

# Podstawy Informatyki

**Katedra Telekomunikacji, EiT**

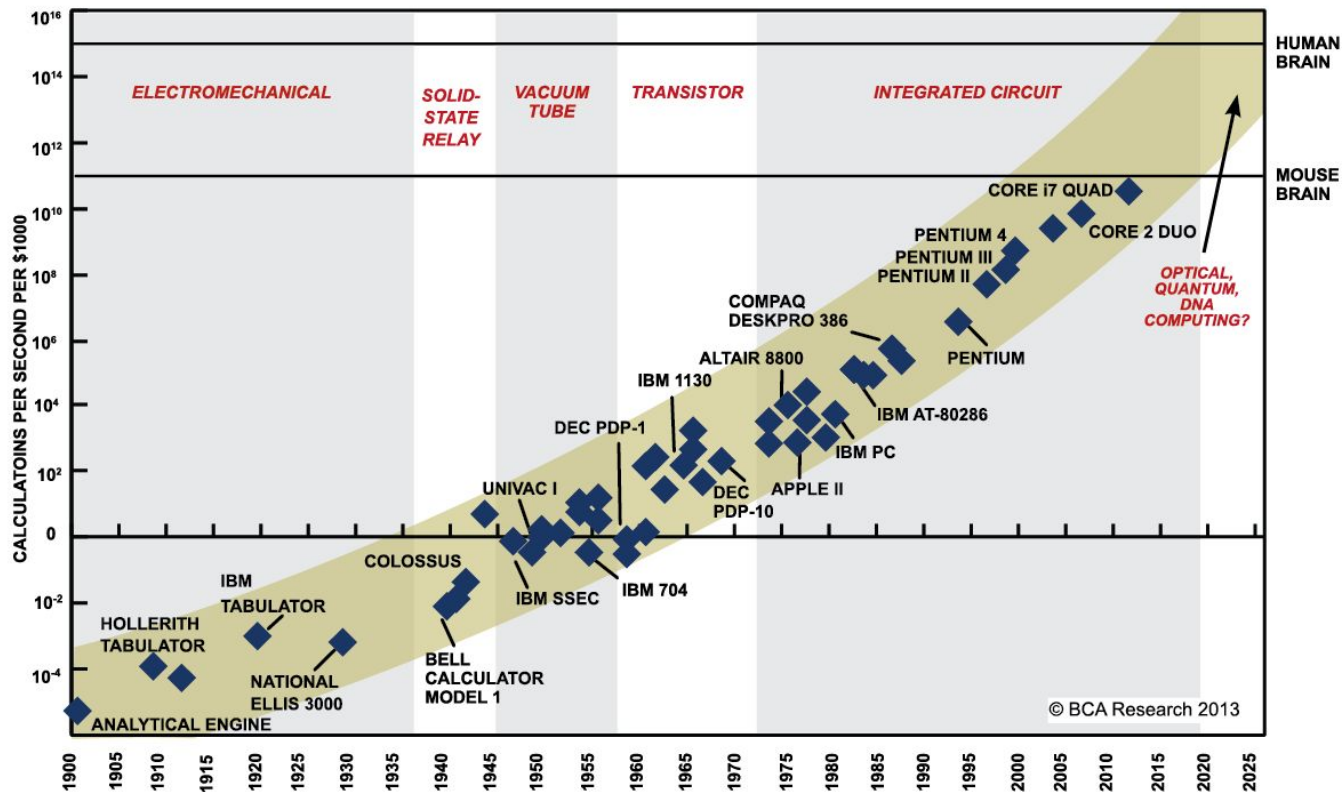
dr inż. Jarosław Bułat

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

# Ciekawe materiały

- » Od czego zależy popularność języka programowania:  
<https://www.youtube.com/watch?v=QyJZzq0v7Z4> (Why Isn't Functional Programming the Norm? – Richard Feldman)
- » Współcześni programiści zbyt często piszą w “ultrawygodnych językach”, zbyt wysoko (w sensie abstrakcji) od sprzętu. Przez to nie czują jakie ograniczenia ma sprzęt.
- » Prawo Moor’a jest martwe (2020), w procesach technologicznych  $\leq 7$  nm nie osiąga się taktowania 5+ GHz ze względu na ograniczenia fizyki
- » Pojedynczy rdzeń/wątek nie będzie szybszy niż jest
- » Wniosek:

**w przyszłości będzie nacisk na optymalizację kodu**



SOURCE: RAY KURZWEIL, "THE SINGULARITY IS NEAR: WHEN HUMANS TRANSCEND BIOLOGY", P.67, THE VIKING PRESS, 2006. DATAPOINTS BETWEEN 2000 AND 2012 REPRESENT BCA ESTIMATES.

# Plan prezentacji

- » Zasięg zmiennych
- » Struktury - złożony typ danych
- » Standardy kodowania
- » Instrukcje warunkowe
- » Instrukcja switch
- » Typ wyliczeniowy
- » Pętle
- » Budowa komputera cd...
- » System operacyjny (Linux), podstawowe pojęcia, ekosystem

# Jak długo żyją zmienne?

# Zasięg zmiennych

» Zasięg jest definiowany przez  
blok {...}

```
int main(){  
    // code  
    // code  
  
    // code  
    // code  
}
```

# Zasięg zmiennych

» Zasięg jest definiowany przez  
blok {...}

```
int main(){  
    // code  
    // code  
    {  
        // code  
        // code  
    }  
    // code  
    // code  
}
```

# Zasięg zmiennych

```
int x = 1, g = 2;

int main(){
    int x = 2;           // x == 1 (l:4)
                        // ok
                        // x == 2 (l:8)
    if (x > 1) {
        x++;             // x == 3 (l:8)
        int x = 4;       // x == 4 (l:12)
        x++;             // x == 5 (l:12)
    }
    cout << x;            // x == 3 (l:8)
    cout << g;            // g == 2 (l:4)

    int x = -1;          // ERROR
                        // redeclaration
}
```

- » Zasięg jest definiowany przez blok {...}
- » Zmienna istnieje od deklaracji do końca pliku albo do zamknięcia bloku
- » Zmienne globalne istnieją od deklaracji do końca pliku



