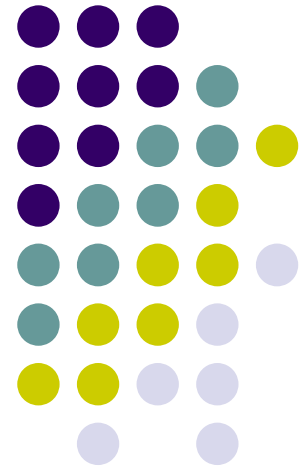


# Średniozaawansowane programowanie w C++

---

Wykład #6  
25 listopada 2020 r.



# Plan

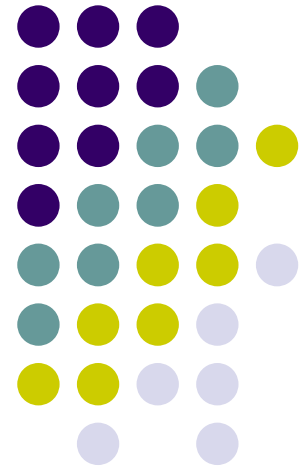


1. Mechanizm wyjątków
2. `std::string`
3. `boost::lexical_cast`
4. `std::regex` (wyrażenia regularne)
5. Biblioteka `ncurses`

# Mechanizm wyjątków

---

Każdy z nas  
jest zupełnie wyjątkowy



# Wyjątki (1)



```
#include <exception>
```

```
class BładParsowania : std::exception    // klasa błędu – może być pusta
{
    public:
        int linia;
        BładParsowania (int ln) : linia (ln) {}
};
```

```
std::auto_ptr <std::vector <Procedura*> > parsuj (std::istream &is)
{
    char bufor [256];
    std::unique_ptr <std::vector <Procedura*> > program;
    int linia = 1;
    while (is.good ())
    {
        is.getline (bufor);
        if (/* nie potrafię sparsować bufora */)
            throw BładParsowania (linia); // rzucamy wyjątek!
        linia++;
        // (...) dorzucamy procedurę do programu
    }
    return program;
}
```

# Wyjątki (2)

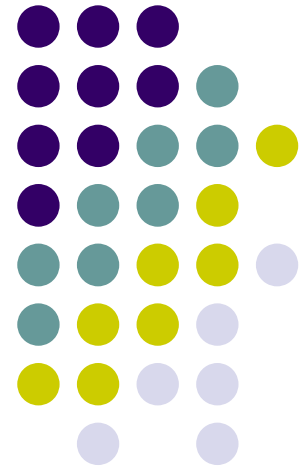


```
int main ()
{
    std::unique_ptr <std::vector <Procedura*> > procedura;

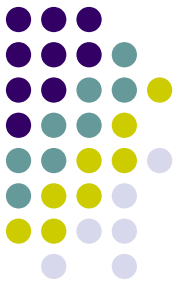
    try { // blok objęty mechanizmem wyjątków
        procedura = parsuj (std::cin); // może rzucić wyjątek
    }
    catch (BładParsowania &bp) // łapiemy wyjątek danego typu
    {
        std::cerr << „Bład składni w linii ” << bp.linia << std::endl;
        return 1;
    }
    catch (std::exception &e) // łapie wyjątki dziedziczące po exception
    {
        std::cerr << „Wystąpił nieznany bład” << std::endl;
        return 2;
    }
    return 0;
}
```

# std::string

Dobre stringi nie są złe :)



# std::string



```
#include <string>
```

```
std::string nazwa_pliku = wczytaj_nazwe ();
if (nazwa_pliku == „passwd”) {
    std::cerr << „Nie możesz wybrać tego pliku!” << std::endl;
    return 1;
}

if (nazwa_pliku.find („xxx”) != string::npos)
    std::cerr << „Próbujesz otworzyć brzydkie rzeczy!” << std::end;

std::ifstream plik (nazwa_pliku.c_str ());
wczytaj_z_pliku (plik);

nazwa_pliku.clear ();    // czyści stringa

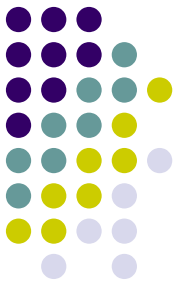
if (!nazwa_pliku.empty ())
    std::cerr << „Czyszczenie stringa się nie powiodło!” << std::endl;

std::string inny_napis = „Naprawde inny napis”;
inny_napis += „ w stylu C!”
```

