Változó méretű tömb

Darvay Zsolt

Babeş–Bolyai Tudományegyetem, Kolozsvár

Tartalomjegyzék

Változó méretű tömb

```
#include <iostream>

using namespace std;
class bejaro; // előzetes deklaráció

class tomb {
   int* t;
   int n;
   int max;
   int delta;
```

A tomb osztály metódusai

```
public:
   tomb(int* e, int hossz,
      int maxhossz = 10, int d = 2);
   ~tomb();
   void hozzaad(int a);
   void kiir();
   int& operator[](int a);
   int getn();
   bejaro* letrehozbejaro();
```

Belső osztály

```
// folytatódik a tomb osztály
  class iterator // belső osztály
  {
  private:
    int *aktualis;
  public:
    iterator(int *a);
    iterator &operator++();
    iterator operator++(int);
    bool operator ==(const iterator &a);
    bool operator !=(const iterator &a);
    int &operator *();
  };
// tovább folytatódik a tomb osztály
```

A tomb osztály további metódusai

```
// folytatódik a tomb osztály
   iterator begin();
   iterator end();
};
```

A bejaro osztály

```
class bejaro {
   tomb *v;
   int aktualis;
public:
   bejaro(tomb *v);
   bejaro& operator++();
   bejaro& operator++(int);
   int& operator*();
   bool vege();
};
```

A bejaro osztály metódusai

```
bejaro::bejaro(tomb *v) {
  aktualis = 0;
  this->v = v;
}

bejaro & bejaro::operator++() {
  aktualis++;
  return *this;
}

bejaro & bejaro::operator++(int) {
  aktualis++;
  return *this;
}
```

A bejaro osztály metódusai

```
int & bejaro::operator*()
{
  return (*v)[aktualis];
}

bool bejaro::vege()
{
  return aktualis >= v->getn();
}
```

A tomb osztály konstruktora és destruktora

```
tomb::tomb(int* e, int hossz, int maxhossz, int d) {
  n = hossz;
  max = maxhossz;
  delta = d;
  t = new int[max];
  for (int i = 0;i < n;i++) {
    t[i] = e[i];
  }
}

tomb::~tomb() {
  delete[] t;
}
```

Új elem hozzáadása

```
void tomb::hozzaad(int a)
{
  if (n == max)
  {
    max += delta;
    int *p = new int[max];
    for (int i = 0; i < n; i++)
    {
      p[i] = t[i];
    }
    delete[]t;
    t = p;
  }
  t[n++] = a;
}
```

Kiírás

```
void tomb::kiir()
{
   for (int i = 0; i < n; i++)
   {
      cout << t[i] << " ";
   }
   cout << "Maximum: " << max << " \n";
}
```

További metódusok

```
int& tomb::operator[](int a) {
  if (a < 0 || a >= n) {
    throw "Hiba";
  }
  return t[a];
}

int tomb::getn() {
  return n;
}

bejaro* tomb::letrehozbejaro() {
  return new bejaro(this);
}
```

A begin és end tagfüggvények

```
tomb::iterator tomb::begin()
{
  return iterator(t);
}

tomb::iterator tomb::end()
{
  return iterator(t + n);
}
```

A belső osztály metódusai

```
tomb::iterator::iterator(int *a) {
  aktualis = a;
}

tomb::iterator& tomb::iterator::operator++() {
  aktualis++;
  return *this;
}

tomb::iterator tomb::iterator::operator++(int)
{
  iterator temp(*this);
  aktualis++;
  return temp;
}
```

A belső osztály metódusai

```
bool tomb::iterator::operator==(const iterator& a)
{
  return aktualis == a.aktualis;
}

bool tomb::iterator::operator!=(const iterator& a)
{
  return aktualis != a.aktualis;
}

int &tomb::iterator::operator*()
{
  return *aktualis;
}
```

A fő függvény

```
int main()
{
  int t[] = { 1, 2, 3, 4, 5, 6, 7, 8 };
  tomb v(t, 8);
  v.kiir();
  v.hozzaad(9);
  v.hozzaad(10);
  v.kiir();
  v.hozzaad(11);
  v.kiir();
  v.hozzaad(12);
  v.hozzaad(13);
  v.kiir();
```

Bejárás első változata

```
bejaro *b;
for (b = v.letrehozbejaro();!b->vege();(*b)++)
{
   cout << **b << " ";
}
cout << endl;
delete b;
```