# Zasięg zmiennych

```
int x = 1, g = 2;

int main(){
    // x == 1 (l:4)
    int x = 2;      // ok
                   // x == 2 (l:8)

    if (x > 1) {
        x++;        // x == 3 (l:8)
        int x = 4;  // x == 4 (l:12)
        x++;        // x == 5 (l:12)
    }
    cout << x;      // x == 3 (l:8)
    cout << g;      // g == 2 (l:4)

    int x = -1;     // ERROR
                   // redeclaration
}
```

- » Zasięg jest definiowany przez blok {...}
- » Zmienna istnieje od deklaracji do końca pliku albo do zamknięcia bloku
- » Zmienne globalne istnieją od deklaracji do końca pliku
- » W bloku (zagnieżdżonym) zmienna przykrywa ale nie kasuje zmiennej o tej samej nazwie
- » Błąd gdy powtórna deklaracja zmiennej w jednym bloku

# Zasięg zmiennych

```
#include <iostream>

using namespace std;

int main(){
    int x = 2;

    if (x > 1) {
        int result = x * 2;
    }

    cout << result;
}
```

Zasięg zmiennej od deklaracji do końca bloku (lub pliku)

# Zasięg zmiennych

```
#include <iostream>
```

```
using namespace std;
```

```
int main(){
```

```
    int x = 2;
```

```
    if (x > 1) {
```

```
        int result = x * 2;
```

```
    }
```

```
    cout << result;
```

```
}
```

```
~/D/P/lab_04_struct_condition> g++ ex12.cc
```

```
ex12.cc: In function 'int main()':
```

```
ex12.cc:21:10: error: 'result' was not declared in this scope
```

```
    cout << result;
```

```
          ^~~~~~
```

Zasięg zmiennej od deklaracji do końca bloku (lub pliku)



quiz

**PI04\_scope**

**socrative.com**

- login
- student login

Room name:

**KWANTAGH**

# Jak opisać złożony obiekt?

**wiele parametrów  
różne typy (int/float)**

# struct Name{...};

```
4: struct Product{
5:     int weight;
6:     float price;
7: };
8:
9: int main(){
10:     Product car;
11:     car.weight = 1e6;           // 1000000
12:     car.price = 100000;
13:     Product egg = {1, 0.5};
14:
15:     cout << egg.weight << "\n";
16:     cout << egg.price << "\n";
17:
18:     egg = car;                  // copy
19:     // egg.weight = car.weight;
20:     // egg.price = car.price;
21:
22:     cout << egg.weight << "\n";
23:     cout << egg.price << "\n";
24: }
```

- » Złożony typ danych (agregator elementów różnego typu)
- » **Nowy, własny** typ danych
- » 4-7: definicja typu

# struct Name{...};

```
4: struct Product{
5:     int weight;
6:     float price;
7: };
8:
9: int main(){
10:     Product car;
11:     car.weight = 1e6;           // 1000000
12:     car.price = 100000;
13:     Product egg = {1, 0.5};
14:
15:     cout << egg.weight << "\n";
16:     cout << egg.price << "\n";
17:
18:     egg = car;                  // copy
19:     // egg.weight = car.weight;
20:     // egg.price = car.price;
21:
22:     cout << egg.weight << "\n";
23:     cout << egg.price << "\n";
24: }
```

- » Złożony typ danych (agregator elementów różnego typu)
- » **Nowy, własny** typ danych
- » 4-7: definicja typu
- » 10: deklaracja zmiennej **car** o typie **Product**

# struct Name{...};

```
4: struct Product{
5:     int weight;
6:     float price;
7: };
8:
9: int main(){
10:     Product car;
11:     car.weight = 1e6;           // 1000000
12:     car.price = 100000;
13:     Product egg = {1, 0.5};
14:
15:     cout << egg.weight << "\n";
16:     cout << egg.price << "\n";
17:
18:     egg = car;                  // copy
19:     // egg.weight = car.weight;
20:     // egg.price = car.price;
21:
22:     cout << egg.weight << "\n";
23:     cout << egg.price << "\n";
24: }
```

- » Złożony typ danych (agregator elementów różnego typu)
- » **Nowy, własny** typ danych
- » 4-7: definicja typu
- » 10: deklaracja zmiennej **car** o typie **Product**
- » **Dostęp do elementów** (pól) za pomocą operatora **"."**



# struct Name{...};

```
4: struct Product{
5:     int weight;
6:     float price;
7: };
8:
9: int main(){
10:     Product car;
11:     car.weight = 1e6;           // 1000000
12:     car.price = 100000;
13:     Product egg = {1, 0.5};
14:
15:     cout << egg.weight << "\n";
16:     cout << egg.price << "\n";
17:
18:     egg = car;                  // copy
19:     // egg.weight = car.weight;
20:     // egg.price = car.price;
21:
22:     cout << egg.weight << "\n";
23:     cout << egg.price << "\n";
24: }
```

- » Złożony typ danych (agregator elementów różnego typu)
- » **Nowy, własny** typ danych
- » 4-7: definicja typu
- » 10: deklaracja zmiennej **car** o typie **Product**
- » Dostęp do elementów (pól) za pomocą operatora “.”
- » 13: deklaracja + inicjalizacja
- » 18: kopiowanie
- » 19-20: kopiowanie (nie używać)

# Deklaracja + definicja

```
struct Product{  
    int weight;  
    float price;  
}car,egg;           // global variable  
  
int main(){  
    car.weight = 1e6;           // 1000000  
    car.price = 100000;  
    // car={1,1};           // only from c++11  
    egg = car;  
  
    cout << egg.weight << "\n";  
    cout << egg.price << "\n";  
}
```

- » Deklaracja + definicja
- » Globalny zasięg
- » Przypisanie wszystkich pól struktury tylko podczas inicjalizacji albo w standardzie **c++11**

# typedef struct

```
// C
typedef struct{
    int weight;
    float price;
}Product;

struct X{
    int i;
};
typedef struct X Y;

typedef unsigned int uint;

int main(){
    Product p1;           // ok

    X p2;                 // error in C
    struct X p3;          // ok
    Y p4;                 // ok

    uint u;               // unsigned int
}
```

- » W języku C wymagana jawna definicja typu
- » Przykład definicji własnego typu **uint** w C/C++
- » Zasięg definicji do końca pliku
- » Zazwyczaj definiowany globalnie, często dla kilku plików \*.cc (#include<...>)

# Rozmiar struktury

» **Rozmiar struktury**  
!= sumia składowych

```
using namespace std;

struct Example{
    char x;
    double y;
}ex;

int main(){
    cout << sizeof(Example) << "\n";
    cout << sizeof(ex) << "\n";
    // 16 bytes ?? why not 9 ???
}
```

