# Strings, arrays (and pointers)

- Use class **string** in C++ for strings manipulation;
  C functions are still available through **cstring** library;
- Pointers and arrays stay the same as in C,
- there are iterators and othe constructions in STL (Standard Template Library)

# Strings in C++

```
string str = “Hello world!”;
```

- .length() – length of the string

  ```
      cout << str.length();
  ```

- .at(i) – access to the character at position i, safed (exception could be thrown)

  ```
      cout << str.at(i);
  ```

- [i] – access to character at position i

  ```
      cout << str[i];
  ```

# String, array and vector
(safety)

Array is traditional core C/C++ approach;

- Bounds are not checked,
- Acces using []

string and vector are C++ (STL)

- Exception could be thrown (out_of_range from stdexcept),
- Access using both: at() and []

# stl-vec-sec.cpp 1/2

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

using namespace std;

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

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

# stl-vec-sec.cpp 2/2

```
cout << "vector - .at(), safe access" <<  endl;
 try {
      for (i = 0; i < 2 * N; i++) //index out of range
            cout << v.at(i) << ' ';
 }
 catch (out_of_range e) {
    cout << “Exception catched!"  << endl;
    cout << “Its description:" << e.what() << endl;
 }
 cout << endl << “array – [], traditional approach" <<  endl;
 for (i = 0; i < 2 * N; i++)
     cout << trad[i] << ' ';
```

# stl-vec-sec.cpp výstup

```
vector - .at(), safe access
0 2 4 6 8 Exception catched!
Its description: vector [] access out of range

array – [], traditional approach
0 2 4 6 8 2293528 8 2293728 4012536 4012556
```

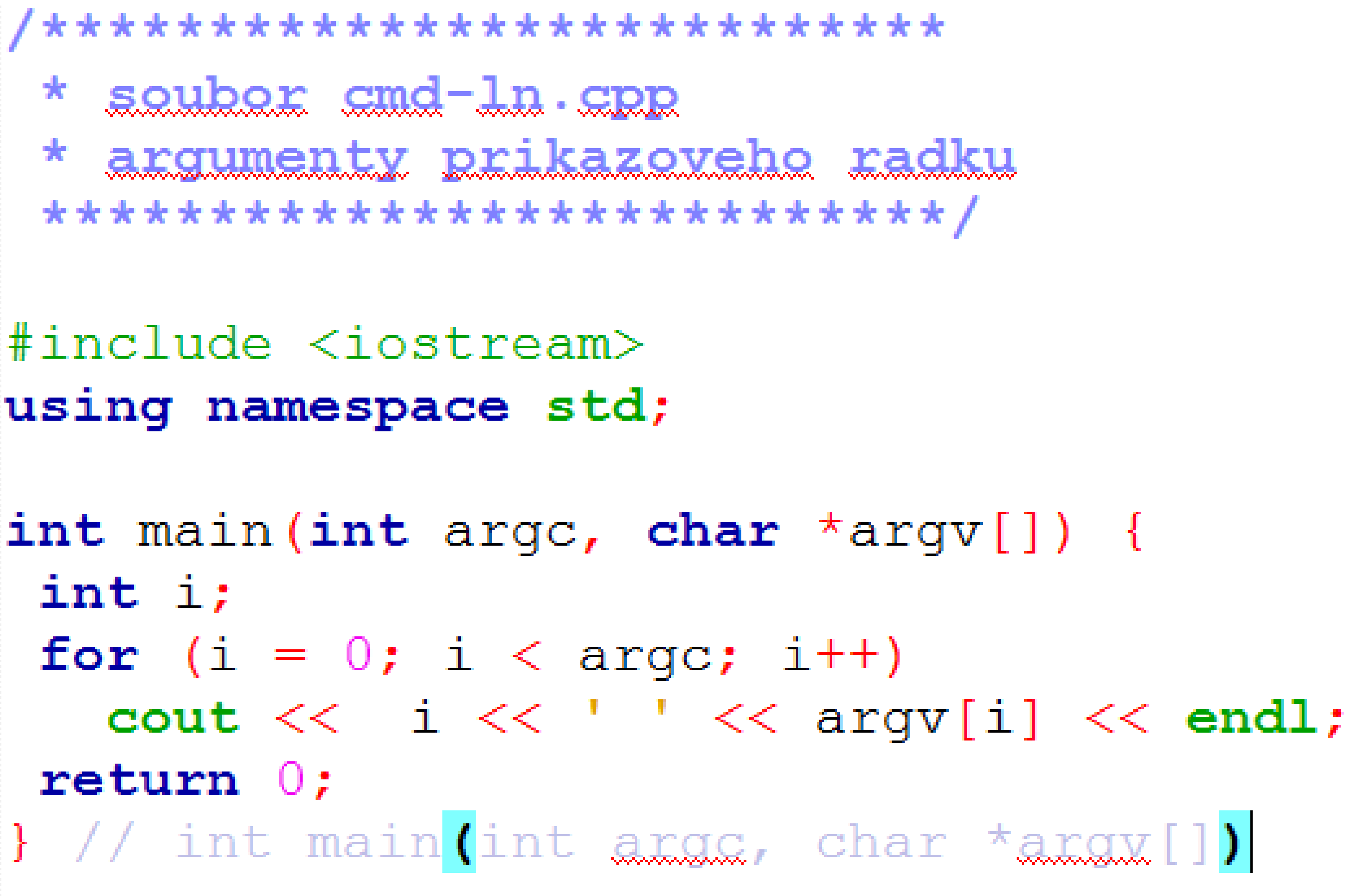
# Command line parameters (overview)

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

- `argc` means argument counter;
- `argv` means argument values (C-strings).

