

# Podstawy Informatyki

**Katedra Telekomunikacji, EiT**

dr inż. Jarosław Bułat

[kwant@agh.edu.pl](mailto:kwant@agh.edu.pl)

# Plan prezentacji

- » Klasyfikacja języków programowania, składnia, semantyka, ....
- » Przetwarzanie kodu źródłowego do programu
- » Drugi program w C++
  - podstawowe I/O,
  - deklaracje zmiennych
- » GIT - pull, commit, push



# Projekty

*na zaliczenie i for fun*

# Projekty

- » “Na zaliczenie” czyli zamiast chodzić na lab - tylko dla “pro”
  - u prowadzących Lab
  - u mnie
- » “for fun”
  - AI/ML - TensorFlow light (coś lekkiego na RPI)
  - software+hardware (jakiś robot, czujnik, etc...)
  - chętnie w zespole (git+discord)
  - ja będę się przyglądał, pomagał, naprowadzał ale nie będę lokomotywą (sami musicie się zorganizować)

# Elementy języka programowania

# Elementy języka programowania

- » Składnia (ang. syntax)
- » Semantyka
- » Typy danych
- » Biblioteki standardowe

# Elementy języka programowania

- » **Składnia** (ang. syntax) – rodzaje dostępnych symboli i zasady według których możemy je łączyć
- » Semantyka – analiza składni nie bierze pod uwagę znaczenia (!)
- » Typy danych – kod poprawny składniowo nie musi być poprawny semantycznie
- » Biblioteki standardowe – jak tworzyć polecenia i wyrażenia
- jaką postać mają instrukcje sterujące
- jak zapisać deklarację (zmiennej/funkcji/...)

# Elementy języka programowania

- » **Składnia** (ang. syntax)
- » Semantyka
- » Typy danych
- » Biblioteki standardowe

```
// decoded, check next synchro
if (adts_head_idx_+5 < superframe_cifs_){
    if (crc_errors<num_aus)
        adts_head_idx_ += 5;
    else{
        adts_head_idx_ += 4;
    }
} else{
    adts_head_idx_ = 0;
    superframe_cifs_ = 0;
    return;
}
CircshiftBuff(data);
```



# Elementy języka programowania

- » Składnia (ang. syntax)
  - » **Semantyka**
  - » Typy danych
  - » Biblioteki standardowe
- precyzyjna definicja poszczególnych symboli oraz ich funkcji w programie
  - najczęściej jest to słowna definicja (formalizmy są niepraktyczne)
  - część błędów semantycznych można wychwycić w trakcie kompilacji (brak wywoływanej funkcji), część dopiero w trakcie wykonania kodu
    - np. czy każda nazwa (identyfikator) jest zadeklarowany przed pierwszym użyciem

# Elementy języka programowania

- » Składnia (ang. syntax) – precyzyjna definicja poszczególnych symboli oraz ich funkcji w programie
- » **Semantyka** – najczęściej jest to słowna definicja (formalizmy są niepraktyczne)
- » Typy danych – część błędów semantycznych można wychwycić w trakcie kompilacji (brak wywoływanej funkcji), część dopiero w trakcie wykonania kodu
- » Biblioteki standardowe

```
if (a = b){  
    // something ...  
}
```

```
int calculateArea(int width, int height){  
    return width + height;  
}
```

# Elementy języka programowania

- » Składnia (ang. syntax)
  - » Semantyka
  - » **Typy danych**
  - » Biblioteki standardowe
- typy danych na których możemy operować ich właściwości, dozwolone operacje
  - typy wbudowane (podstawowe) zazwyczaj:
    - liczby całkowite (int)
    - liczby zmiennoprzecinkowe (float, double)
    - ciągi tekstowe (char[])

# Elementy języka programowania

- » Składnia (ang. syntax)
  - » Semantyka
  - » **Typy danych**
  - » Biblioteki standardowe
- typy danych na których możemy operować ich właściwości, dozwolone operacje
  - typy wbudowane (podstawowe)
- zazwyczaj:
- liczby całkowite (int)
  - liczby zmiennoprzecinkowe (float, double)
  - ciągi tekstowe (char[])

```
if ("1" == true ){  
}
```

```
// semantic error  
// no syntax error
```

# Elementy języka programowania

- » Składnia (ang. syntax)
  - » Semantyka
  - » **Typy danych**
  - » Biblioteki standardowe
- typy danych na których możemy operować ich właściwości, dozwolone operacje
  - typy wbudowane (podstawowe) zazwyczaj:
    - liczby całkowite (int)
    - liczby zmiennoprzecinkowe (float, double)
    - ciągi tekstowe (char[])
  - **statyczne typowanie**
    - jawne
    - inferencyjne (automatyczne)
  - **dynamiczne typowanie**

# Elementy języka programowania

- » Składnia (ang. syntax)
- » Semantyka
- » **Typy danych**
- » Biblioteki standardowe

```
// C++  
int result = 0;
```

```
// result is of the type int  
// 1.3 is of the type float
```

```
result = 1.8;  
// result == 1, still int
```

- typy danych na których możemy operować ich właściwości, dozwolone operacje
- typy wbudowane (podstawowe)  
zazwyczaj:
  - liczby całkowite (int)
  - liczby zmiennoprzecinkowe (float, double)
  - ciągi tekstowe (char[])
- **statyczne typowanie**
  - **jawne**
  - inferencyjne (automatyczne)
- **dynamiczne typowanie**

# Elementy języka programowania

- » Składnia (ang. syntax)
- » Semantyka
- » **Typy danych**
- » Biblioteki standardowe

```
// C++  
float result;  
int input = 1;  
  
result = input;  
// conversion int->float
```

- typy danych na których możemy operować ich właściwości, dozwolone operacje
- typy wbudowane (podstawowe) zazwyczaj:
  - liczby całkowite (int)
  - liczby zmiennoprzecinkowe (float, double)
  - ciągi tekstowe (char[])
- **statyczne typowanie**
  - jawne
  - inferencyjne (automatyczne)
- **dynamiczne typowanie**

# Elementy języka programowania

- » Składnia (ang. syntax)
- » Semantyka
- » **Typy danych**
- » Biblioteki standardowe

```
# python
```

```
result = 1      # int  
result = 1.0    # float  
result = 'abc'  # str
```

- typy danych na których możemy operować ich właściwości, dozwolone operacje
- typy wbudowane (podstawowe)  
zazwyczaj:
  - liczby całkowite (int)
  - liczby zmiennoprzecinkowe (float, double)
  - ciągi tekstowe (char[])
- **statyczne typowanie**
  - jawne
  - inferencyjne (automatyczne)
- **dynamiczne typowanie**



# Elementy języka programowania

- » Składnia (ang. syntax)
  - » Semantyka
  - » **Typy danych**
  - » Biblioteki standardowe
- typy danych na których możemy operować ich właściwości, dozwolone operacje
  - typy wbudowane (podstawowe) zazwyczaj:
    - liczby całkowite (int)
    - liczby zmiennoprzecinkowe (float, double)
    - ciągi tekstowe (char[])
  - **słabe albo mocne typowanie**
    - subiektywne, lepiej używać określenia statycznie/dynamicznie typowany język