# Rozmiar struktury

```
using namespace std;
```

```
struct Example{  
    char x;  
    double y;  
}ex;
```

```
int main(){  
    cout << sizeof(Example) << "\n";  
    cout << sizeof(ex) << "\n";  
    // 16 bytes !? why not 9 ???  
}
```

- » Rozmiar struktury  
!= sumia składowych
- » Data structure padding:
  - początek zmiennej
  - długość zmiennej
- » CPU czyta/pisze pamięć w paczkach minimum 32 bitowych
- » Upakowanie danych w strukturach jest nieefektywne obliczeniowo
- » Upakowanie jest możliwe:  
**#pragma pack(1)**

# Zagnieżdżenie struktury

```
struct Product{  
    int weight;  
    struct Price{  
        float us;  
        float eu;  
    }product_price;  
};  
  
int main(){  
    Product car;  
    car.weight = 1e6;  
    car.product_price.us = 60;  
    car.product_price.eu = 50;  
  
    // product_price.eu = 50;  
}
```

- » Struktura może zawierać składowe (prawie) dowolnego typu
- » Struktura w strukturze: definicja + deklaracja
- » Deklaracja jest niezbędna

# Zagnieżdżenie struktury

```
struct Product{  
    int weight;  
    struct Price{  
        float us;  
        float eu;  
    }product_price;  
};  
  
int main(){  
    Product car;  
    car.weight = 1e6;  
    car.product_price.us = 60;  
    car.product_price.eu = 50;  
  
    // product_price.eu = 50; ERROR  
}
```

- » Struktura może zawierać składowe (prawie) dowolnego typu
- » Struktura w strukturze: definicja + deklaracja
- » Deklaracja jest niezbędna
- » **Zasięg Price** tylko w obrębie **Product**

# Jak żyć bez struktur?

```
struct Product{  
    int weight;  
    float price;  
};  
  
int main(){  
    Product egg = {1, 0.5};  
  
    int productWeightEgg = 1e6;  
    float productPriceEgg = 100000;  
  
    int productWeightCar = productPriceEgg;  
    float productPriceCar = productWeightEgg;  
}
```

- » Stare i niskopoziomowe języki mogą nie mieć struktur:
  - Basic
  - Asembler
- » Da się ale musi być bardzo mocno uzasadnienie



# Jak żyć bez struktur?

```
struct Product{  
    int weight;  
    float price;  
};  
  
int main(){  
    Product egg = {1, 0.5};  
  
    int productWeightEgg = 1e6;  
    float productPriceEgg = 100000;  
  
    int productWeightCar = productPriceEgg;  
    float productPriceCar = productWeightEgg;  
}
```

- » Stare i niskopoziomowe języki mogą nie mieć struktur:
  - Basic
  - Asembler
- » Da się ale musi być bardzo mocno uzasadnienie
- » **Jaki błąd zrobiłem w kodzie?**

# Jak żyć bez struktur?

```
struct Product{  
    int weight;  
    float price;  
};  
  
int main(){  
    Product egg = {1, 0.5};  
  
    int productWeightEgg = 1e6;  
    float productPriceEgg = 100000;  
  
    int productWeightCar = productPriceEgg;  
    float productPriceCar = productWeightEgg;  
}
```

- » Stare i niskopoziomowe języki mogą nie mieć struktur:
  - Basic
  - Asembler
- » Da się ale musi być bardzo mocno uzasadnienie
- » Jaki błąd zrobiłem w kodzie?



quiz

**PI04\_struct**

socrative.com

- login
- student login

Room name:

**KWANTAGH**

# Co by było gdyby...

## **czyli instrukcje warunkowe**

# Instrukcja warunkowa - IF ... else

```
using namespace std;

int main(){
    int age;
    cout << "enter you age:";
    cin >> age;

    cout << "your age is:" << age << endl;

    if (age > 20) {
        cout << "it is above 20\n";
    }
}
```

- » Kontrola wykonania programu
- » Możliwość “rozgałęziania”
- » Dowolnie złożone **wyrażenie logiczne**
- » Przy okazji: **cin >> age;**
- » If == instrukcja assemblera
- » **Wcięcia - indentacja (!!!)**

# Instrukcja warunkowa - **IF ... else**

```
int main(){  
    int x = 3;  
  
    if (x > 0) {  
        //...  
    }  
  
    if (x > 0) {  
        //...  
    } else {  
        //...  
    }  
  
    if (x > 3) {  
        //...  
    } else if (x > 0 && x <= 3) {  
        //...  
    } else {  
        //...  
    }  
}
```

- » inaczej: *wyrażenie warunkowe*
- » Kontrola wykonania programu
- » Złożony warunek: **else**
- » Wielokrotny warunek: **else if**
  - warunki sprawdzane kolejno
  - pierwszy spełniony kończy całą złożoną instrukcję
- » Wyrażenia logiczne powinny być rozłączne
- » **Najpierw najczęstsze warunki**

# Instrukcja warunkowa - **IF ... else**

```
int x = 3;
int y = 4;

if (x > 2) {
    if (y > 4) {
        //...
    }
}

if (x > 2 && y > 4) {
    //...
}

if (x) {
    cout << "non zero\n";
}

if (!x) {
    cout << "zero!!\n";
}
```

- » Warunek zagnieżdżony oznacza && i da się uprościć
- » **if(x)** oznacza dla wszystkich wartości oprócz 0
- » **if(!x)** oznacza tylko dla  $x == 0$

# Instrukcja warunkowa - IF ... else

```
int x = 3, y = 4;

if (true) {           // always true
    //...             // debug purpose
} else {              // not in production code
    //...
}

// not recommended
if (x < 3)
    cout << "single instruction\n";

if (x < 3) cout << "... \n";
if (x > 3) cout << "... \n";

if (x)
    if (y < 3) {
        //...
    } else {
        //...
    }
}
```

- » Przykłady **jak nie pisać kodu**
- » **If (true)** nie powinno się znaleźć w kodzie produkcyjnym
- » Nie łączyć instrukcji z warunkiem w jednej linii
- » Zawsze stosować klamery, nawet przy jednej instrukcji



# Operator trójargumentowy

```
int main(){  
    int x = 4, y=0;  
  
    int z = (x > 3) ? (3) : (y -= 1);  
  
    if (x > 3) {  
        z = 3;  
    } else {  
        y -= 1;  
        z = y;  
    }  
}
```

