

# Języki i Paradygmaty Programowania II

## Laboratorium 5: Szablony

# W dzisiejszym odcinku

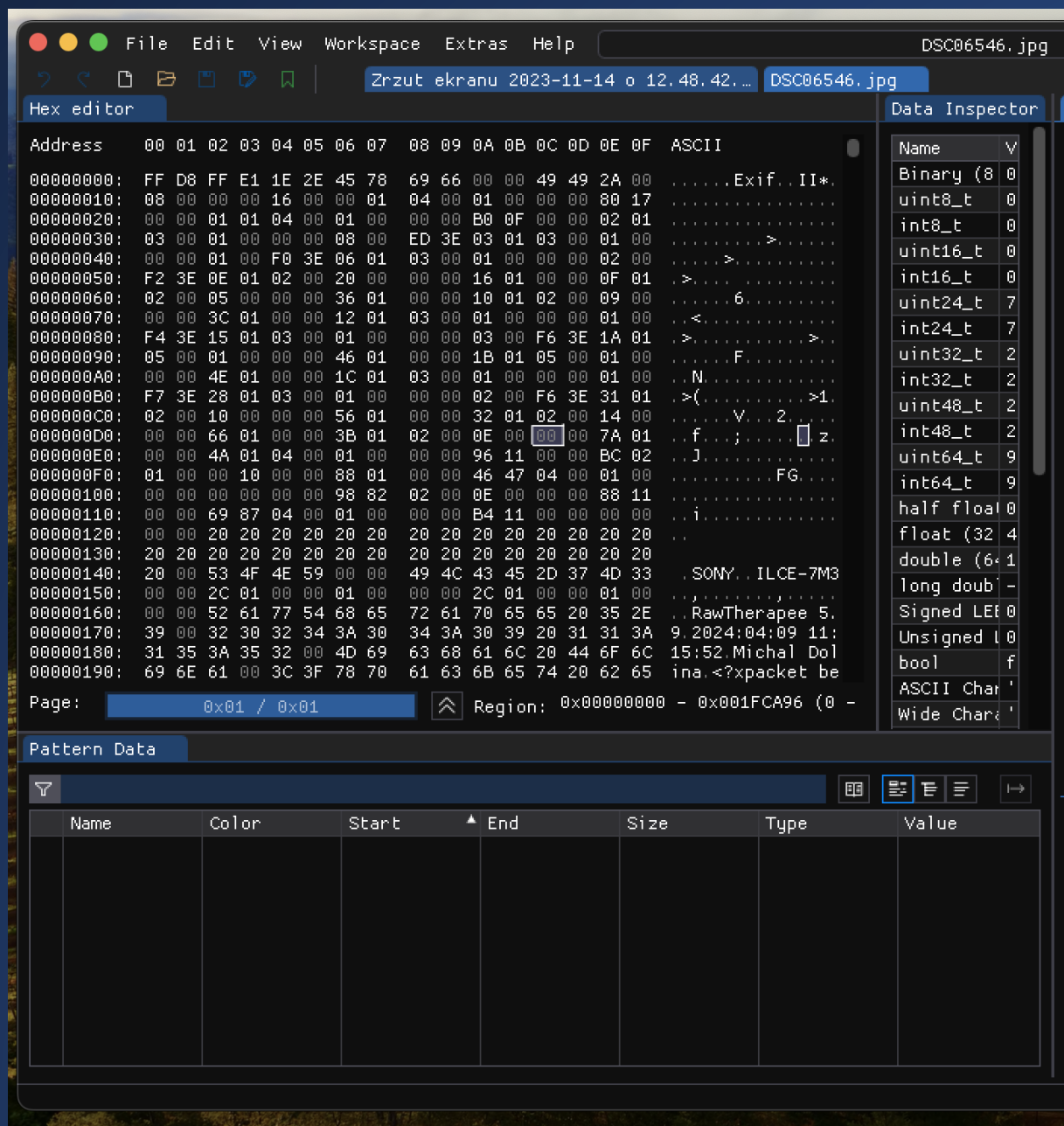
- Obsługa Plików Binarnych w C++
- Wyjątki
- Run-Time Type Information
- Rzutowanie w C++