# Elementy języka programowania

- » Składnia (ang. syntax)
  - » Semantyka
  - » Typy danych
  - » **Biblioteki standardowe (i środowisko uruchomieniowe)**
- zazwyczaj podstawowy zestaw funkcji/procedur do obsługi:
    - standardowego wejścia/wyjścia (konsola)
    - plików (pamięci masowej)
    - pamięci operacyjnej
    - wielowątkowości
    - operacje na ciągach tekstowych (text)
    - podstawowe typy danych + operacje na nich
  - początkujący programiści często traktują bibliotekę standardową jako część **implementacji** języka

# C++

C++ to “nowy, lepszy C”

# C++

- » Język programowania ogólnego przeznaczenia
- » Wysoka wydajność, bezpośredni dostęp do zasobów (wada i zaleta)
- » Wieloplatformowość (hardware/OS)
- » **Wieloparadygmatowy** (proceduralny, obiektowy, generyczny)
- » Umożliwia abstrakcję danych
- » Zachowanie zgodności z “C”
- » Jedyny język posiadający dobrej jakości kompilator na praktycznie wszystkie platformy sprzętowe i programowe
- » Szybki rozwój w ostatnich latach (C++17, 20, 23...) co 3 lata nowy standard, wiele rozszerzeń np. SYCL (Intel używa w OneAPI)
- » Stary, trudny, niebezpieczny

# C++

and	const_cast	for	or_eq	template	wchar_t	{...}
and_eq	continue	friend	private	this	while	"\n"
asm	default	goto	protected	throw	xor	/*...*/
auto	delete	if	public	true	xor_eq	//
bitand	do	inline	register	try		< > <= >=
bitor	double	int	reinterpret_cast	typedef		== !=
bool	dynamic_cast	long	return	typeid		=
break	else	mutable	short	typename		- + / *
case	enum	namespace	signed	union		<< >>
catch	explicit	new	sizeof	unsigned		...
char	export	not	static	using		
class	extern	not_eq	static_cast	virtual		
compl	false	operator	struct	void		
const	float	or	switch	volatile		

# Zmienne

- » **zmienna** to “konstrukcja programistyczna”, posiada:
  - **nazwę** (etykieta, identyfikator)
  - **miejsce** przechowywania (adres, wielkość)
  - **wartość** (stan)
  - **typ**
- » w kodzie programu możemy odwołać się do zmiennej przez nazwę lub miejsce (adres w pamięci)
- » zmienną można odczytać, zapisać
- » **w C++ zmienną trzeba zadeklarować**

# Typy zmiennych

- » **znakowe**: przechowywanie znaków, cyfr, symboli, ... **char**
- » **całkowito-liczbowe**:
  - ze znakiem (signed), wartości: -128...0...+127 **int**
  - bez znaku (unsigned), wartości: 0...255 **signed int**  
**unsigned int**
- » **zmiennie-przecinkowe** **float**
  - pojedynczej, podwójnej i poczwórnej precyzji **double**
- » **logiczne**: bool **bool**
- » **inne**: void, null
- » **zakresy zmiennych w pliku nagłówkowym <climits>**

Group	Type names*	Notes on size / precision
Character types	<code>char</code>	Exactly one byte in size. At least 8 bits.
	<code>char16_t</code>	Not smaller than <code>char</code> . At least 16 bits.
	<code>char32_t</code>	Not smaller than <code>char16_t</code> . At least 32 bits.
	<code>wchar_t</code>	Can represent the largest supported character set.
Integer types (signed)	<code>signed char</code>	Same size as <code>char</code> . At least 8 bits.
	<code>signed short int</code>	Not smaller than <code>char</code> . At least 16 bits.
	<code>signed int</code>	Not smaller than <code>short</code> . At least 16 bits.
	<code>signed long int</code>	Not smaller than <code>int</code> . At least 32 bits.
	<code>signed long long int</code>	Not smaller than <code>long</code> . At least 64 bits.
Integer types (unsigned)	<code>unsigned char</code>	(same size as their signed counterparts)
	<code>unsigned short int</code>	
	<code>unsigned int</code>	
	<code>unsigned long int</code>	
	<code>unsigned long long int</code>	
Floating-point types	<code>float</code>	
	<code>double</code>	Precision not less than <code>float</code>
	<code>long double</code>	Precision not less than <code>double</code>
Boolean type	<code>bool</code>	
Void type	<code>void</code>	no storage
Null pointer	<code>decltype(nullptr)</code>	



# Deklaracje zmiennych

```
#include <iostream>
```

```
int main(){  
    int a;
```

```
}
```

# Deklaracje zmiennych

```
#include <iostream>
```

```
int main(){
```

```
    int a;
```

```
// declaration
```

```
    int b, c, d;
```

```
/* declaration */
```

```
    float myNumber;
```

```
}
```

# Deklaracje zmiennych

```
#include <iostream>
```

```
int main(){
```

```
    int a;
```

```
// declaration
```

```
    int b, c, d;
```

```
/* declaration */
```

```
    float myNumber;
```

```
    a = 1;
```

```
// assignment
```

```
    b = -3;
```

```
    c = d = 7;
```

```
    /*
```

```
     * |-wartosc symbol_przypisania wyrażenie [terminator]
```

```
     * |-value assignemnt_symbol expression [ending statement]
```

```
    */
```

```
}
```

# Inicjalizacja zmiennych

```
#include <iostream>
```

```
int main(){  
    int a = 1;           // declaration and initialization  
    int b(3);  
    int c;  
    float myNumber;  
  
}
```

# Inicjalizacja zmiennych

```
#include <iostream>
```

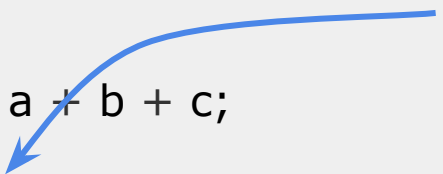
```
int main(){  
    int a = 1;           // declaration and initialization  
    int b(3);  
    int c;  
    float myNumber;  
  
    c = -3;  
    int d = a + b + c;  
  
}
```

# Inicjalizacja zmiennych

```
#include <iostream>
```

```
int main(){  
    int a = 1;           // declaration and initialization  
    int b(3);  
    int c;  
    float myNumber;  
  
    c = -3;  
    int d = a + b + c;  
  
    std::cout << "d=" << d << std::endl;  
    std::cout << "float=" << myNumber << std::endl;  
}
```

**cout == console output**



# Inicjalizacja zmiennych

```
#include <iostream>
```

```
int main(){  
    int a = 1;           // declaration and initialization  
    int b(3);  
    int c;  
    float myNumber;  
  
    int d = a + b + c;  
    c = -3;  
  
    std::cout << "d=" << d << std::endl;  
    std::cout << "float=" << myNumber << std::endl;  
}
```

# Inicjalizacja zmiennych

```
#include <iostream>
```

```
int main(){  
    int a = 1;           // declaration and initialization  
    int b(3);  
    int c;  
    float myNumber;
```

```
    int d = a + b + c;    ← wrong (very wrong!!!)  
    c = -3;
```

```
    std::cout << "d=" << d << std::endl;  
    std::cout << "float=" << myNumber << std::endl;
```

```
}
```





quiz

**PI02\_init**

**socrative.com**

- login
- student login

Room name:

**KWANTAGH**

# stała zmienna !@#\$\$%^

```
// #define stala 10  
const int stala = 10;
```

```
std::cout << stala << std::endl;
```

- » **Const** to stała
- » Trzeba **zainicjalizować** podczas deklaracji

# stała zmienna !@#\$\$%^

```
// #define stala 10  
const int stala;
```

```
std::cout << stala << std::endl;
```

- » Const to stała
- » Trzeba **zainicjalizować** podczas deklaracji, jeżeli nie to:  
**error: uninitialized const 'stala'**