Bejárás más változatai

```
bejaro &b2 = *v.letrehozbejaro();
for (; !b2.vege(); b2++) {
   cout << *b2 << ' ';
}
cout << endl;
delete &b2;

bejaro b3(&v);
for (; !b3.vege(); b3++) {
   cout << *b3 << ' ';
}
cout << endl;
```

Bejárás iterátorral és általánosított for utasítással

```
 for (tomb::iterator i = v.begin();i != v.end();i++)
 {
  cout << *i << " ";
 }
 cout << endl;

 for (int elem : v)
  cout << elem << " ";
 cout << endl;
}
```

Kimenet

```
1 2 3 4 5 6 7 8 Maximum: 10
1 2 3 4 5 6 7 8 9 10 Maximum: 10
1 2 3 4 5 6 7 8 9 10 11 Maximum: 12
1 2 3 4 5 6 7 8 9 10 11 12 13 Maximum: 14
1 2 3 4 5 6 7 8 9 10 11 12 13
1 2 3 4 5 6 7 8 9 10 11 12 13
1 2 3 4 5 6 7 8 9 10 11 12 13
1 2 3 4 5 6 7 8 9 10 11 12 13
1 2 3 4 5 6 7 8 9 10 11 12 13
```

Változó méretű tömb sablonokkal

```
#include <iostream>
#include <algorithm>
using namespace std;
template <class T>
class bejaro;
template <class T>
class tomb {
   T* t;
   int n;
   int max;
   int delta;
```

Az osztály metódusai

```
public:
   tomb(T* e, int hossz,
      int maxhossz = 10, int d = 2);
   ~tomb();
   void hozzaad(T a);
   void kiir();
   T& operator[](int a);
   int getn();
   bejaro<T>* letrehozbejaro();
```

A belső osztály

```
class iterator
//class iterator :
  //public std::iterator<std::input_iterator_tag, T>
//class iterator : public
  //std::iterator<std::random_access_iterator_tag, T>
{
  //typedef iterator self_type;
  //typedef std::input_iterator_tag iterator_category;
  //typedef T value_type;
  //typedef int difference_type;
  //typedef T& reference;
  //typedef T* pointer;
```

A belső osztály metódusai

```
private:
   T* aktualis;
public:
   iterator(T* a);
   iterator& operator++();
   iterator operator++(int);
   iterator& operator--();
   bool operator ==(const iterator& a);
   bool operator !=(const iterator& a);
   T& operator *();
   int operator -(const iterator& a);
   bool operator <(const iterator& a);
};
```

A begin és end tagfüggvények

```
// folytatódik a tomb osztály
   iterator begin();
   iterator end();
};
```

A bejaro osztály

```
template <class T>
class bejaro {
   tomb<T>* v;
   int aktualis;
public:
   bejaro(tomb<T>* v);
   bejaro& operator++();
   bejaro& operator++(int);
   T& operator*();
   bool vege();
};
```

A bejaro osztály konstruktora

```
template <class T>
bejaro<T>::bejaro(tomb<T>* v)
{
   aktualis = 0;
   this->v = v;
}
```

A bejaro osztály ++ operátora

```
template <class T>
bejaro<T>& bejaro<T>::operator++()
{
  aktualis++;
  return *this;
}

template <class T>
bejaro<T>& bejaro<T>::operator++(int)
{
  aktualis++;
  return *this;
}
```

További metódusok

```
template <class T>
T& bejaro<T>::operator*()
{
  return (*v)[aktualis];
}

template <class T>
bool bejaro<T>::vege()
{
  return aktualis >= v->getn();
}
```

A tomb osztály konstruktora és destruktora

```
template <class T>
tomb<T>::tomb(T* e, int hossz, int maxhossz, int d) {
  n = hossz;
  max = maxhossz;
  delta = d;
  t = new T[max];
  for (int i = 0; i < n; i++) {
    t[i] = e[i];
  }
}

template <class T>
tomb<T>::~tomb() {
  delete[] t;
}
```

A hozzaad metódus

```
template <class T>
void tomb<T>::hozzaad(T a) {
  if (n == max)
  {
    max += delta;
    T* p = new T[max];
    for (int i = 0; i < n; i++)
    {
      p[i] = t[i];
    }
    delete[]t;
    t = p;
  }
  t[n++] = a;
}
```

Kiírás

```
template <class T>
void tomb<T>::kiir()
{
   for (int i = 0; i < n; i++)
   {
      cout << t[i] << " ";
   }
   cout << "Maximum: " << max << endl;
}
```