# Pliki binarne - przypomnienie

- Surowe dane – rzadko zrozumiałe dla człowieka
- Hexedytor
- Szybsze niż pliki tekstowe
- Pozwalają na „zrzut pamięci”





# fstream

- Koncept strumieni - dla plików
- Klasa bazowa dla ifstream i ofstream
- Operatory strumienia >> <<
- Tryby pracy z plikiem



# Tryb pracy z plikiem

- `ios::in` - wejście
- `ios::out` - wyjście
- `ios::ate` – "at end"
- `ios::trunc` – truncate – usuwamy dane istniejące
- `ios::append` - dopisujemy
- `ios::binary` – binary mode



# Wczytywanie z pliku binarnego

- Otwieramy strumień
- Tryb otwarcia
- Pobieramy dane – get, readsome, getline, read lub >>
- Zamykamy strumień
- Przewijanie – tellg\* i seekg\*





# Przykład

```
#include <fstream>
int main()
{
    int x;
    streampos pos;
    ifstream infile;
    infile.open("silly.dat", ios::binary | ios::in);
    infile.seekp(243, ios::beg); // move 243 bytes into the file
    infile.read(&x, sizeof(x));
    pos = infile.tellg();
    infile.seekp(0, ios::end); // seek to the end of the file
    infile.seekp(-10, ios::cur); // back up 10 bytes
    infile.close();
}
```





# More cpp way

```
#include <fstream>
```

```
#include <iterator>
```

```
#include <vector>
```

```
int main()
```

```
{
```

```
    std::ifstream input( "C:\\Final.gif", std::ios::binary );
```

```
    // copies all data into buffer
```

```
    std::vector<unsigned char> buffer(std::istreambuf_iterator<char>(input), {});
```

```
}
```



# Zapis do pliku binarnego

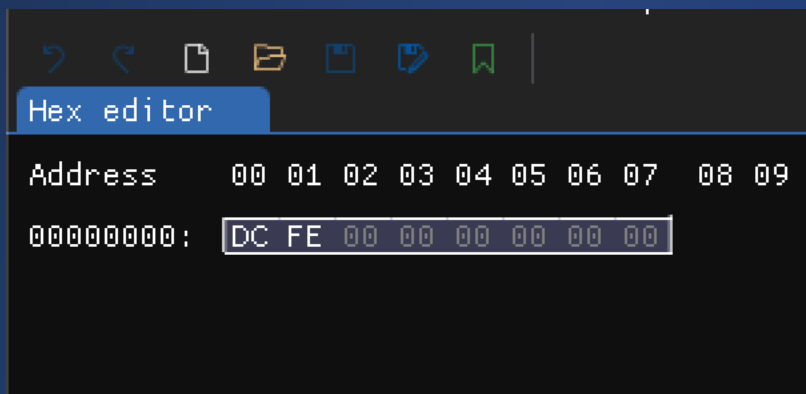
- Otwieramy strumień
- Tryb otwarcia
- Zapisujemy dane – put, write lub <<
- Zamykamy strumień
- Przewijanie – tellp\* i seekp\*



# Przykład

```
#include <iostream>
#include <fstream>
#include <cstdint>
using namespace std;
int main()
{
    ofstream file;
    uint64_t myuint = 0xFEDC;
    file.open("test.bin", ios::binary);
    file.write(reinterpret_cast<char*>(&myuint), sizeof(myuint));
    file.close();
    return 0;
}
```





# Wyjątki

- „Sytuacja Wyjątkowa”
- Dawno temu – zwrócenie wartości
- Słowa kluczowe: try, catch, throw
- Noexcept (C++11)



# try... catch

```
try {
```

```
// code here
```

```
}
```

```
catch (int param) { cout << "int exception"; }
```

```
catch (char param) { cout << "char exception"; }
```

```
catch (...) { cout << "default exception"; }
```



# Rzucanie wyjątków

```
#include <iostream>
```

```
using namespace std;
```

```
int main ()
```

```
{
```

```
    try
```

```
    {
```

```
        throw 20;
```

```
    }
```

```
    catch (int e)
```

```
    {
```

```
        cout << "An exception occurred. Exception Nr. " << e << '\n';
```

```
    }
```

```
    return 0;
```

```
}
```