# stała zmienna !@#\$\$%^

```
// #define stala 10
const int stala = 10;

stala++;

std::cout << stala << std::endl;
```

- » Const to stała
- » Trzeba zainicjalizować podczas deklaracji, jeżeli nie to:  
**error: uninitialized const 'stala'**
- » **Próba zmiany "stałej"** kończy się komunikatem:  
**error: increment of read-only variable 'stala'**  
**stala++;**

# stała zmienna !@#\$\$%^

```
// #define stala 10
const int stala = 10;

stala++;

std::cout << stala << std::endl;
```

Zaletą stałej “**const**” nad stałą deklarowaną jako **#define**, jest TYP. Ma typ, **kompilator może optymalizować i sprawdzić czy zgadza się przypisanie.**

- » Const to stała
- » Trzeba zainicjalizować podczas deklaracji, jeżeli nie to:  
**error: uninitialized const ‘stala’**
- » **Próba zmiany “stałej”** kończy się komunikatem:  
**error: increment of read-only variable ‘stala’**  
**stala++;**



quiz

**PI02\_const**

**socrative.com**

- login
- student login

Room name:

**KWANTAGH**

Mam równanie  $y=4-x$  czy  
mogę je zapisać w C++?

# NIE

- » Równania w sensie matematycznym nie da się zapisać, zaimplementować, ... (w czystym C/C++)
- » “Równanie” będzie traktowane jako **przypisanie** według schematu:

$$y = 4 - x ;$$

**I-wartość** **symbol\_przypisania** **wyrażenie** **[terminator]**

- » Wyrażenie obliczone i wstawione do przypisane (wstawione, podstawione) do zmiennej
- » Wyrażenie może być złożone: różne operacje arytmetyczne, stałe, zmienne, etc...



# Operacje arytmetyczne

```
5: int x = 1 + 2 + 3 + 4;           // 10      (std::cout<<x;)
6: int y = 20 - x;                   // -10
7: y = x * 3;                       // 30(10*3)
8: float xy = 10 * 0.73;             // 7.3
9: xy = 10.0 / 7;                   // 1.42857
10: int z = x / 3;                   // 3 (10/3)
11: z = x / 6;                      // 1 (10/6==1.6666)
12: z = x % 6;                      // 4 (reszta z dzielenia 10/6)
13:                                     // 10==1*6+4
14: unsigned int u = x - y; // 10-30==4294967276
```

- » 6: instrukcja nie powodująca żadnego efektu
- » dzielenie “x/y” na liczbach całkowitych to floor(x/y)
- » **xy** to nie jest **x\*y**
- » reszta z dzielenia “%” nie jest zdefiniowana dla liczb rzeczywistych
- » “unsigned” nie przyjmuje wartości ujemnych

# Priorytet (kolejność) operacji

```
5: int a = 1 + 2 * 3;           // 7
6: int b = 2 * 3 + 1;           // 7
7: int c = 2 * (3 + 1);         // 8
8:
9: a = 2 - 2 - 2;               // -2
10: b = (2 - 2) - 2;            // -2
11: c = 2 - (2 - 2);            // 2
12:
13: c = 2-(a = 1);              // 1
14: a = (b = 3, b + 2);         // b = 3
15:                             // a = b +2
```

- » Priorytety jak w “matematyce”, najpierw \*, / a potem +, -
- » Łączność od lewej do prawej (linia 9-11)
- » Nie stosować operacji z linii 13 i 14 (nieczytelne)
- » **Masz wątpliwości jaka kolejność, wstaw nawiasy**

# Priorytet (kolejność) operacji

```
6: int a = 3;
7: int b = 2;
8: int c = 7;
9: int d = 1;
10:
11: int x = 0;
12: x = x + a;
13: x = x + b;
14: x = x + c;
15: x = x + d;
16:
17: int y = a + b + c + d;
18:
19: int x1 = a + b;
20: int x2 = c + d;
21: int r = x1 + x2;
```

- » 12-15:
  - musi się wykonać sekwencyjnie (akumulowanie)
  - **Szybkie**: jedna instrukcja
- » 17: niemożliwe do zapisania jeżeli akumulujemy dużo liczb
- » 19:21:
  - **3 dodawania zamiast 4** ale potencjalnie każde dodawanie to dwie instrukcje
  - **możliwość zrównoleglenia!!!**
- » AMD Ryzen: **4xALU**, 2xload/store, **2xFPU**
- » **szybkość operacji arytmetycznych**:
  - **int** szybsze (i więcej!) od **float**
  - **“+,-”** szybsze od **“\*”** szybsze od **“/”**

# zapis skrócony

```
6: int a = 1;  
7: a = a + 1;           // 2  
8:  
9: int b = 1;  
10: b += 1;             // 2  
11: b *= a + 1;         // 6    b=b*(a+1)
```

- » dozwolone “skrótowe” operacje: **+=**, **-=**, **\*=**, **/=**, **%=**, **>>=**, **<<=**, **&=**, **|=**, **^=**
- » ma uzasadnienie sprzętowe (dedykowane instrukcje asemblera w CPU)
- » czytelniejsze
  - **my\_variable1**=**my\_variable1**+**my\_variable2**;
  - **my\_variable1**+=**my\_variable2**;
- » czytelniejsze jeżeli etykieta zmiennej jest długa
  - **superframe\_cifs**\_=**superframe\_cifs**+**cifs\_per\_tr**;
  - **superframe\_cifs**+=**cifs\_per\_tr**;

# inkrementacja/dekrementacja

```
6: float a = 1; // 1
7: a = a + 1;    // 2
8: a += 1;      // 3
9: a++;         // 4
10: ++a;        // 5
11:
12: a = 0;
13: float b = ++a; // 1
14:
15: a = 0;
16: float c = a++; // 0
```

- » inkrementacja/dekrementacja:
  - bardzo szybka
  - pojedyncza (krótka) instrukcja
  - różne typy danych (np. float)
  - często używane
  - czytelny kod
- » 7-9: współczesne kompilatory wyprodukują ten sam kod binarny
- » 13: **preinkrementacja** wykona się przed przypisaniem
- » 16: **postinkrementacja** wykona się po przypisaniu
- » 9: operator lewostronny
- » 10: operator prawostronny



quiz

**PI02\_arytm**

**socrative.com**

- login
- student login

Room name:

**KWANTAGH**

# Operatory ToDo

- » Do przestudiowania dla zaawansowanych:
  - operator jednoargumentowy
  - operator dwuargumentowy
  - operator lewostronny
  - operator prawostronny

# Uproszczenia

moje - na potrzeby prezentacji



# Uproszczenia

```
#include <iostream>
```

```
int main(){
```

```
    int a = 1;
```

```
    std::cout << "d=" << d << std::endl;
```

```
    std::cout << "float=" << myNumber << std::endl;
```

```
}
```

# Uproszczenia

```
#include <iostream>
```

```
int main(){
```

```
    int a = 1;
```

```
    std::cout << "d=" << d << std::endl;
```

```
    std::cout << "float=" << myNumber << std::endl;
```

```
    return 0;
```

```
}
```

# Uproszczenia

```
#include <iostream>
```

```
using namespace std;
```

```
int main(){
```

```
    int a = 1;
```

```
    std::cout << "d=" << d << std::endl;
```

```
    std::cout << "float=" << myNumber << std::endl;
```

```
}
```

