5. lekce

Knihovna STL, vektor a seznam

Knihovna STL

- STL – **Standard Template Library** (standardní knihovna šablon)

- obsahuje **datové kontejnery (šablony) –** patří sem **vektor, seznam, mapa, množina, pole, zásobník, fronta, ….**

- datový kontejner je speciální objekt, do kterého můžeme vkládat další objekty, ale i jednoduché proměnné

- usnadňuje práci s alokací paměti

- při použití STL je nutné **nainkludovat konkrétní datový kontejner**, který chceme použít (standardní datové kontejnery jsou součástí standardu – nic nemusíme programovat)

- STL používá **iterátory** (iterátor je speciální proměnná, pomocí které je možné datový kontejner procházet – ukazatel zapouzdřený *(definovaný ve struktuře objektu)* do objektu)
- nejjednodušší je použít pro iterátor datový typ auto

Příklad procházení datového kontejneru – platí pro všechny typy datových kontejnerů, nejen pro vector

vector <int> vektor ={1,5,2,4};
// definice proměnné vektor: vector – typ datového kontejneru, int – obsahuje proměnné typu integer, vektor je název proměnné, vektor jsme naplnili prvky(integery) 1,5,2,4

a) bez iterátoru

– pracuje s jednotlivými integery kontejneru

```
for (const auto & i : vektor)
{
    cout<<i<<endl;
}
```

for (const auto & i : vektor) – projde všechny prvky datového kontejneru vektor
const – prvky kontejneru nebudeme měnit
auto - kompilátor si dosadí sám datový typ proměnné
& - bereme referenci (adresu v paměti) na prvky z datového kontejneru vektor, v případě datového kontejneru, který bude obsahovat objekty, by kopírování objektů do i trvalo dlouho – práce s adresami v paměti je rychlejší
vektor – kontejner, který procházíme cyklem
i je řídící proměnná cyklu - integer z vectoru vektor – v příkladu bude postupně 1, 5, 2, 4

b) s iterátorem

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

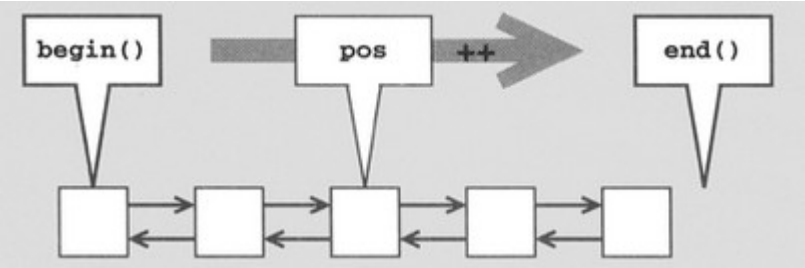
for (auto i = vektor.begin(); i != vektor.end(); i++)

auto - kompilátor si dosadí sám datový typ proměnné i

i – iterátor

vektor.begin() **-** ukazuje na první prvek vektoru,

vektor.end() - ukazuje za poslední prvek, proto je ***i != vektor.end()***



Pomocí iterátoru je možné zpracovávat vector oběma směry ++/--

cout<<*i<<endl; - vypisujeme všechny integery z vectoru, protože je i iterátor, který představuje adresy integerů ve vectoru, musíme použít *, která nás odkáže na hodnotu pod adresou každého integeru.

Vektor

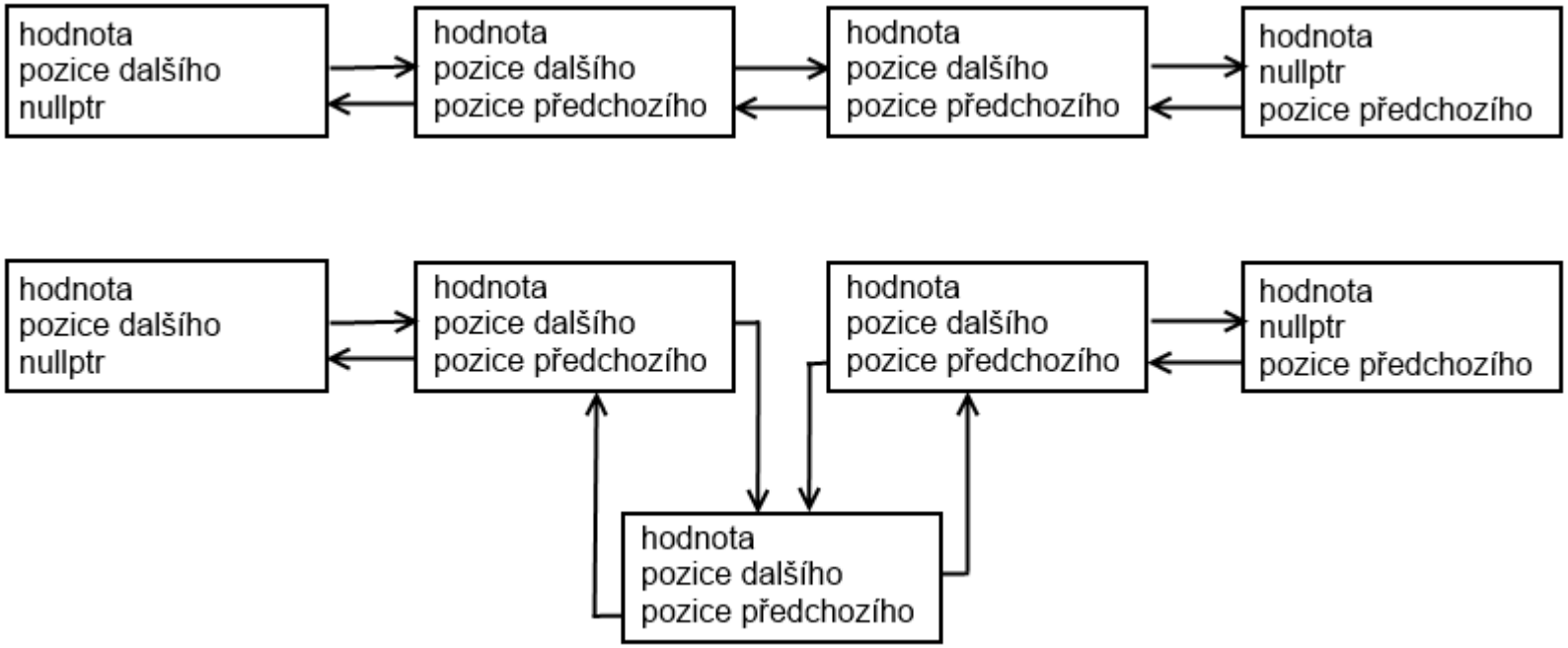
- zapouzdřené klasické pole (pole ve vektoru)
- nemusíme řešit alokaci, realokaci a uvolňování (řeší sama knihovna **#include <vector>**)