- » W nawiasach po “?” muszą być dwa wyrażenia, musi być wynik (przypisanie do z), nie może być np. `cout<< “...”`;

# Operator trójargumentowy

```
int main(){  
    int x = 4, y=0;  
  
    int z = (x > 3) ? (3) : (y -= 1);  
  
    if (x > 3) {  
        z = 3;  
    } else {  
        y -= 1;  
        z = y;  
    }  
}
```

- » W nawiasach po “?” muszą być dwa wyrażenia, musi być wynik (przypisanie do z), nie może być np. `cout<< “...”`;

# Instrukcja wielokrotnego wyboru

» Wartości wyboru muszą być znane podczas kompilacji !!!

```
int main(){  
    char c = 'a';  
  
    switch (c) {  
        case '0':  
            cout << "0\n";  
            cout << "zero\n";  
            break;  
        case 50:  
            cout << "2\n";  
            break;  
        case 'd':  
        case 'e':  
        case 'f': {  
            cout << "d-f\n";  
            break;  
        }  
        default:  
            cout << "other\n";  
    }  
}
```

# Instrukcja wielokrotnego wyboru

```
int main(){
    char c = 'a';

    switch (c) {
        case '0':
            cout << "0\n";
            cout << "zero\n";
            break;
        case 50:
            cout << "2\n";
            break;
        case 'd':
        case 'e':
        case 'f': {
            cout << "d-f\n";
            break;
        }
        default:
            cout << "other\n";
    }
}
```

- » Wartości wyboru muszą być znane podczas kompilacji !!!
- » Nie można napisać **case x:** gdzie x to zmienna

# Instrukcja wielokrotnego wyboru

```
int main(){
    char c = 'a';

    switch (c) {
        case '0':
            cout << "0\n";
            cout << "zero\n";
            break;
        case 50:
            cout << "2\n";
            break;
        case 'd':
        case 'e':
        case 'f': {
            cout << "d-f\n";
            break;
        }
        default:
            cout << "other\n";
    }
}
```

- » Wartości wyboru muszą być znane podczas kompilacji !!!
- » Nie można napisać **case x:** gdzie x to zmienna
- » Klamerki są opcjonalne
- » Zwrócić uwagę na **break;**
- » Pierwszy “udany case” kończy instrukcję - break przechodzi do końca
- » “default” jest opcjonalne



quiz

**PI04\_if**

**socrative.com**

- login
- student login

Room name:

**KWANTAGH**

Nie rozumiem swojego  
kodu... **why?** !@#\$%^

# Standardy kodowania

- » **Zasady służące ujednoliceniu wyglądu i zachowania kodu źródłowego**
  - łatwiejsza analiza swojego i cudzego kodu
  - mniejsza szansa popełnienia błędu
  - ~~ułatwia~~ ułatwia pracę zespołową
- » Formatowanie kodu
- » Konwencje nazewnicze
- » Komentowanie kodu
- » **Wzorce projektowe** (ang. design patterns)



# Standardy kodowania

- » Google C++ Style Guide

<https://google.github.io/styleguide/cppguide.html>

- » Formatowanie

- » Komentarze

- » Nazwy (konwencje)

- » Funkcje

- » Klasy

- » Zasięg zmiennych

- » Pliki nagłówkowe

- » **Zadanie domowe:** przeczytać C++ Style Guide

```
struct Product{
    int weight;
    float price;
}car,egg;

int main(){
    car.weight = 1e6;
    car.price = 100000;

    if (car.weight > 1e6) {
        cout << "heavy\n";
        egg.weight = 10;
        egg.price = 1;
    } else {
        cout << "small\n";
        egg.weight = 1;
        egg.price = 2;
    }
}
```

```
struct xsfa{
    int w; float c;
}a1,a2;

int main(){
    a1.w = 1e6;
    a1.c = 100000;

    if (a1.w > 1e6)
    {
        cout << "heavy\n";
        a2.w = 10;
        a2.c = 1;
    }else{cout << "small\n";
    a2.w = 1;
    a2.c = 2;
    }
}
```

```
struct Product{
    int weight;
    float price;
}car,egg;

int main(){
    car.weight = 1e6;
    car.price = 100000;

    if (car.weight > 1e6) {
        cout << "heavy\n";
        egg.weight = 10;
        egg.price = 1;
    } else {
        cout << "small\n";
        egg.weight = 1;
        egg.price = 2;
    }
}
```

```
struct xsfa{
    int w; float c;
}a1,a2;

int main(){
    a1.w = 1e6;
    a1.c = 100000;

    if (a1.w > 1e6)
    {
        cout << "heavy\n";
        a2.w = 10;
        a2.c = 1;
    }else{cout << "small\n";
    a2.w = 1;
    a2.c = 2;
    }
}
```

# C/C++ (22+1) vs Python (15)

```
int main(){
    int x = 3;
    int y = 4;

    if (x > 2) {
        if (y > 4) {
            //...
        }
    }

    if (x > 2 && y > 4) {
        cout << "interval\n";
    }

    if (x) {
        cout << "non zero\n";
    }

    if (!x) {
        cout << "zero!!\n";
    }
}
```

```
x = 0
y = 4

if x > 2:
    if y > 4:
        print(x)

if x > 2 and y > 4:
    print('interval')

if x:
    print('non zero')

if x == 0:
    print('zero')
```

# enum - typ wyliczeniowy union

# enum - deklaracja

- » Typ zmiennej, w której wartości są ograniczone do ograniczonego zbioru

```
enum Color{  
    RED = 0,  
    BLUE,  
    GREEN  
};  
  
enum CarCompany{  
    AUDI = 0,  
    BMW = 3,  
    FORD = 4,  
    FIAT = 7  
};  
  
enum State{  
    UNINITIALIZED,  
    INITIALIZED,  
    CONFIGURED,  
    ACTIVE,  
    IDLE,  
    QUITTING  
};  
  
int main(){  
    Color c = GREEN;  
    enum State s = IDLE;  
  
    cout << s << endl;  
}
```