# Uproszczenia

```
#include <iostream>
```

```
using namespace std;
```

```
int main(){
```

```
    int a = 1;
```

```
    std::cout << "d=" << d << std::endl;
```

```
    cout << "float=" << myNumber << endl;
```

```
}
```

# Uproszczenia

```
#include <iostream>
```

```
using namespace std;
```

```
int main(){
```

```
    int a = 1;
```

```
    cout << "d=" << d << endl;
```

```
    cout << "float=" << myNumber << endl;
```

```
}
```

# Uproszczenia

```
#include <iostream>
```

```
int main(){
```

```
    int a = 1;
```

```
    cout << "d=" << d << endl;
```

```
    cout << "float=" << myNumber << endl;
```

```
}
```

# Uproszczenia

```
int main(){  
    int a = 1;  
  
    cout << "d=" << d << endl;  
    cout << "float=" << myNumber << endl;  
  
}
```

# Uproszczenia

```
int a = 1;
```

```
cout << "d=" << d << endl;
```

```
cout << "float=" << myNumber << endl;
```



# Uproszczenia

```
int a = 1;
```

```
cout << "d=" << d << endl;
```

```
cout << "float=" << myNumber << endl;
```

Napisałem program  
kto go przetworzy do asm?

```
#include <iostream>
```

```
int main(){  
    std::cout << "Hello world" << std::endl;  
}
```

```
~/D/P/lab_02_fistCPP> g++ ex1.cpp -o ex1  
~/D/P/lab_02_fistCPP> ls -al  
razem 40  
drwxrwxr-x 2 kwant kwant 4096 paź 7 18:33 ./  
drwxrwxr-x 5 kwant kwant 4096 paź 7 18:29 ../  
-rwxrwxr-x 1 kwant kwant 9216 paź 7 18:33 ex1*  
-rw-rw-r-- 1 kwant kwant 76 paź 7 18:29 ex1.cpp  
~/D/P/lab_02_fistCPP> ./ex1  
Hello world  
~/D/P/lab_02_fistCPP> █
```

## Przetwarzanie kodu źródłowego na program (plik wykonywalny)

kod źródłowy (\*.cc), nagłówki (\*.h)

łączenie plików, parsowanie symboli #...

pliki obiektowe (\*.o)

biblioteki statyczne (\*.a) →

plik wykonywalny

biblioteki współdzielone (\*.so) →

wejście →

Editor (IDE)

Preprocesor

Compiler

Linker

Loader

CPU

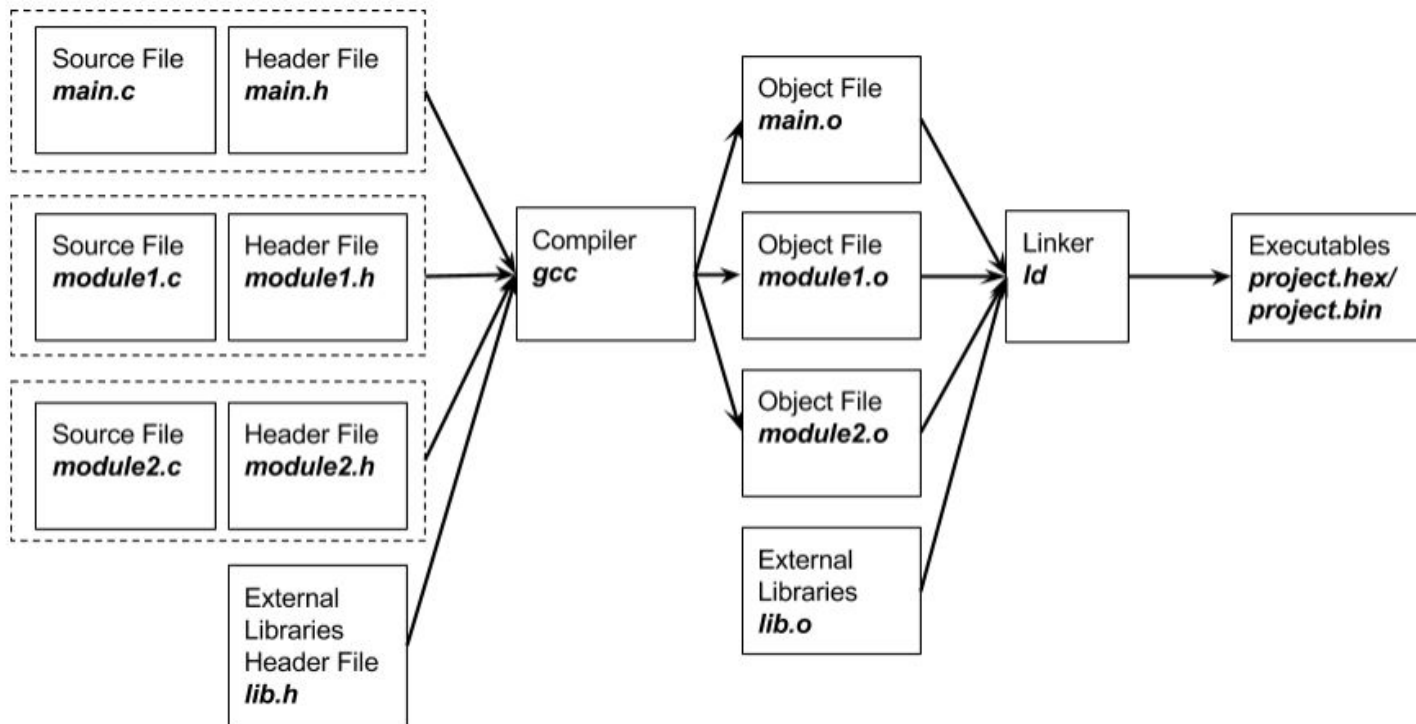
pisanie kodu

“kompilator”