Więcej fantastycznych metod i operatorów klasy string:

[www.cplusplus.com/reference/string/string](http://www.cplusplus.com/reference/string/string)

# std::to\_string



```
#include <string>
```

```
string to_string (int val);  
string to_string (long val);  
string to_string (long long val);  
string to_string (unsigned val);  
string to_string (unsigned long val);  
string to_string (unsigned long long val);  
string to_string (float val);  
string to_string (double val);  
string to_string (long double val);
```

// Przykład

```
std::string napis = "333";           // string zawierający liczbę  
int liczba = 44;                     // int zawierający liczbę  
  
liczba = std::stoi (napis); // string → int  
napis = std::to_string (liczba); // int → string
```

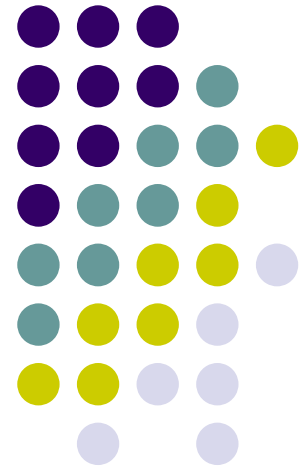
[http://www.cplusplus.com/reference/string/to\\_string/](http://www.cplusplus.com/reference/string/to_string/)

<http://www.cplusplus.com/reference/string/>



# boost::lexical\_cast

Konwersje pomiędzy typami  
przechowującymi logicznie  
tożsame wielkości



# boost::lexical\_cast



```
#include <boost/lexical_cast.hpp>
```

```
std::string napis = "333";      // string zawierający liczbę  
int liczba = 44;                // int zawierający liczbę
```

```
liczba = boost::lexical_cast <int> (napis); // string → int  
napis = boost::lexical_cast <std::string> (liczba); // int → string
```

```
int main (int argc, char *arg [])  
{  
    try {  
        // Konwersja char* → short  
        short ile = boost::lexical_cast <short> (arg [1]);  
    }  
    catch (boost::bad_lexical_cast &)  
    {  
        // Nie można dokonać konwersji  
    }  
    return 0;  
}
```

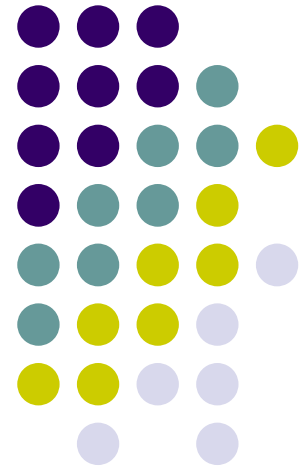
Więcej fantastycznych rzeczy o lexical\_cast:

[www.boost.org/doc/libs/release/libs/conversion/lexical\\_cast.htm](http://www.boost.org/doc/libs/release/libs/conversion/lexical_cast.htm)