# enum - deklaracja

- » Typ zmiennej, w której wartości są ograniczone do ograniczonego zbioru
- » Szczególnie przydatne do wyliczania właściwości obiektu
- » **Zwiększa czytelność kodu**

```
enum Color{  
    RED = 0,  
    BLUE,  
    GREEN  
};
```

```
enum CarCompany{  
    AUDI = 0,  
    BMW = 3,  
    FORD = 4,  
    FIAT = 7  
};
```

```
enum State{  
    UNINITIALIZED,  
    INITIALIZED,  
    CONFIGURED,  
    ACTIVE,  
    IDLE,  
    QUITTING  
};
```

```
int main(){  
    Color c = GREEN;  
    enum State s = IDLE;  
  
    cout << s << endl;  
}
```

# enum - deklaracja

- » Typ zmiennej, w której wartości są ograniczone do ograniczonego zbioru
- » Szczególnie przydatne do wyliczania właściwości obiektu
- » **Zwiększa czytelność kodu**
- » Często używane w komunikacji m2m
- » Implementowane jako **unsigned int**
- » **Podmienia NAME na wartość całkowitą**
- » Wartości (NAME) wyliczane od 0, chyba że chcemy inaczej
- » Wartości przeważnie dużymi literami

```
enum Color{  
    RED = 0,  
    BLUE,  
    GREEN  
};  
  
enum CarCompany{  
    AUDI = 0,  
    BMW = 3,  
    FORD = 4,  
    FIAT = 7  
};  
  
enum State{  
    UNINITIALIZED,  
    INITIALIZED,  
    CONFIGURED,  
    ACTIVE,  
    IDLE,  
    QUITTING  
};  
  
int main(){  
    Color c = GREEN;  
    enum State s = IDLE;  
  
    cout << s << endl;  
}
```



# enum - deklaracja

- » Często używane razem z “case”
- » wartości (RED, BLUE, ...) są znane podczas kompilacji, mogą być użyte jako “case”
- » Zazwyczaj deklarowane jako zmienne globalne z zasięgiem w wielu plikach (przypadek z użyciem case w komunikacji)

```
enum Color{  
    RED = 0,  
    BLUE = 1,  
    GREEN  
};  
  
int main(){  
  
    Color c = GREEN;  
  
    switch (c) {  
        case RED:  
            cout << "R\n";  
            break;  
        case BLUE:  
            cout << "B\n";  
            break;  
        case GREEN:  
            cout << "G\n";  
            break;  
        default:  
            cout << "UN\n";  
    }  
  
    cout << "Color: " << c;  
}
```



quiz

**PI04\_enum**

**socrative.com**

- login
- student login

Room name:

**KWANTAGH**

# union

- » Coś jak struktura, z jednym polem ale mogąca przybierać różny typ
- » Rzadko używane
- » Substytut zmiennych mogących zmieniać typ w trakcie wykonywania programu
- » Spotyka na niskim poziomie (czyste “C”, sterowniki, etc...)
- » Specyficzne użycie
- » Przeczytaj więcej jeżeli chcesz :)

Mam fantazję wypisać liczby  
całkowite z zakresu 1...1000

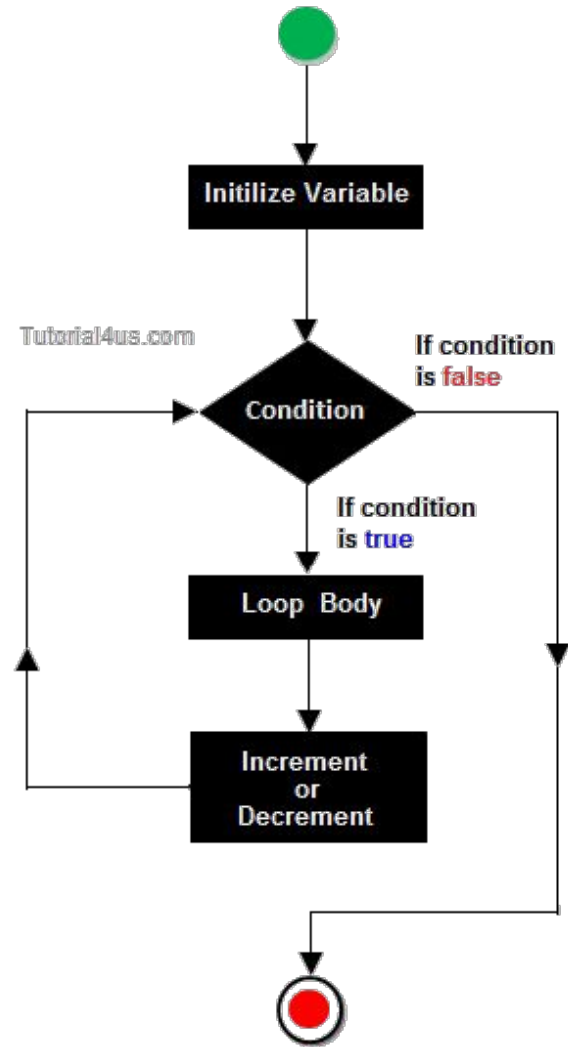
Czy muszę zrobić 1000 “cout” ???

# Pełta for

**for( a = 5; a <= 10; a++ )**

**Initilization**      **Condition**      **Increment (++)  
or  
Decrement (--)**

Tutorial4us.com



# Pętla for

```
int main(){  
    for (int i = 0; i < 100; ++i) {  
        cout << "iterator: " << i << endl;  
    }  
  
    // cout << i;      // error: i out of scope  
}
```

- » **Iteratorów** może być wiele
- » Warunek może być złożony
- » Trzecie wyrażenie może być wielokrotnie
- » **Wykonać 100 razy kod**
- » Wykonać taką samą operację na wszystkich elementach zbioru

# Pętla while

```
int main(){  
  
    int startCounter = 10;  
  
    while (startCounter--){  
        cout << startCounter << ", ";  
    }  
  
    cout << "liftoff!!!" << endl;  
}
```

- » Rezultat: **9, 8, 7, 6, 5, 4, 3, 2, 1, 0, liftoff !!!**
- » Najpierw sprawdzany warunek, potem wykonywana pętla
- » Instrukcje w bloku pętli mogą się nigdy nie wykonać

# Pętla do-while

```
int main(){  
  
    int startCounter = 10;  
  
    do {  
        cout << startCounter << ", ";  
    }while (startCounter--);  
  
    cout << "liftoff!!!" << endl;  
}
```

- » Rezultat: **10, 9, 8, 7, 6, 5, 4, 3, 2, 1, 0, liftoff !!!**
- » Najpierw instrukcje w pętli, potem warunek
- » Instrukcje w bloku wykonają się co najmniej raz