OS

wyjście

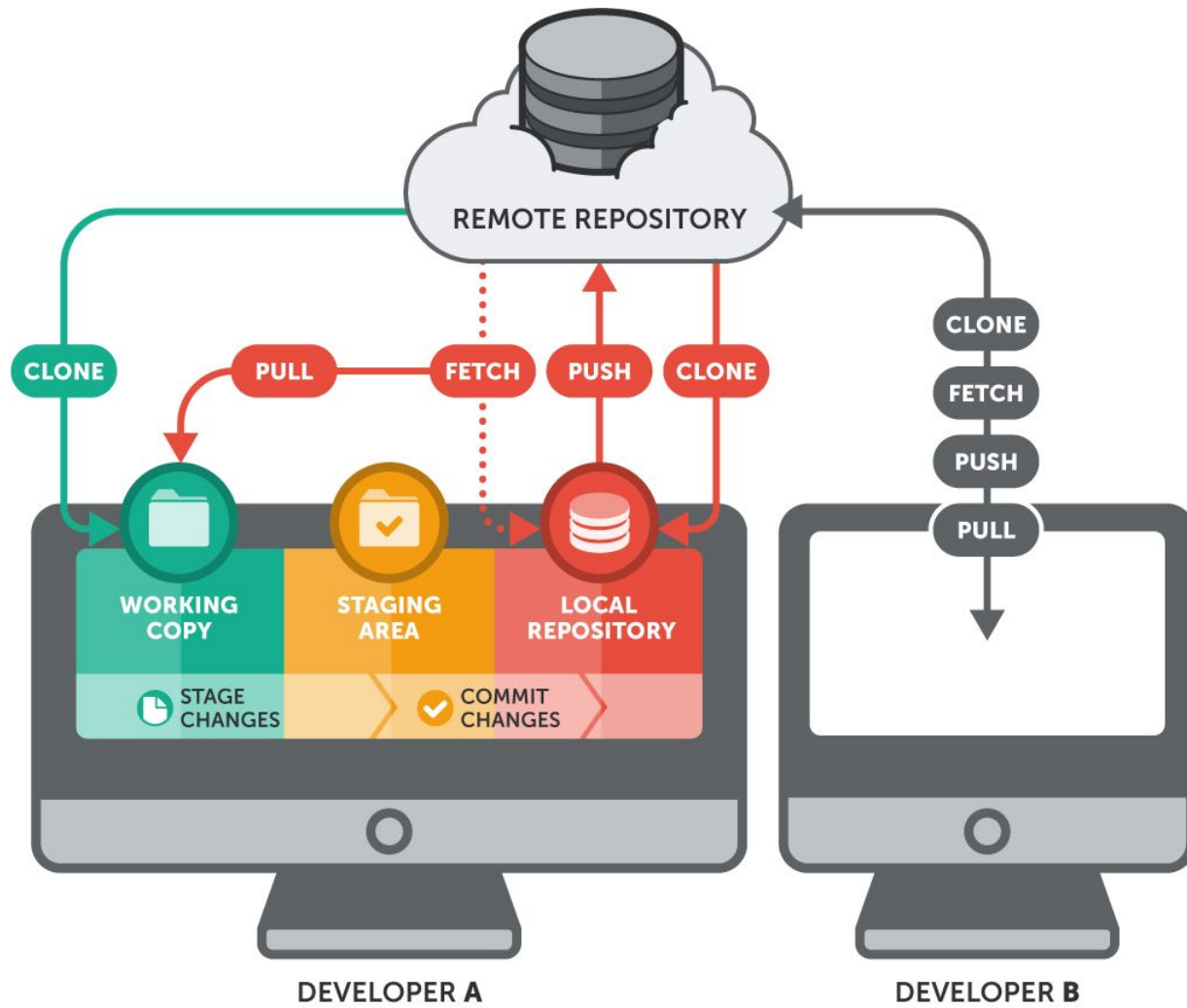
# kompilator - wiele plików





# GIT

podstawowy “workflow”



# GIT - podstawy

» TODO: konsola linuxa (shell)



# GIT

```
> git clone https://gitlab.com/gr/pro.git
```

- » TODO: konsola linuxa (shell)
- » Ściągnij (clone == skopiuj/sklonuj) repozytorium zewnętrzne (remote) do lokalnego katalogu

# GIT - podstawy

```
> git clone https://gitlab.com/gr/pro.git
```

- » TODO: konsola linuxa (shell)
- » Ściągnij (clone == skopiuj/sklonuj) repozytorium zewnętrzne (remote) do lokalnego katalogu
- » “Repozytorium” to coś więcej niż tylko katalog z plikami, zawiera również historię zmian, opisy, metadane, etc...
- » Od tego momentu na lokalnym dysku będzie częściowa kopia repozytorium

# GIT - podstawy

```
> git clone https://gitlab.com/gr/pro.git
```

- » TODO: konsola linuxa (shell)
- » Ściągnij (clone == skopiuj/sklonuj) repozytorium zewnętrzne (remote) do lokalnego katalogu
- » “Repozytorium” to coś więcej niż tylko katalog z plikami, zawiera również historię zmian, opisy, metadane, etc...
- » Od tego momentu na lokalnym dysku będzie częściowa kopia repozytorium
- » Zostanie utworzony katalog **pro**

# GIT - podstawy

```
> git clone https://gitlab.com/gr/pro.git  
> cd pro/
```

- » TODO: konsola linuxa (shell)
- » Ściągnij (clone == skopiuj/sklonuj) repozytorium zewnętrzne (remote) do lokalnego katalogu
- » “Repozytorium” to coś więcej niż tylko katalog z plikami, zawiera również historię zmian, opisy, metadane, etc...
- » Od tego momentu na lokalnym dysku będzie częściowa kopia repozytorium
- » Zostanie utworzony katalog **pro**

# GIT - podstawy

```
> git clone https://gitlab.com/gr/pro.git  
> cd pro/  
> echo "xxx" > text.txt
```

- » TODO: konsola linuxa (shell)
- » Ściągnij (clone == skopiuj/sklonuj) repozytorium zewnętrzne (remote) do lokalnego katalogu
- » "Repozytorium" to coś więcej niż tylko katalog z plikami, zawiera również historię zmian, opisy, metadane, etc...
- » Od tego momentu na lokalnym dysku będzie częściowa kopia repozytorium
- » Zostanie utworzony katalog pro
- » **Zmień** coś w katalogu

# GIT - podstawy

```
> git clone https://gitlab.com/gr/pro.git  
> cd pro/  
> echo "xxx" >text.txt  
> git add text.txt
```

- » TODO: konsola linuxa (shell)
- » Ściągnij (clone == skopiuj/sklonuj) repozytorium zewnętrzne (remote) do lokalnego katalogu
- » "Repozytorium" to coś więcej niż tylko katalog z plikami, zawiera również historię zmian, opisy, metadane, etc...
- » Od tego momentu na lokalnym dysku będzie częściowa kopia repozytorium
- » Zostanie utworzony katalog pro
- » Zmień coś w katalogu
- » Zarejestruj nowy plik (dodaj plik do repozytorium lokalnego)

# GIT - podstawy

```
> git clone https://gitlab.com/gr/pro.git  
> cd pro/  
> echo "xxx" >text.txt  
> git add text.txt  
> git commit -m "first version"
```

- » TODO: konsola linuxa (shell)
- » Ściągnij (clone == skopiuj/sklonuj) repozytorium zewnętrzne (remote) do lokalnego katalogu
- » "Repozytorium" to coś więcej niż tylko katalog z plikami, zawiera również historię zmian, opisy, metadane, etc...
- » Od tego momentu na lokalnym dysku będzie częściowa kopia repozytorium
- » Zostanie utworzony katalog pro
- » Zmień coś w katalogu
- » Zarejestruj nowy plik (dodaj plik do repozytorium lokalnego)
- » **Zarejestruj** zmianę w pliku

# GIT - podstawy

```
> git clone https://gitlab.com/gr/pro.git  
> cd pro/  
> echo "xxx" >text.txt  
> git add text.txt  
> git commit -m "first version"  
> git push
```

- » TODO: konsola linuxa (shell)
- » Ściągnij (clone == skopiuj/sklonuj) repozytorium zewnętrzne (remote) do lokalnego katalogu
- » "Repozytorium" to coś więcej niż tylko katalog z plikami, zawiera również historię zmian, opisy, metadane, etc...
- » Od tego momentu na lokalnym dysku będzie częściowa kopia repozytorium
- » Zostanie utworzony katalog pro
- » Zmień coś w katalogu
- » Zarejestruj nowy plik (dodaj plik do repozytorium lokalnego)
- » Zarejestruj zmianę pliku
- » Prześlij (**wypchnij**) zmianę do zewnętrznego repozytorium



# GIT - podstawy

```
> git clone https://gitlab.com/gr/pro.git  
> cd pro/  
> echo "xxx" >text.txt  
> git add text.txt  
> git commit -m "first version"  
> git push  
> git push origin master
```

(tylko pierwszy raz)

