      Stack(int max = 0);

      void push(const Item &e);
      Item pop();
      Item top()    const;
      bool empty()  const;

   private:
      …
};
```



Osztály diagram 1. változat



Osztály diagram 2. változat



Verem-típus sablon-metódusai

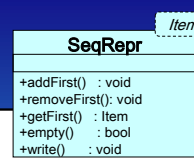
```
template <typename Item>
Stack<Item>::Stack(int max = 0){
   if (max==0) repr = new LinkedRepr<Item>();
   else        repr = new ArrayRepr<Item>(max);
}

template <typename Item>
void Stack<Item>::push(const Item &e){ repr->addFirst(e); }

template <typename Item>
Item Stack<Item>:: pop(){
   Item e = repr->getFirst();
   repr->removeFirst();
   return e;
}

template <typename Item>
Item Stack<Item>:: top()   const { return repr->getFirst(); }

template <typename Item>
bool Stack<Item>:: empty() const { return repr->empty(); }
```

Verem-típus műveletei „inline” módon

```
template <typename Item>
class Stack
{
   public:
      Stack(int max = 0){
         if (max==0) repr = new LinkedRepr<Item>();
         else        repr = new ArrayRepr<Item>(max);
      }
      void push(const Item &e){ repr->addFirst(e); }
      Item pop(){
         Item e = repr->getFirst();
         repr->removeFirst();
         return e;
      }
      Item top()   const { return repr->getFirst(); }
      bool empty() const { return repr->empty(); }

   private:
      SeqRepr<Item> *repr;
};
```

stack.hpp

Reprezentáció



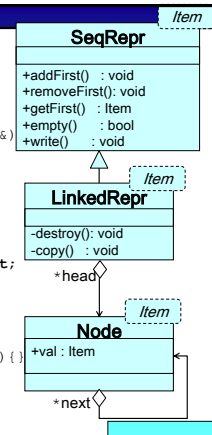

```
template <typename Item>
class SeqRepr
{
   public:
      virtual void addFirst(const Item &e) = 0 ;
      virtual void removeFirst() = 0 ;
      virtual Item getFirst() const = 0 ;
      virtual void empty() const = 0 ;
      virtual void write(std::ostream&) const = 0 ;
      virtual ~SeqRepr(){};
};
```

seqrepr.hpp

Láncolt reprezentáció

```
template <typename Item>
class LinkedRepr : public SeqRepr<Item>{
public:
   LinkedRepr(){ head = NULL;}
   LinkedRepr(const LinkedRepr&);
   LinkedRepr& operator=(const LinkedRepr&);
   ~LinkedRepr(){ destroy(); }

   virtual void addFirst(const Item &e);
   virtual void removeFirst();
   virtual Item getFirst() const;
   virtual void empty() const;
   virtual void write(std::ostream&) const;
private:
   struct Node{
      Item val;
      Node *next;
      Node(int e, Node *n):val(e), next(n){}
   };
   Node *head;
   void destroy();
   void copy(const LinkedRepr&);
};
```

Node<Item>



linkedrepr.hpp

Láncolt reprezentáció metódusai

```
template <typename Item>
void LinkedRepr<Item>::addFirst(const Item& e) {
   try{
      head = new Node(e,head);
   }catch(std::bad_alloc o) { throw FULLSTACK; }
}
template <typename Item>
void LinkedRepr<Item>::removeFirst() {
   if(head==NULL) throw EMPTYSTACK;
   Node *p = head;
   head = head->next;
   delete p;
}
template <typename Item>
Item LinkedRepr<Item>::getFirst() const {
   if(head==NULL) throw EMPTYSTACK;
   return head->val;
}
template <typename Item>
bool LinkedRepr<Item>::empty() const {
   return head==NULL;
}
```

linkedrepr.hpp

Láncolt reprezentáció - segédfüggvények

```
template <typename Item>
void LinkedRepr<Item>::copy(const LinkedRepr& r){
   if(r.head==NULL) head = NULL;
   else {
      try{
         head = new Node(r.head->value,NULL);
         Node *q = head; Node *p = r.head->next;
         while(p!=NULL){
            q->next = new Node(p->value,NULL);
            q = q->next; p = p->next;
         }
      }catch(std::bad_alloc o){
         destroy();
         throw FULLSTACK;
      }
   }
```

```
template <typename Item>
void LinkedRepr<Item>::destroy(){
   Node *p;
   while(head!=NULL){
      p = head;
      head = head->next;
      delete p;
   }
}
```

linkedrepr.hpp

Láncolt reprezentáció – másoló konstruktor, értékadás operátor