Funkce vektoru:

- ***begin, end*** - vrací iterátory (zapouzdřený ukazatel) na začátek a konec vektoru (za poslední prvek)
- ***front, back*** - vrací hodnoty na začátku a na konci vektoru
- ***[]*** - vrací prvek na dané pozici *(v závorkách definujeme index)*
- ***at*** - vrací prvek na dané pozici, *probíhá kontrola hranic (při indexu mimo rozsah pole aplikace nespadne, ale vyhodí výjimku, kterou musíme zachytit pomocí try a catch)*
- ***size*** - vrací velikost *(počet prvků)* vektoru *(pole ve vektoru)*
- ***empty*** - vrací true, když je vektor prázdný, jinak false
- ***reverse*** - převrátí pořadí prvků
- ***clear*** - vymaže všechny prvky vektoru
- ***insert (iterátor pozice, hodnota)*** - vloží prvek (hodnota) na danou pozici (iterátor pozice), všechny prvky od iterátoru pozice posune o jednu pozici doprava
- ***push_back*** - vloží prvek na konce vektoru

List

- zapouzdřený obousměrně zřetězený spojový seznam
- nemusíme řešit alokaci, realokaci a uvolňování (řeší sama knihovna **#include <list>**)



Vkládání prvků do seznamu

Funkce seznamu *(vše jako u vektoru, kromě [] a at, navíc push_front)*:

- ***begin, end*** - vrací iterátory (zapouzdřený ukazatel) na začátek a konec seznamu
- ***front, back*** - vrací hodnoty na začátku a na konci seznamu
- ***size*** - vrací velikost (počet prvků) seznamu
- ***empty*** - vrací ***true***, když je seznam prázdný, jinak ***false***
- ***reverse*** - převrátí pořadí prvků
- ***clear*** - vymaže všechny prvky seznamu
- ***insert (iterátor pozice, hodnota)*** - vloží prvek (hodnota) na danou pozici (iterátor pozice), všechny prvky od iterátoru pozice posune o jednu pozici doprava
- ***push_back*** - vloží prvek na konce seznamu
- ***push_front*** - vloží prvek na začátek listu
- ***sort*** - vzestupné řazení prvků seznamu, používá operátor < (pokud jsou prvky seznamu objekty, musí být definovaný operátor <)

Porovnání vektoru a listu

Vektor	List
Vkládání prvků do vektoru je lineární *(vloží se prvek a všechny prvky za ním se posunují o pozici doprava – pokud nevkládáme na poslední pozici příkazem push_back)*	Vkládání prvků do seznamu je konstantní *(zabere pouze jednu operaci, po vložení prvku se nemění pozice prvků za vloženým prvkem)*
Vektor zabírá méně místa v paměti než list	List zabírá více místa v paměti než vektor *(navíc má ukazatele na další a předchozí prvek)*
Ukazatele do vektoru ztrácí platnost při realokaci	Ukazatele do listu jsou platné po celou dobu existence listu *(není zde realokace)*
U vektoru můžeme přistupovat k libovolnému prvku *(pomocí indexu)*	List musíme procházet pomocí iterátoru
Vektor má horší využití operační paměti *(Po realokaci je vytvořeno nové pole v novém místě RAM, prvky originálního pole jsou přeneseny do nového pole a původní pole je z paměti odstraněno. Místo v RAM, které je nyní volné je omezeno pro další využití, protože má omezenou velikost – RAM ho nemůže použít pro paměťové objekty větší)*	Není problém se zvětšováním listu z hlediska RAM, protože se paměť alokuje pro každý prvek seznamu odděleně.