- » TODO: konsola linuxa (shell)
- » Ściągnij (clone == skopiuj/sklonuj) repozytorium zewnętrzne (remote) do lokalnego katalogu
- » "Repozytorium" to coś więcej niż tylko katalog z plikami, zawiera również historię zmian, opisy, metadane, etc...
- » Od tego momentu na lokalnym dysku będzie częściowa kopia repozytorium
- » Zostanie utworzony katalog pro
- » Zmień coś w katalogu
- » Zarejestruj nowy plik (dodaj plik do repozytorium lokalnego)
- » Zarejestruj zmianę pliku
- » Prześlij (**wypchnij**) zmianę do zewnętrznego repozytorium

# GIT - podstawy

```
> git clone https://gitlab.com/gr/pro.git
> cd pro/
> echo "xxx" >text.txt
> git add text.txt
> git commit -m "first version"
> echo "+yyy" >>text.txt
> git commit -m "second version"
> git push
```

» Nie muszę za każdym razem robić  
"push"

# GIT - podstawy

```
> git clone https://gitlab.com/gr/pro.git
> cd pro/
> echo "xxx" > text1.txt
> echo "xxx" > text2.txt
> git add .
```

- » Nie muszę za każdym razem robić "push"
- » Mogę utworzyć wiele plików i dodać je "hurtem" (rekurencyjnie w podkatalogach)

# GIT - podstawy

```
> git clone https://gitlab.com/gr/pro.git
> cd pro/
> echo "xxx" >text1.txt
> echo "xxx" >text2.txt
> git add .
> git commit -am "first version"
> git push
```

- » Nie muszę za każdym razem robić "push"
- » Mogę utworzyć wiele plików i dodać je "hurtem"
- » Mogę również rejestrować zmiany "**hurtowo**", komenda będzie dotyczyć tylko zmodyfikowanych plików (lub nowo utworzonych)

# GIT - podstawy

```
> git clone https://gitlab.com/gr/pro.git
> cd pro/
> echo "xxx" >text1.txt
> echo "xxx" >text2.txt
> git add .
> git commit -am "first version"
> git push
> rm text2.txt
> git add .
> git commit -am "temp. no longer ..."
```

- » Nie muszę za każdym razem robić "push"
- » Mogę utworzyć wiele plików i dodać je "hurtem"
- » Mogę również rejestrować zmiany "hurtowo", komenda będzie dotyczyć tylko zmodyfikowanych plików (lub nowo utworzonych)
- » Kasowanie pliku to rejestracja jego usunięcia !!!

# GIT - podstawy

```
> git clone https://gitlab.com/gr/pro.git
> cd pro/
> echo "xxx" >text1.txt
> echo "xxx" >text2.txt
> git add .
> git commit -am "first version"
> git push
> rm text2.txt
> git add .
> git commit -am "temp. no longer ..."
> git push
```

- » Nie muszę za każdym razem robić "push"
- » Mogę utworzyć wiele plików i dodać je "hurtem"
- » Mogę również rejestrować zmiany "hurtowo", komenda będzie dotyczyć tylko zmodyfikowanych plików (lub nowo utworzonych)
- » Kasowanie pliku to rejestracja jego usunięcia !!!
- » Zmiana zarejestrowana "na serwerze"

# GIT - podstawy

```
> git clone https://gitlab.com/gr/pro.git
> cd pro/
> echo "xxx" >text1.txt
> echo "xxx" >text2.txt
> git add .
> git commit -am "first version"
> git push
> rm text2.txt
> git add .
> git commit -am "temp. no longer ..."
> git push
```

- » Nie muszę za każdym razem robić "push"
- » Mogę utworzyć wiele plików i dodać je "hurtem"
- » Mogę również rejestrować zmiany "hurtowo", komenda będzie dotyczyć tylko zmodyfikowanych plików (lub nowo utworzonych)
- » Kasowanie pliku to rejestracja jego usunięcia !!!
- » Zmiana zarejestrowana "na serwerze"
- » **Od tego momentu zmiana jest dostępna dla innych developerów**

# GIT - team working

```
> git clone https://gitlab.com/gr/pro.git
> cd pro/
> echo "xxx" >text.txt
> git add text.txt
> git commit -m "first version"
> git push
```

» Ty zrobiłeś/zrobiłaś "**push**" czyli serwer (remote) został uaktualniony o nową wersję



# GIT - team working

```
> git clone https://gitlab.com/gr/pro.git
> cd pro/
> echo "xxx" >text.txt
> git add text.txt
> git commit -m "first version"
> git push
```

» Ty zrobiłeś/zrobiłaś **push** czyli serwer (remote) został uaktualniony o nową wersję

```
> git clone https://gitlab.com/gr/pro.git
> git pull
> cat text.txt
xxx
```

» Inny developer...  
gdzieś na drugim końcu świata...  
uaktualnia swoje lokalne repozytorium  
**pobierając najnowsze wersje plików do  
swojego lokalnego katalogu**

# GIT - wizualizacja

```
> git clone https://gitlab.com/gr/pro.git
> cd pro/
> echo "xxx" >text.txt
> git add text.txt
> git commit -m "first version"
> echo "+yyy" >>text.txt
> git commit -m "second version"
> git push
```

» Graficzna reprezentacja zmian dwóch  
"commitów"

# GIT - wizualizacja

```
> git clone https://gitlab.com/gr/pro.git  
> cd pro/  
> echo "xxx" >text.txt  
> git add text.txt  
> git commit -m "first version"  
> echo "+yyy" >>text.txt  
> git commit -m "second version"  
> git push
```

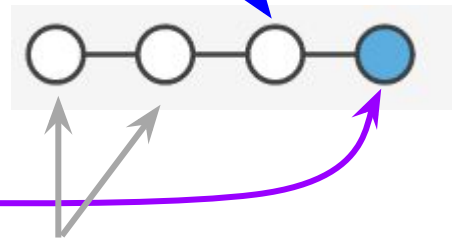
» Graficzna reprezentacja zmian dwóch  
"commitów"



# GIT - wizualizacja

```
> git clone https://gitlab.com/gr/pro.git
> cd pro/
> echo "xxx" > text.txt
> git add text.txt
> git commit -m "first version"
> echo "+yyy" >> text.txt
> git commit -m "second version"
> git push
```

» Graficzna reprezentacja zmian dwóch  
"commitów"

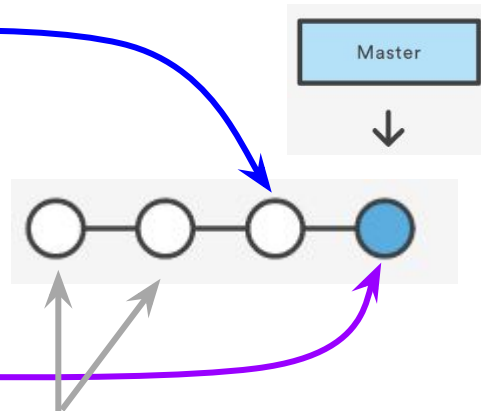


» Wcześniejsze kropki to wcześniejsze zmiany, niekoniecznie tego pliku

# GIT - wizualizacja

```
> git clone https://gitlab.com/gr/pro.git
> cd pro/
> echo "xxx" > text.txt
> git add text.txt
> git commit -m "first version"
> echo "+yyy" >> text.txt
> git commit -m "second version"
> git push
```

» Graficzna reprezentacja zmian dwóch  
"commitów"