# while vs do-while

```
while (not edge) {  
    run();  
}
```

```
do {  
    run();  
} while (not edge);
```





quiz

**PI04\_for**

**socrative.com**

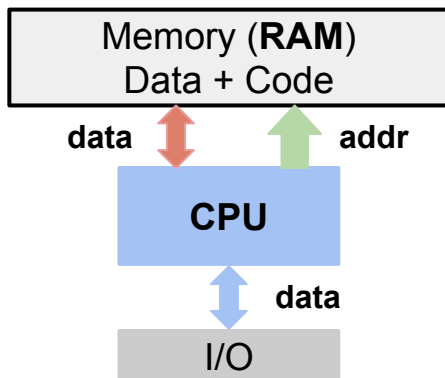
- login
- student login

Room name:

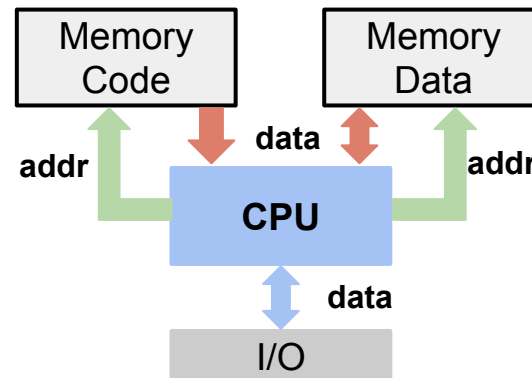
**KWANTAGH**

# Jak jest zbudowany komputer?

# von Neumann vs Harvard



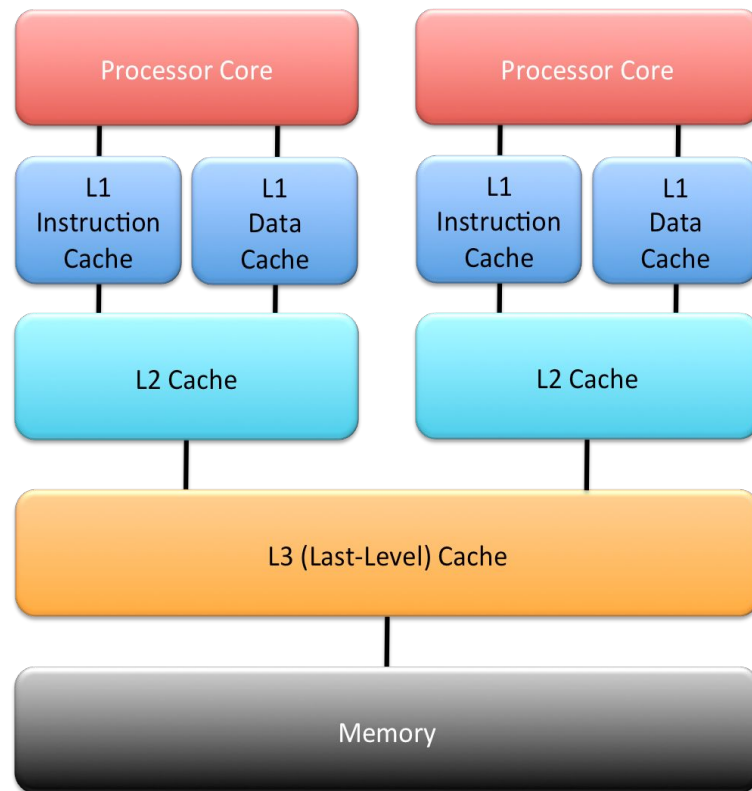
- » Jedna pamięć RAM, jedna magistrala - **taniej**
- » PC, serwery



- » Dwie pamięci, dwie magistrale: równoległy dostęp do danych i instrukcji (**szybciej**)
- » Kod chroniony przed zmianą
- » DSP, uC (krótki program)

# 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)

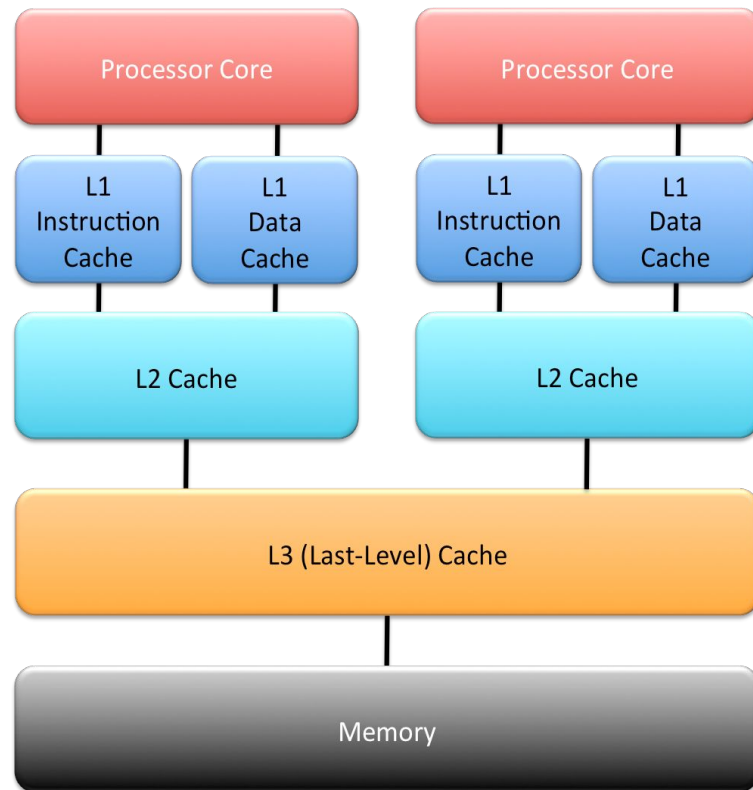


# jest w rzeczywistości?

- » 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)



Czy muszę sam zarządzać  
całym komputerem?