# Wyrażenia regularne

---

boost::regex  
std::regex



# Wyrażenia regularne



Znak	Znaczenie	Przykład
<code>^ &amp;</code>	Początek i koniec linii	
<code>.</code>	Dowolny (jeden) znak	<code>ma.a</code> mata mama mana
<code>[abc] [a-z]</code>	Zbiór / zakres znaków	<code>[mt]a[mt]a</code> mata mama tata tama
<code>[^0-9]</code>	Dopełnienie zbioru	<code>buziaczek[^0-9]</code> buziaczekx
<code>\d</code>	Cyfra	<code>buziaczek\d\d</code> buziaczek13
<code>\s</code>	Biały znak	
<code>⊛*</code>	Dowolna ilość ( $\geq 0$ )	<code>a*ron</code> ron aron aaron
<code>⊛+</code>	Co najmniej jeden	<code>a+ron</code> aron aaron
<code>⊛?</code>	Opcjonalnie (0 lub 1)	<code>k?los</code> klos los
<code>⊛{n, m}</code>	Od n do m wystąpień	<code>ha{1,3}lo</code> halo haalo haaalo
<code>(⊛)</code>	Grupa	<code>(ba)+</code> ba baba bababa
<code>⊛ ⊛</code>	Alternatywa	<code>ta((ma) (to))</code> tama tato

# boost::regex\_match



```
#include <boost/regex.hpp>
// C++11
#include <regex>

boost::regex data_urodzin („\\d{1,2}\\.\d{1,2}\\.(\\d{2}|\\d{4})");

std::string data;
std::cout << „Podaj date urodzin w formacie dd.mm.rrr: ";
std::cin >> data;

if (! boost::regex_match (data, data_urodzin))
    std::cout << „Zły format!" << std::endl;
```

# boost::regex\_search



```
#include <boost/regex.hpp>
#include <boost/lexical_cast.hpp>

using namespace std;

string tekst („Koszykowa 75/306");
boost::regex wzorzec („([a-zA-Z]+)\\s(\\d+)/ (\\d+)");
boost::match_results <string::const_iterator> dop;

boost::regex_search (tekst.begin(), tekst.end(), dop, wzorzec);

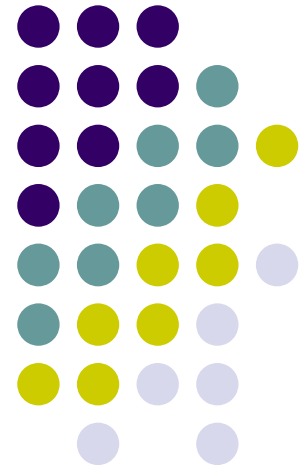
if (dop.size () < 4) return 0; // złe dopasowanie

string ulica (dop[1].first, dop[1].second);
int nr = boost::lexical_cast <int> (string (dop[2].first, dop[2].second));
int pok = boost::lexical_cast <int> (string (dop[3].first, dop[3].second));
```

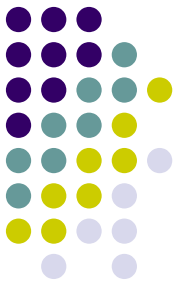
# Biblioteka ncurses

---

„Grafika” pod konsolą



# Inicjalizacja wirtualnego ekranu



```
#include <ncurses.h>

// Inicjuje bibliotekę
initscr ();

// Wczytuje znaki z bufora wejściowego bez oczekiwania na znak końca linii
cbreak ();

// Nie wyświetla wciśniętego klawisza na ekranie
noecho ();

// Nie czeka na wciśnięcie znaku przy wywołaniu getch()
nodelay ();

// Uruchamia kolory
start_color();

// (...) miziamy sobie :)

// Kończymy pracę z ncurses
endwin();
```



# Atrybuty i pisanie



```
#include <ncurses.h>

// Tworzy kolor (litera, tło) o indeksie #1
init_pair (1, COLOR_RED, COLOR_BLACK);

// Czyści cały ekran
clear ();

// Ustawia kolor na #1
attron (COLOR_PAIR(1));

// Włącza wyróżnione („podrubię”) litery oraz podkreślenie
attron (A_BOLD | A_UNDERLINE);

// Piszę tekst w pozycji x = 4, y = 7
mvprintw (7, 4, „Mój tekst”); // uwaga na odwrotną kolejność y, x !

// Wyłącza kolor i pogrubienie
attroff (COLOR_PAIR(1) | A_BOLD);
```

Ekran ncurses nie przesuwają się jak normalnie w konsoli, ale położenie znaków pozostaje stałe!

# Programowanie jest fantastyczne!!!