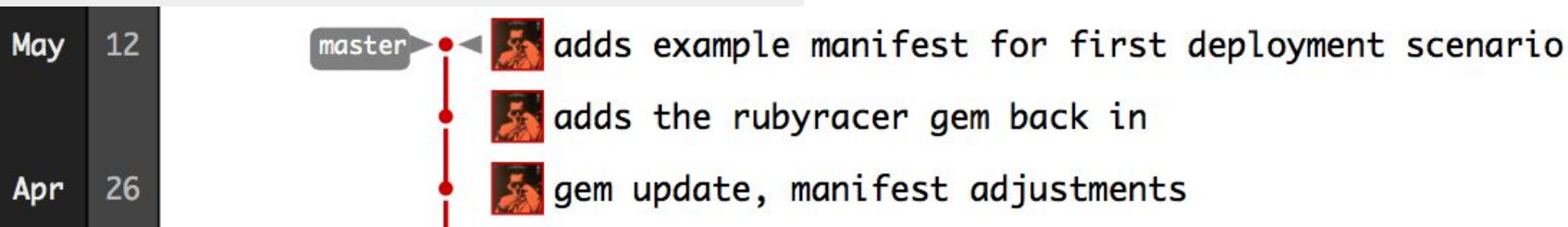


» Wcześniejsze kropki to wcześniejsze zmiany, niekoniecznie tego pliku  
» Ostatnia kropka/zmiana/commit to jest chwila obecna "now"

# GIT - wizualizacja

```
> git clone https://gitlab.com/gr/pro.git
> cd pro/
> echo "xxx" > text.txt
> git add text.txt
> git commit -m "first version"
> echo "+yyy" >> text.txt
> git commit -m "second version"
> git push
```

- » Graficzna reprezentacja zmian dwóch "commitów"
- » Wygląd wizualizacji zależy od narzędzia (gitlab, github, IDE, etc...)

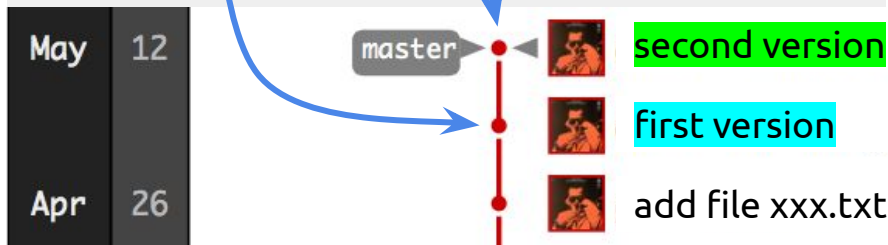


# GIT - wizualizacja

```

> git clone https://gitlab.com/gr/pro.git
> cd pro/
> echo "xxx" > text.txt
> git add text.txt
> git commit -m "first version"
> echo "+yyy" >> text.txt
> git commit -m "second version"
> git push
  
```

- » Graficzna reprezentacja zmian dwóch „commitów”
- » Wygląd wizualizacji zależy od narzędzia (gitlab, github, IDE, etc...)
- » Wizualizacja często zawiera:
  - datę
  - opis (commit message)



# GIT - konfiguracja

```
> git clone https://gitlab.com/gr/pro.git
```

- » Logowanie, konfiguracja
- » Repozytorium może być niepubliczne, więc “clone” wymaga uwierzytelnienia



# GIT - konfiguracja

```
> git clone https://gitlab.com/gr/pro.git
```

- » Logowanie, konfiguracja
- » Repozytorium może być niepubliczne, więc “clone” wymaga uwierzytelnienia
- » Warto zmienić konfigurację (żeby **git** nie pytał za każdym razem o dane):

# GIT - konfiguracja

```
> git clone https://gitlab.com/gr/pro.git  
> git config --global user.name "Your name"
```

- » Logowanie, konfiguracja
- » Repozytorium może być niepubliczne, więc “clone” wymaga uwierzytelnienia
- » Warto zmienić konfigurację (żeby **git** nie pytał za każdym razem o dane):
  - imię, nazwisko

# GIT - konfiguracja

```
> git clone https://gitlab.com/gr/pro.git  
> git config --global user.name "Your name"  
> git config --global user.email your@email.com
```

- » Logowanie, konfiguracja
- » Repozytorium może być niepubliczne, więc “clone” wymaga uwierzytelnienia
- » Warto zmienić konfigurację (żeby **git** nie pytał za każdym razem o dane):
  - imię, nazwisko
  - e-mail

# GIT - konfiguracja

```
> git clone https://gitlab.com/gr/pro.git  
> git config --global user.name "Your name"  
> git config --global user.email your@email.com  
> git config --global push.default simple  
> git config --global credential.helper "cache  
--timeout=3600"
```

- » Logowanie, konfiguracja
- » Repozytorium może być niepubliczne, więc “clone” wymaga uwierzytelnienia
- » Warto zmienić konfigurację (żeby **git** nie pytał za każdym razem o dane):
  - imię, nazwisko
  - e-mail
  - konfiguracja metody push (safe)
  - zapamiętywanie hasła przez 1h

# GIT - konfiguracja

```
> git clone https://gitlab.com/gr/pro.git  
> git config --global user.name "Your name"  
> git config --global user.email your@email.com  
> git config --global push.default simple  
> git config --global credential.helper "cache  
--timeout=3600"
```

- » Logowanie, konfiguracja
- » Repozytorium może być niepubliczne, więc “clone” wymaga uwierzytelnienia
- » Warto zmienić konfigurację (żeby **git** nie pytał za każdym razem o dane):
  - imię, nazwisko
  - e-mail
  - konfiguracja metody push (safe)
  - zapamiętywanie hasła przez 1h
- » Konfiguracja jest zapisywana w pliku ~/.gitconfig

# GIT - konfiguracja

```
> git clone https://gitlab.com/gr/pro.git
> git config --global user.name "Your name"
> git config --global user.email your@email.com
> git config --global push.default simple
> git config --global credential.helper "cache
    --timeout=3600"
> cd pro/
> echo "xxx" >text.txt
> git add text.txt
> git commit -m "first version"
> echo "+yyy" >>text.txt
> git commit -m "second version"
> git push
```

- » Logowanie, konfiguracja
- » Repozytorium może być niepubliczne, więc "clone" wymaga uwierzytelnienia
- » Warto zmienić konfigurację (żeby **git** nie pytał za każdym razem o dane):
  - imię, nazwisko
  - e-mail
  - konfiguracja metody push (safe)
  - zapamiętywanie hasła przez 1h
- » Konfiguracja jest zapisywana w pliku ~/.gitconfig

# GIT - konfiguracja

```
> git clone https://gitlab.com/gr/pro.git
> git config --global user.name "Your name"
> git config --global user.email your@email.com
> git config --global push.default simple
> git config --global credential.helper "cache
    --timeout=3600"

> cd pro/
> echo "xxx" >text.txt
> git add text.txt
> git commit -m "first version"
> echo "+yyy" >>text.txt
> git commit -m "second version"
> git push
```

- » Logowanie, konfiguracja
- » Repozytorium może być niepubliczne, więc “clone” wymaga uwierzytelnienia
- » Warto zmienić konfigurację (żeby **git** nie pytał za każdym razem o dane):
  - imię, nazwisko
  - e-mail
  - konfiguracja metody push (safe)
  - zapamiętywanie hasła przez 1h
- » Konfiguracja jest zapisywana w pliku `~/.gitconfig`
- » W praktyce klonowanie i konfigurację wykonuje się tylko raz