# Wyjątki wielopziomowe

```
try {  
    try  
    {  
        // code here  
    }  
    catch (int n)  
    {  
        throw;  
    }  
}  
catch (...)  
{  
    cout << "Exception occurred";  
}
```



# Własne wyjątki

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

class myexception: public exception {
    virtual const char* what() const throw() {
        return "My exception happened";
    }
} myex;

int main () {
    try {
        throw myex;
    }
    catch (exception& e) {
        cout << e.what() << '\n';
    }
    return 0;
}
```



# Noexcept

- Sprawdzamy czy fragment kodu może rzucić wyjątek
- Pozwala zadeklarować że funkcja może zgłaszać wyjątki (C++17) – jeśli rzucimy wyjątek z funkcji noexcept – zadziała mechanizm `std::terminate`

```
void myFunction() noexcept {  
    // Function implementation  
}
```



# RTTI

RTTI (Run Time Type Information – informacja o typie w trakcie wykonywania programu) jest techniką stosowaną w nowoczesnych obiektowych językach programowania. Technika ta polega na dołączeniu do kodu programu informacji o typach (klasach), czasami też ich właściwościach i dostępnych metodach.

~Polska Wiki



# RTTI w C++

- Nagłówek `<typeinfo>` lub `import (C++20)`
- Konstrukcja `typeid(X)`
- `std::typeinfo`
- `name()`, `hash_code()`, `before()`, `operator==`



# Przykład

```
#include <iostream>
```

```
#include <typeinfo>
```

```
#include <string>
```

```
int main () {  
    int a;  
    double b;  
    std::string str;  
    std::wstring wstr;  
    std::cout << typeid(a).name() << std::endl;  
    std::cout << typeid(b).name() << std::endl;  
    std::cout << typeid(str).name() << std::endl;  
    std::cout << typeid(wstr).hash_code() << std::endl;  
}
```



```
promeger@MacBook-Air-Jakub Desktop % ./a.out
i
d
NSt3__112basic_stringIcNS_11char_traitsIcEENS_9allocatorIcEEEE
15067349643040121425
promeger@MacBook-Air-Jakub Desktop %
```



# Rzutowanie typów w C++

- Rzutowanie – zamiana z jednego typu na drugi
- „Classic way” – `T = (T)(variable);`
- Operatory rzutowania



# Operatory rzutowania w C++

- `static_cast` – Najczęściej używany – rzutowanie statyczne (compile time)
- `dynamic_cast` – Rzutowanie dynamiczne (kiedy nie można sprawdzić podczas kompilacji). Najczęściej używany przy dziedziczeniu
- `const_cast` – Konwersja z uwzględnieniem stałych. Pozwala np. na modyfikacje w metodzie `const`
- `reinterpret_cast` – Konwersja wskaźnika na dowolny inny typ wskaźnikowy



# Przykłady – static cast

```
#include <iostream>
using namespace std;
// Driver code
int main()
{
    float f = 3.5;
    // Implicit type case
    // float to int
    int a = f;
    cout << "The Value of a: " << a;
    // using static_cast for float to int
    int b = static_cast<int>(f);
    cout << "\nThe Value of b: " << b;
}
```



# Dynamic cast

```
#include <iostream>
using namespace std;
class Base {
    virtual void print() {cout << "Base" << endl;}
};
class Derived1 : public Base {
    void print() {cout << "Derived1" << endl;}
};

int main()
{
    Derived1 d1;


    Base* bp = dynamic_cast<Base*>(&d1);
    Derived1* dp2 = dynamic_cast<Derived1*>(bp);
    return 0;
}
```



# Const cast

```
#include <iostream>
using namespace std;
class student
{
    private:
    int roll;
    public:
    // constructor
    student(int r):roll(r) {}

    void fun() const
    {
        ( const_cast <student*> (this) )->roll = 5;
    }
    int getRoll() { return roll; }
};
```



# Reinterpret cast

```
#include <iostream>
using namespace std;
int main()
{
    int* p = new int(65);
    char* ch = reinterpret_cast<char*>(p);
    cout << *p << endl;
    cout << *ch << endl;
    cout << p << endl;
    cout << ch << endl;
    return 0;
}
```



# Zadanka





Dziękujemy za uwagę