# OS

- » Nie jest niezbędny, można hardcorowo - **bare metal** (bare machine):  
Arudino, IoT, automatyka (oprogramowanie falowników, etc...)
- » **Jest interfejsem pomiędzy człowiekiem a sprzętem**
- » wiki: oprogramowanie zarządzające komputerem, tworzy środowisko do uruchamiania i kontroli zadań użytkownika:
  - przydzielanie czasu procesora zadaniom,
  - przydzielanie pamięci operacyjnej (RAM) zadaniom,
  - synchronizacja pomiędzy zadaniami (IPC),
  - obsługa sprzętu (np. dostęp do HDD przez dwa procesy)

# Funkcje OS

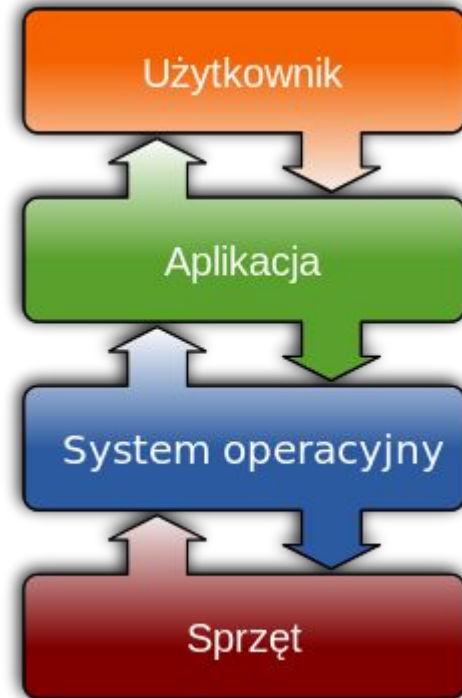
- » Zarządza zadaniami (powołuje, kończy)
- » Zarządza akcjami (IPC, przerwania, eventy, ...)
- » Zarządza zasobami (prawa dostępu, czas dostępu, ograniczenie ilości)
- » Umożliwia komunikację z użytkownikiem

# Proces

- » Instancja (egzemplarz) wykonywanego programu
- » Aplikacja może mieć wiele procesów (zrównoleglenie pracy)
- » Proces może mieć wiele wątków
- » Każdy proces ma identyfikator (PID)
- » OS przyznaje procesowi zasoby (RAM, CPU, ...)
- » Proces może powołać nowy proces

# Interfejs człowiek-maszyna

- » Zadania użytkownika (aplikacje) nie komunikują się bezpośrednio ze sprzętem
- » OS pośredniczy w komunikacji
- » OS kontroluje dostęp do zasobów
  - prawa dostępu
  - czas dostępu
- » OS kolejkuje zadania
  - priorytety
  - optymalizacja
- » OS jest Bogiem ;-) ma władzę nad życiem i śmiercią aplikacji



# Cechy OS

- » Wielodostęp
- » Wielozadaniowość
- » Wieloprocusowość
- » Wielowątkowość
- » Wywłaszczalność

# wielozadaniowość

- » Cecha systemu pozwalająca “równoczesne” działanie wielu procesom
- » I/O (dysk, klawiatura, sieć) jest znacznie wolniejszy niż CPU
  - uruchom dwa programy równocześnie,
    - #1 czeka na I/O
    - #2 korzysta z CPU
  - OS musi zapewnić brak kolizji zasobów (np. #1, #2 chcą 100% CPU)
- » Scheduler (pl. *dyspozytor*), planowanie, priorytety, wieloprocessorowość
- » RAM - dobro rzadkie
- » Wiele CPU, jeden kernel (Linux: 4096)

# Wielodostęp

- » Współdzielenie jednego komputera przez wielu użytkowników
- » Zdalne logowanie - mainframe
- » Prawa dostępu, zagwarantowanie separacji:
  - pamięci operacyjnej
  - pamięci masowej



# Ochrona i zarządzanie pamięcią

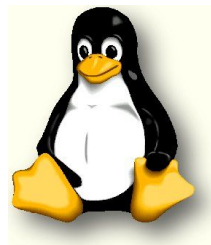
- » Wydzielenie procesom obszarów pamięci
  - próba odczytu innego obszaru kończy się przerwaniem procesu
- » Wsparcie sprzętowe - MMU
- » Ochrona pamięci uniemożliwia odczyt/nadpisanie chronionej pamięci innego procesu
- » Po co chronić RAM?
  - hasła
  - klucze
  - przechwycenie procesu (kradzież tożsamości)
  - możliwość awarii systemu (wirusy)

# Linux

- » **1964** – początki systemu **Multics** (*Multiplexed Information and Computing Service*)
- » **1969** – pierwszy system **Unix** napisana w asemblerze w ośrodku Bell Labs AT&T
- » **1973** – przepisanie kodu **Unixa** na język C przez D. Ritchie i B. Kernighana
- » **Lata 80** – rozwój technologii: **TCP/IP** (*Transmission Control Protocol / Internet Protocol*), **GNU OS** (*GNU is Not Unix*), **POSIX** (*Portable Operating System Interface*)
- » **1991** – Linus Torvalds, fiński student, tworzy jądro systemu operacyjnego **Linux**
- » **1991-...** – powstanie i rozwój wielu odmian Linuxa, społeczność Open Source
- » **2008** – **Android**: (Linux jako “firmware”)