# GIT - konfiguracja

```
> git clone https://gitlab.com/gr/pro.git
> git config --global user.name "Your name"
> git config --global user.email your@email.com
> git config --global push.default simple
> git config --global credential.helper "cache
  --timeout=3600"

> cd pro/
> echo "xxx" >text.txt
> git add text.txt
> git commit -m "first version"
> echo "+yyy" >>text.txt
> git commit -m "second version"
> git push
```

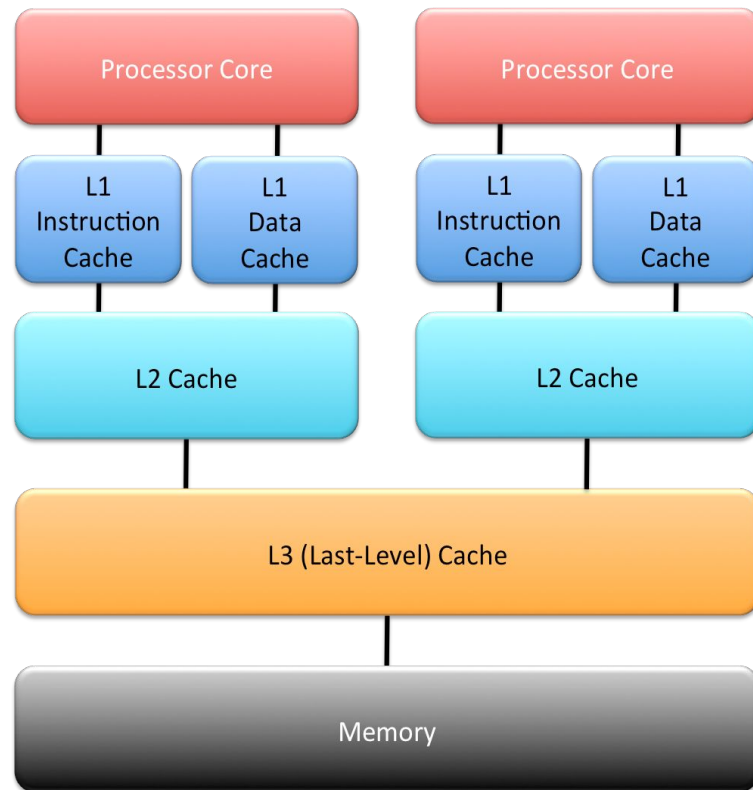
- » Logowanie, konfiguracja
- » Repozytorium może być niepubliczne, więc “clone” wymaga uwierzytelnienia
- » Warto zmienić konfigurację (żeby **git** nie pytał za każdym razem o dane):
  - imię, nazwisko
  - e-mail
  - konfiguracja metody push (safe)
  - zapamiętywanie hasła przez 1h
- » Konfiguracja jest zapisywana w pliku `~/.gitconfig`
- » W praktyce klonowanie i konfigurację wykonuje się tylko raz
- » [W warunkach naszego labu na początku każdego zajęcia !!!](#)





# jest w rzeczywistości?

- » na zewnątrz: von Neumann
- » w środku: “to skomplikowane”
- » RAM (ang. Random Access Memory)  
DDR4-2400, CL15 to: 2.4 GT/s (x64bits),  
**1 bajt po 50 ns, następny po 15 ns.**
- » Zen (AMD-Ryzen):
  - L1: 64 KiB instruction + 32 KiB data
  - L2: 512 KiB (per core)
  - L3: 8 MiB (per CXX quad-core)

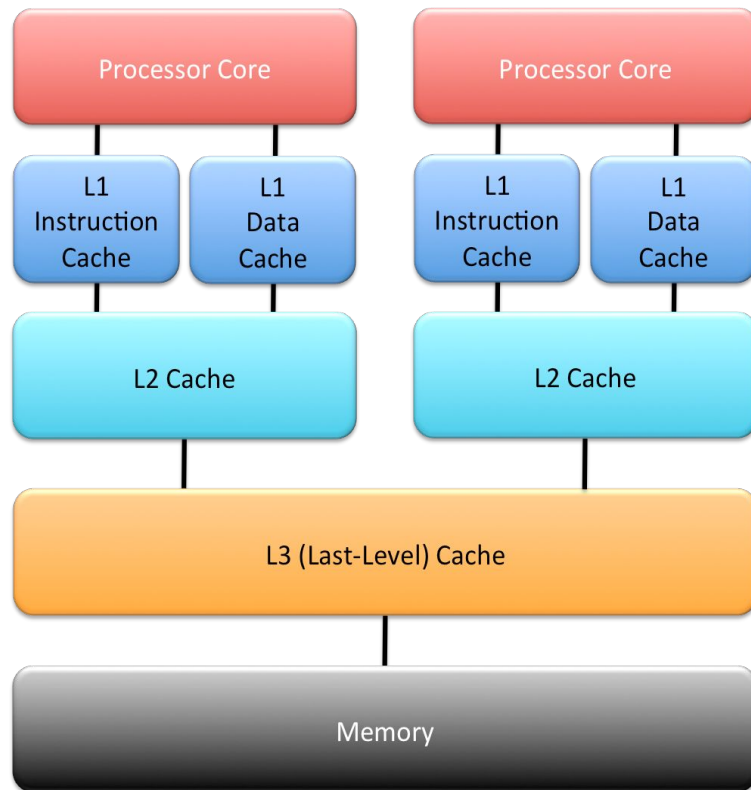


# jest w rzeczywistości?

- » na zewnątrz: von Neumann
- » w środku: “to skomplikowane”
- » RAM (ang. Random Access Memory)  
DDR4-2400, CL15 to: 2.4 GT/s (x64bits),  
**1 bajt po 50 ns, następny po 15 ns.**
- » Zen (AMD-Ryzen):

Ryzen 7 1800X	Lecture (Go/s)	Ecriture (Go/s)	Copie (Go/s)	Latence (ns)
L1	745,63	373,97	737,93	1,3
L2	482,66	338,53	476,62	8,5
L3	171,02	114,65	241,16	46,6

<https://www.techpowerup.com/>



<https://microkerneldude.files.wordpress.com/2015/04/architecture2.png>

# cykl rozkazowy procesora

- » **IF** Instruction Fetch  
*pobranie*
- » **ID** Instruction Decode  
*dekodowanie*
- » **EX** Execute  
*wykonanie*
- » **MEM** Memory access  
*zapis odczyt RAM*
- » **WB** Register write back

Instr No.	Pipeline Stage						
1	IF	ID	EX	MEM	WB		

# cykl rozkazowy procesora

- » **IF** Instruction Fetch  
*pobranie*
- » **ID =** Instruction Decode  
*dekodowanie*
- » **EX** Execute  
*wykonanie*
- » **MEM** Memory access  
*zapis odczyt RAM*
- » **WB** Register write back

Instr No.	Pipeline Stage						
<b>1</b>	IF	ID	EX	MEM	WB		
<b>2</b>		IF	ID	EX	MEM	WB	
<b>3</b>			IF	ID	EX	MEM	WB
<b>4</b>				IF	ID	EX	MEM
<b>5</b>					IF	ID	EX
Clock Cycle	1	2	3	4	5	6	7

[https://en.wikipedia.org/wiki/Instruction\\_pipelining](https://en.wikipedia.org/wiki/Instruction_pipelining)

Dziękuję