Bevezetés a programozásba 2

9. Előadás: template folytatás, STL

ISMÉTLÉS

template

- Típussal paraméterezett függvény vagy osztály
- pl. vector<T>

```
template <typename T>
class TC {
   T mező;
   T fv(T a);
};


  ...
TC<int> tci;
TC<string> tcs;
```

ISMÉTLÉS

template példa

```
template <typename T>
class Tomb {
   T *_m; int _s;
public:
   Tomb(int s) : _s(s) { _m=new T[_s];}
   ~Tomb() {delete[] _m;}
   T operator[](int i) const {return _m[i];}
};
...
Tomb<int> t(10); int k = t[1];
Tomb<string> t2(10); string s = t2[1];
```

ISMÉTLÉS

template függvény

```
template <typename T>
T maxt(const T& a, const T& b)
{
  return a > b ? a : b;
}
...
char k = maxt('a','b');
int i = maxt(3,4);
```

ISMÉTLÉS

std::function

- még egy Gomb implementációs lehetőség, tevékenységet std::function-ben tárolni
- A szignatúra így nem beégetett, hanem fordítási idejű paraméter

```
class StdFuncButton: .. {
    std::function<void()> f;
public:
    virtual void action() { f(); }
};
void fv() {ezt kellene meghívni}
StdFuncButton <void()> *b = new StdFuncButton <void()>
(..., fv);
```

a funktor

```
struct Funktor {
...
  operator()(int a) { ... }
};

Funktor fkt;
fkt(5);
```

a funktor

```
struct Funktor {
   int _x;
   Funktor(int x) : _x(x)
   operator()(int a) { ...x... }
};

Funktor fkt(10);
fkt(5);
```

a funktor

- Olyan osztály/objektum, amelyik rendelkezik operator() tagfüggvénnyel
- Callable, zárójelet mögé írva olyan, mintha függvényt hívnánk meg
- Template programozásnál szövegszerű helyettesítésnél ez kihasználható

funktor átadása template paraméterként

```
struct Funktor {
   int _x;
   Funktor(int x) : _x(x)
   operator()(int a) { ...x... }
};

Funktor fkt(10);
fkt(5);
```

```
template <typename Fun>
void fv(Fun f)
{
   ...
   f( ... )
   ...
}

fv(Funktor(10))
```

STL

- Standard Template Library
- vector már ismerős
- van benne még sok konténer
  - vector, list, map, és set a jellegzetes példák
- és sok algoritmus.

vector

- az elemek a memóriában egymás mellett állnak
  - ezért hatékony az elem közvetlen címzése
  - de nem hatékony középre beszúrni vagy törölni

list

- Minden eleme tudja hogy hol van a következő
- „láncolt lista”
- emiatt a beszúrás/törlés hatékony, az indexelés viszont nem

Iterátorok

- A konténerek elemkezelésére való
- a mutatók szintaxisa ihlette

```
vector<int> v;
...
for (vector<int>::iterator it=
v.begin(); it!=v.end(); ++it) {
  cout << *it << ” ”;
}
```

Iterátorok

- A konténerek elemkezelésére való
- a mutatók szintaxisa ihlette

```
list<int> v;
...
for (list<int>::iterator it=
v.begin(); it!=v.end(); ++it) {
   cout << *it << ” ”;
}
```

Iterátorok

```
vector<int> v;
...
sort(v.begin(), v.end())
```

Iterátorok

```
bool rendfv(int a, int b);


vector<int> v;
...
sort(v.begin(), v.end(), rendfv)
```

STL algoritmusok

- A legtöbb STL algoritmus feltételez bizonyos meglevő műveletet
  - tipikus példa az operator<
- Saját típusokhoz ezeket meg kell valósítani

STL map

- Asszociatív adatszerkezet

```
map<string, int> m;
m[”jan”]=31;
m[”feb”]=28;

map<string, int>::iterator it =
  m.find(”jan”);
if (it!=m.end()) {
  ...//megvan az elem
}
```

pair

- Az STL map kulcs-érték párokból áll
- A map minden tétele egy pair<T1, T2> aminek van first és second mezője

```
map<string, int> m;

for (map<string, int>::iterator
it=m.begin(); it!=m.end(); ++it) {

  cout << it->first << it->second;
}
```

STL map

- A map belsejében egy rendezőfa van a kulcsokból
- beszúrás, törlés, elem keresése mind logaritmikus idejű
- -> STL hashmap: akár konstans idejű műveletek is lehetségesek

STL algoritmusok

- Minden, amit BevProg1-ből tételként, algoritmusként tanultunk, megtalálható
- Paraméterként konténereket fogadnak, elemeket iterátorral hivatkoznak
- find, binary_search, merge, max_element, accumulate, ...