A [] operátor

```
template <class T>
T& tomb<T>::operator[](int a)
{
   if (a < 0 || a >= n) {
      throw "Hiba";
   }
   return t[a];
}
```

További metódusok

```
template <class T>
int tomb<T>::getn()
{
   return n;
}

template <class T>
bejaro<T>* tomb<T>::letrehozbejaro()
{
   return new bejaro<T>(this);
}
```

A begin és end tagfüggvények

```
template <class T>
typename tomb<T>::iterator tomb<T>::begin()
{
  return iterator(t);
}

template <class T>
typename tomb<T>::iterator tomb<T>::end()
{
  return iterator(t + n);
}
```

Az iterator belső osztály konstruktora

```
template <class T>
tomb<T>::iterator::iterator(T* a)
{
  aktualis = a;
}
```

A ++ operátor

```
template <class T>
typename tomb<T>::iterator& tomb<T>::iterator::operator++()
{
  aktualis++;
  return *this;
}

template <class T>
typename tomb<T>::iterator tomb<T>::iterator::operator++(int)
{
  iterator temp(*this);
  aktualis++;
  return temp;
}
```

További operátorok

```
template <class T>
typename tomb<T>::iterator& tomb<T>::iterator::operator--()
{
  aktualis--;
  return *this;
}

template <class T>
bool tomb<T>::iterator::operator==(const iterator& a)
{
  return aktualis == a.aktualis;
}
```

További operátorok

```
template <class T>
bool tomb<T>::iterator::operator!=(const iterator& a)
{
  return aktualis != a.aktualis;
}

template <class T>
T& tomb<T>::iterator::operator*()
{
  return *aktualis;
}
```

További operátorok

```
template<class T>
int tomb<T>::iterator::operator-(const iterator& a)
{
  return aktualis - a.aktualis;
}

template<class T>
bool tomb<T>::iterator::operator<(const iterator& a)
{
  return aktualis < a.aktualis;
}
```

A fő függvény

```
int main()
{
  double t[] = { 1.1, 2.2, 3.3, 4.4, 5.5, 6.6, 7.7, 8.8 };
  tomb<double> v(t, 8);
  v.kiir();
  v.hozzaad(9.9);
  v.hozzaad(10.10);
  v.hozzaad(11.11);
  v.kiir();
```

A bejárás első változata

```
bejaro<double>* b;
for (b = v.letrehozbejaro(); !b->vege(); (*b)++)
{
  cout << **b << " ";
}
cout << endl;
delete b;
```

További bejárások

```
bejaro<double>& b2 = *v.letrehozbejaro();
for (; !b2.vege(); b2++) {
  cout << *b2 << ' ';
}
cout << endl;
delete& b2;

bejaro<double>  b3(&v);
for (; !b3.vege(); b3++) {
  cout << *b3 << ' ';
}
cout << endl;
```

Bejárás iterátorral

```
for (tomb<double>::iterator i = v.begin(); i != v.end(); i++)
{
  cout << *i << " ";
}
cout << endl;
```

Keresés

```
double a;
cout << "a= ";
cin >> a;
tomb<double>::iterator it = find(v.begin(), v.end(), a);
if (it == v.end())
  cout << "Nem talalhato\n";
else
  cout << "Az elem: " << *it
       << " pozicio: " << it - v.begin() << endl;
```

Tükrözés

```
  reverse(v.begin(), v.end());

  for (auto i = v.begin(); i != v.end(); i++) {
    cout << *i << " ";
  }
  cout << endl;

  for (double elem : v)
    cout << elem << " ";
  cout << endl;
}
```

Lehetséges kimenet

1.1 2.2 3.3 4.4 5.5 6.6 7.7 8.8 Maximum: 10
1.1 2.2 3.3 4.4 5.5 6.6 7.7 8.8 9.9 10.1 11.11 Maximum: 12
1.1 2.2 3.3 4.4 5.5 6.6 7.7 8.8 9.9 10.1 11.11
1.1 2.2 3.3 4.4 5.5 6.6 7.7 8.8 9.9 10.1 11.11
1.1 2.2 3.3 4.4 5.5 6.6 7.7 8.8 9.9 10.1 11.11
1.1 2.2 3.3 4.4 5.5 6.6 7.7 8.8 9.9 10.1 11.11
a= **4.4**
Az elem: 4.4 pozicio: 3
11.11 10.1 9.9 8.8 7.7 6.6 5.5 4.4 3.3 2.2 1.1
11.11 10.1 9.9 8.8 7.7 6.6 5.5 4.4 3.3 2.2 1.1