```
template <typename Item>
LinkedRepr<Item>::LinkedRepr(const LinkedRepr& r){
   copy(r);
}

template <typename Item>
LinkedRepr<Item>& LinkedRepr<Item>::
operator=(const LinkedRepr& r){
   if(this==&r) return *this;
   destroy();
   copy(r);
   return *this;
}
```

linkedrepr.hpp

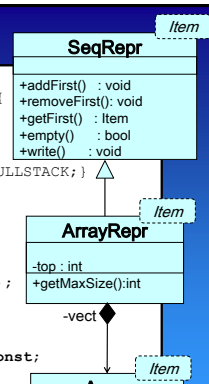
Láncolt reprezentáció – kiírás

```
template <typename Item>
void LinkedRepr<Item>::write(std::ostream& o) const
{
    o << "[";
    Node<Item> *p = head;
    if(p!=NULL){
        o << " " << p->val;
        p = p->next;
    }
    while(p!=NULL){
        o << ", " << p->val ;
        p = p->next;
    }
    o << " ]";
}
```

linkedrepr.hpp

Tömbös reprezentáció

```
template <typename Item>
class ArrayRepr : public SeqRepr<Item>{
public:
   ArrayRepr(int max) {
      try{ vect.resize(max); }
      catch(std::bad_alloc o){ throw FULLSTACK;}
      top=-1;
   }
   Int getMaxSize() const {
      return (int)vect.size();
   }
   virtual void addFirst(const Item &e);
   virtual void removeFirst();
   virtual Item getFirst() const;
   virtual void empty() const;
   virtual void write(std::ostream&) const;

private:
   vector<Item> vect;
   int  top;
};
```



arrayrepr.hpp

Tömbös reprezentáció metódusai

```
template <typename Item>
void ArrayRepr<Item>::addFirst(const Item& e){
   if(top+1==(int)vect.size()) throw FULLSTACK;
   vect[++top] = e;
}

template <typename Item>
void ArrayRepr<Item>::removeFirst(){
   if(top==-1) throw EMPTYSTACK;
   --top;
}

template <typename Item>
Item ArrayRepr<Item>::getFirst() const{
   if(top==-1) throw EMPTYSTACK;
   return vect[top];
}

template <typename Item>
bool ArrayRepr<Item>::empty() const
{ return top==-1; }
```

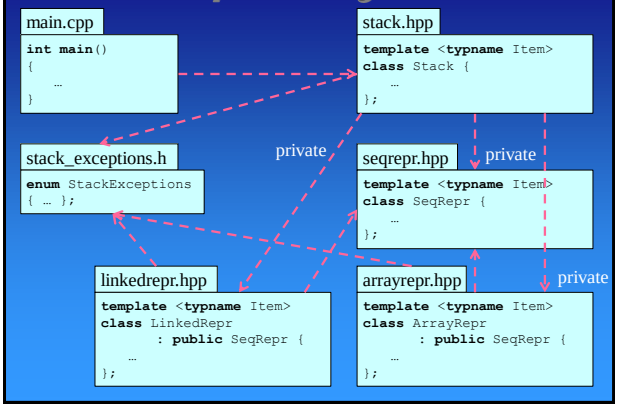
arrayrepr.hpp

Tömbös reprezentáció – kiírás

```
template <typename Item>
void ArrayRepr<Item>::write(std::ostream& o) const
{
    o << "[";
    if(top>=0){
        o << " " << vect[top];
    }
    for(int i=top-1 ; i>=0 ; --i){
        o << ", " << vect[i];
    }
    o << " ]";
}
```

linkedrepr.hpp

Telepítési diagram



Barát függvény-sablon

```
template <typename Item>
class Stack;

template <typename Item>
std::ostream& operator<<(std::ostream& out, const Stack<Item>& s);

template <typename Item>
class Stack
{
  public:
    …
  friend
  std::ostream& operator<< <> (std::ostream& out, const Stack& s);
};

template <typename Item>
std::ostream& operator<< (std::ostream& out, const Stack<Item>& s)
{
    s.repr->write(out);
    return out;
}
```

stack.hpp

Fő program

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

int main()
{
   Stack<int> s;      // Stack<int> s(10);
   try{
      int i;
      while( cin >> i ){
         s.push(i);
      }
   }catch (StackExceptions e){
      if (e == FULLSTACK) cout << "…";
   }
   cout << s << endl;

   while(!s.Empty()){
      cout << s.pop() << endl;}
   }
  return 0;
}
```

main.cpp

Alapértelmezett sablon-paraméterek

```
template <typename Item = int>
class Stack{
public:
   …
};
```

```
Stack<int> y;
Stack<> y;

Stack<Test*> y;
```

```
template <typename Item, int maxsize = 100>
class ArrayRepr{
public:
   ArrayRepr() {
      try{ vect.resize(maxsize); }
      catch(std::bad_alloc o){ throw FULLSTACK;}
      top=-1;
   }
   …
};
```

```
Stack<int, 10> y;
Stack<int> y;
```