Example: `cmd-ln.cpp`

# Command line parameters (example)



```
/****************************
 * soubor cmd-ln.cpp
 * argumenty prikazoveho radku
 ****************************/

#include <iostream>
using namespace std;

int main(int argc, char *argv[]) {
 int i;
 for (i = 0; i < argc; i++)
   cout <<  i << ' ' << argv[i] << endl;
 return 0;
} // int main(int argc, char *argv[])
```

# Set
## (another STL container)

Type of element is defined at creation

- insert() inserts an element,
- find() iterator is returned, otherwise
- end() is returned, when end of the set is reached (element not found).

# Set of integers example stl-set-int.cpp

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

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

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

```
for (int i = 0; i < 10; i++) {
    iter = st.find(i);
    if (iter != st.end())
       cout << i << " is
  element" << endl;
} // for()
```

# Set of characters example stl-set-chr.cpp

Task: avoid vovels contained in a string (display non-vovels).

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

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

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

 string text = “There is Monday Today.";
```

# Set of characters, cont. 2

```
for (int i = 0; i < text.length(); i++) {
    iter = vovels.find(text.at(i));
    if (iter == vovels.end())
        cout << text.at(i);
 } // for()
```

```
There is Monday Today.
ThrsMndTd.
```

# Pointers v C++

- The same as in C,
- Very efficient memory manipulations,
- Follows von Neumann computer architecture,
- in STL replaced with iterator approach.

type *ptr;

ptr – pointer,

*ptr – pointer’s dereference.

# Few lines of code

```
int x, y, *px, *p2x;
px = &x;          /* px points now to x */
*px = 5;          /* the same as x = 5;     */
y = *px + 1;   /*          y = x + 1;  */
*px += 1;      /*          x += 1;     */
(*px)++;   /* x++; () are necessary*/
p2x=px;
   /*p2x and px points to the same thing */
*p2x = *p2x + y;  /*   x = x + y;  */
```

# Array C/C++ kernel approach

```
type name[size];
```

- One imension,
- Homogenous data type (of items), constant size,
- Access method: array-identifier + [] + index,
- Continuous memory area is used,
- array-identifier is a constant pointer to the first array item (element).

# Array example

`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\tvalue" << endl;
 for (int index = 0; index < N; index++) {
    cout << index << '\t' << p[index] << endl;
}
```

# arry-demo.cpp 2

Open array (array size not defined):

```
int c[] = {3, 4, 5};
int count = sizeof(c)/sizeof(int);
cout << “there is: " << count << “ items in array." << endl;
```

# Pointers arithmetic

- Pointer is the address in the memory,
- Data type of a pointer is used when pointer arithmetic is evaluated,
- Two pointers subtraction means number of items between them (their distance),
- Read operators ++ and – as next item or previous item respectively.

## arry-demo.cpp 3

```
int *ptr_int = (c + count) - 1; // starts at 0
 do {
   cout << *ptr_int << '\t';
 } while (ptr_int-- != c);
 cout << endl << “Some change now." << endl;

 ptr_int = c; // ptr_int++ could be enough;
 *(ptr_int + 1) = -4321;
 for(; ptr_int < c+count; ptr_int++) {
   cout << *ptr_int << '\t';
  }
```

# výstup:

```
index   value
0       1.2
1       3.4
2       -1.2
3       1.2345e+069
4       -4e-005
there is: 3 items in array.
5       4       3
Some change now.
3       -4321   5
end
```

# Multidimensional array

type name[dim1][dim2]

- There are one-dimensional arrays in C/C++ *only*,
- solution -> an array is the element of the array (array of arrays),
- Use indexes “row” and “column” for access to items.

# Pointers, constants and arrays

- Traditional C deals with array as with pointers (array-ID is a constant pointer to the array).
- Warning! Decide carefully whot is a constant: pointer or value pointed by the pointer? See:

```
const int ci = 7;
const int *pci; // not inicialized.
const int * const cpc = &ci;
```

# Array copy

- source, destination, number of elements,
- Using either array-name + [],
- Or pointers to a type of array-item (array base type),
- Example: `arry-cpy.cpp`:

```
void copy_array2(const int *a, int *b, int n) {
//a-source, b-destination, n-# of elements
 while (n-- > 0)
   *b++ = *a++;
} // void copy_array2()
```

# Strings
## traditional C approach

- Array of characters, finished by a character with code 0 (end of string), example: `str-cpy.cpp`,
- Family of functions `str...` Header file (library) **cstring**,
- Warning & recomendation>: use C++ `string class` when you are not familiar wit pointers.