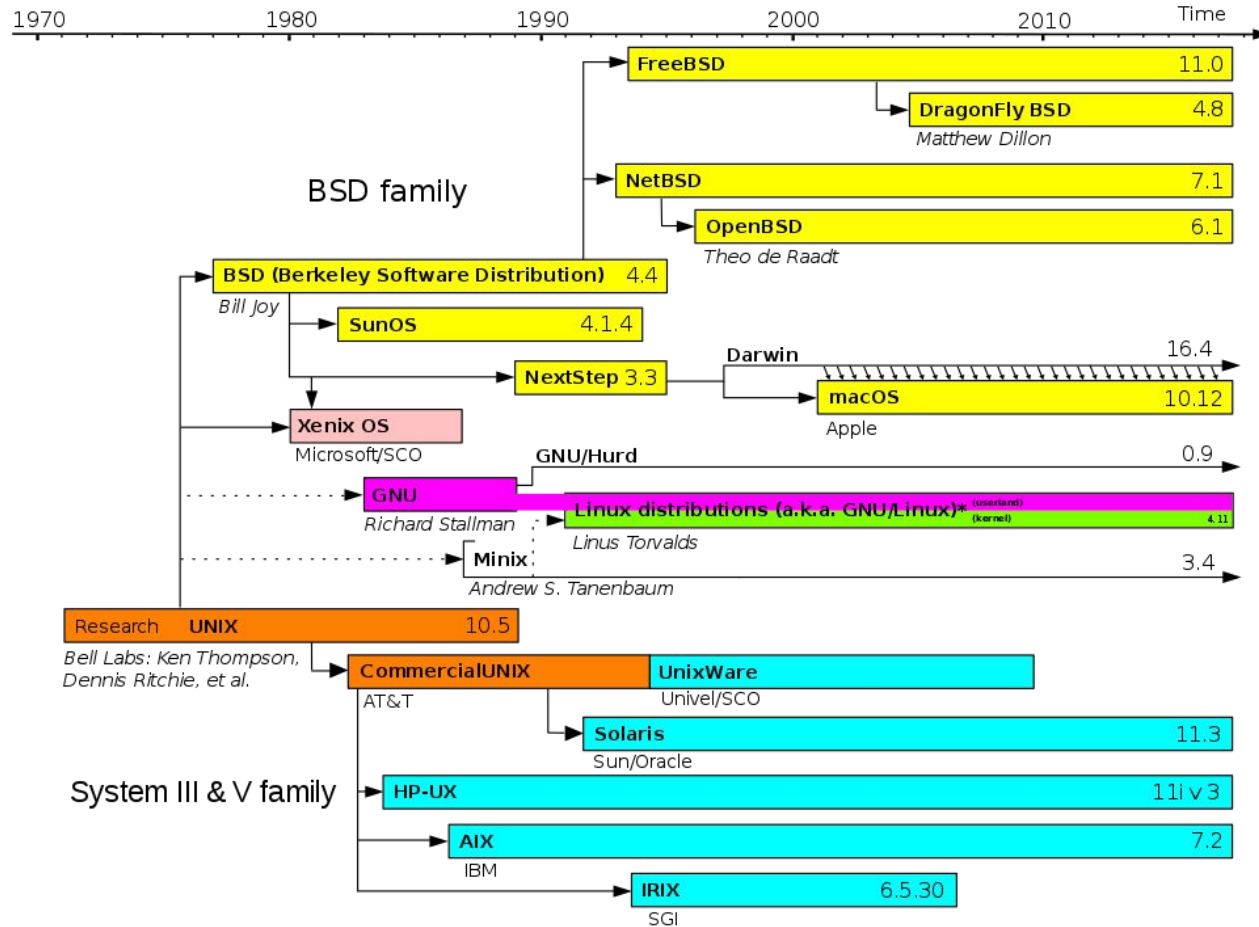
## Główne cechy sytemu:

- » Wielozadaniowość, wielodostępność
- » Skalowalność (smartwatch...TOP500)
- » Stabilność
- » Otwartość (możliwość analizy/zmiany)



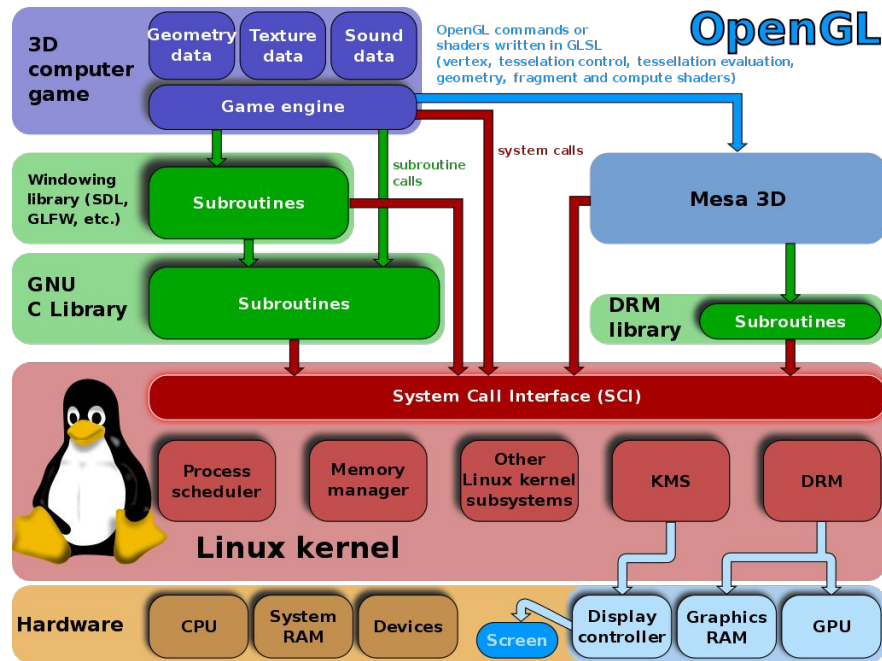
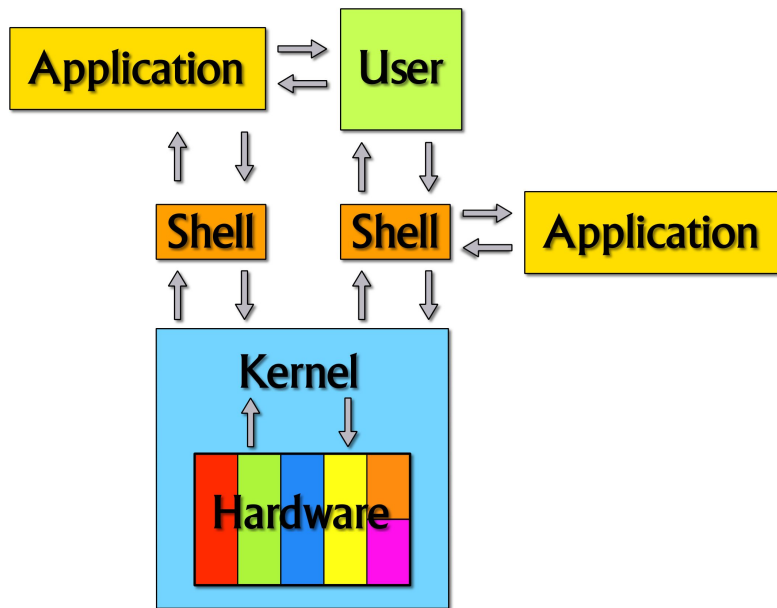
ubuntu





\*The penetration of GNU utilities varies between distributions, some projects use GNU's implementation of the Linux kernel (Linux-libre). Some operating systems mentioned here include GNU utilities to a lesser degree.

# Linux - architektura



# Dlaczego Linux?

- » Jest wszędzie (zwłaszcza w sieciowych urządzeniach)
- » Staje się standardem przemysłowym
- » