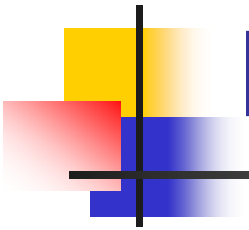
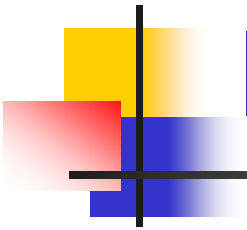


Řetězce, pole a STL

- V C++ je výhodné pro práci s řetězci použít třídu **string**, funkce C jsou stále k dispozici **cstring**,
- ukazatele a pole lze stále používat stejně, jako v C,
- použití iterátorů a dalších konstrukcí STL v příkladech



Množina

Typ prvku určuje šablona

- insert() vloží prvek,
- find() vrátí iterátor odkazující na prvek, nebo
- end() což je iterátor dosažení konce množiny (prvek nenalezen).

Množina čísel int

stl-set-int.cpp

```
#include <set>
#include <iostream>
using namespace std;

int main() {
 set<int> mnozina;
 set<int>::iterator iter;

 mnozina.insert(2);
 mnozina.insert(5);
 mnozina.insert(7);
```

```
for (int i = 0; i < 10; i++) {
      iter = mnozina.find(i);
      if (iter != mnozina.end())
          cout << i << " je prvek
    mnoziny" << endl;
} // for()
```

Množina znaků

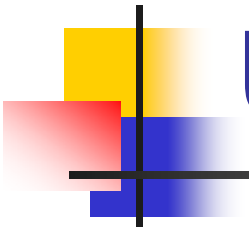
stl-set-chr.cpp

Cílem je vyloučit samohlásky z řetězce

```
#include <set>
#include <string>
#include <iostream>

int main() {
 set<char> samohlasky;
 set<char>::iterator iter;

 samohlasky.insert('a');
 …
 samohlasky.insert(' ');

 string text = "Dnes je sobota.";
```

Množina znaků pokr. 2

```
for (int i = 0; i < text.size(); i++) {
    iter = samohlasky.find(text.at(i));
    if (iter == samohlasky.end())
       cout << text.at(i);
 } // for()
Dnes je sobota.
Dnsjsbt.
```



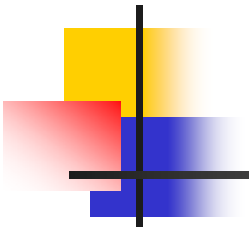
Ukazatele v C++

- Převzato z C,
- velmi efektivní práce v paměti,
- odpovídá von Neumanově architektuře počítače,
- v STL nahrazeno iterátory.

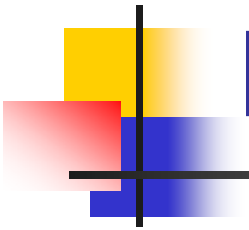
typ *uk;

uk – ukazatel,

*uk – dereference ukazatele.



```
int x, y, *px, *p2x;
px = &x;          /* px nyní ukazuje na x */
*px = 5;          /* totéž co x = 5;      */
y = *px + 1;   /*          y = x + 1;  */
*px += 1;      /*          x += 1;     */
(*px)++;   /* x++; závorky jsou nutné*/
p2x=px;
   /*p2x ukazuje na totéž co px, tj. na x*/
*p2x = *p2x + y;  /*   x = x + y;  */
```



Pole metodami jádra C++

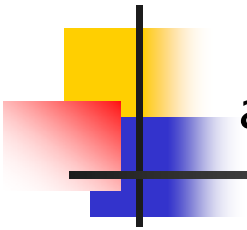
`typ jméno[rozsah];`

- je jednorozměrné,
- kolekce pevně určeného počtu proměnných stejného typu,
- přístup: identifikátor pole a index,
- obsazuje spojitou oblast operační paměti,
- identifikátor pole je konstantní ukazatel na první prvek pole.

Příklad práce s polem

arry-demo.cpp 1

```
const int N = 5;
double p[N] = {1.2, 3.4, -1.2, 123.45e67, -4.e-5};
              //p[0] p[1] p[2]   p[3]        p[4]
cout << "index\thodnota" << endl;
for (int index = 0; index < N; index++) {
   cout << index << '\t' << p[index] << endl;
}
```

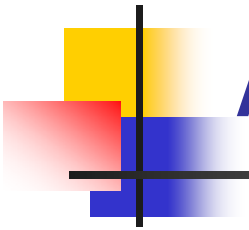


arry-demo.cpp 2

konformní pole (neurčený počet prvků):

```
int c[] = {3, 4, 5};
int pocet = sizeof(c)/sizeof(int);
cout << "pocet prvku pole c je:" << pocet <<
  endl;
```

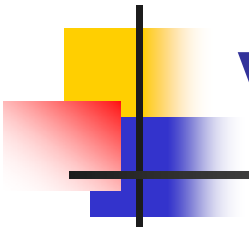
Aritmetika ukazatelů



- ukazatel je adresa v paměti,
- aritmetika probíhá s ohledem na datový typ, s nímž je ukazatel spojen,
- odečtění dvou ukazatelů je počet prvků „pole“ meni nimi,
- operátor ++ a -- je lépe číst jako následující a předcházející prvek.

arry-demo.cpp 3

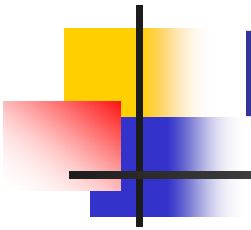
```
int *uk_int = (c + pocet) - 1; // začíná se od 0
 do {
   cout << *uk_int << '\t';
 } while (uk_int-- != c);
 cout << endl << "A ted bude zmena!" << endl;

 uk_int = c; // teď by stačilo jen uk_int++;
 *(uk_int + 1) = -4321;
 for(; uk_int < c+pocet; uk_int++) {
   cout << *uk_int << '\t';
  }
```



výstup:

```
index   hodnota
0       1.2
1       3.4
2       -1.2
3       1.2345e+069
4       -4e-005
pocet prvku pole c je:3
5       4       3
A ted bude zmena!
3       -4321   5
konec
```

Pole a vector – bezpečnost

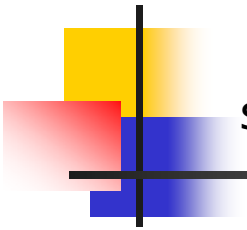


pole je tradiční přístup jádra C/C++

- nekontroluje meze,
- přístup přes []

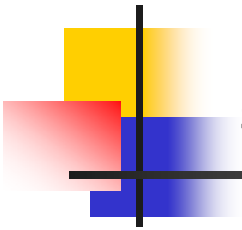
vector je součástí STL

- schopen vyvolat vyjímku (out_of_range ze stdexcept),
- přístup at() ale i []



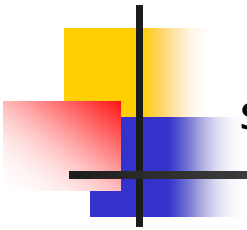

```
#include <vector>
#include <iostream>
#include <exception>
#include <stdexcept>

using namespace std;

int main() {
  const int N = 5;
  vector<int> v(N);
  int klas[N];
  int i;
```

```
for (i = 0; i < N; i++)
 {
     v.at(i) = 2 * i;
     klas[i] = 2 * i;
} // for()
```



stl-vec-sec.cpp 2/2

```
cout << "vector bezpecne pres .at()" <<  endl;
 try {
      for (i = 0; i < 2 * N; i++) //chyba v mezi
           cout << v.at(i) << ' ';
 }
 catch (out_of_range e) {
    cout << "Zachycena vyjimka!"  << endl;
    cout << "Jeji popis je: " << e.what() << endl;
 }
 cout << endl << "pole klasicky pomoci []" <<  endl;
 for (i = 0; i < 2 * N; i++)
     cout << klas[i] << ' ';
```



stl-vec-sec.cpp výstup

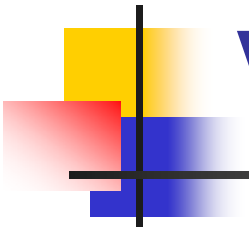
vector bezpecne pres .at()

0 2 4 6 8 Zachycena vyjimka!

Jeji popis je: vector [] access out of range

pole klasicky pomoci []

0 2 4 6 8 2293528 8 2293728 4012536 4012556



Vícerozměrná pole

typ jmeno[dim1][dim2]

- pole je v C++ *jednorozměrné*,
- řešení -> prvkem pole je pole,
- přístup přes indexy, „řádek“, „sloupec“

Ukazatele, konstanty a pole

- Klasický C přístup pracuje s poli jako s ukazateli (jen ID pole je konstantní uk.).
- Pozor! Konstantní je ukazatel nebo to, na co ukazuje:

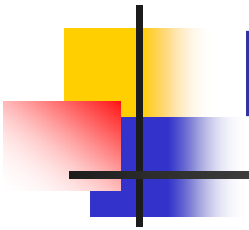
```
const int ci = 7;
const int *pci; // neinic.
const int * const cpc = &ci;
```

Kopírování pole

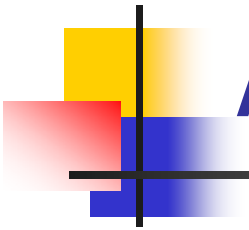
- zdroj, cíl, počet prvků,
- buď přes jméno pole a [],
- nebo pomocí ukazatelů na bázový typ,
- př. soubor `arry-cpy.cpp`:

```
void copy_array2(const int *a, int *b, int n) {
//a-vstupni pole, b-vystupni pole, n-pocet prvku
 while (n-- > 0)
   *b++ = *a++;
} // void copy_array2()
```

Řetězec v C pojetí



- pole znaků ukončené znakem s kódem 0 – zarážka, příklad kopírování `str-cpy.cpp`,
- rodina funkcí `str...` z hlavičky `cstring`,
- pozor, řetězec je chápán jako ukazatel ne jako celý text – pro komfort je lepší C++ třída `string`.



Argumenty příkazového řádku

```
int main(int argc, char *argv[]);
```

- `argc` je počet argumentů,
- `argv` jsou hodnoty (C-řetězce).

Příklad `cmd-ln.cpp`.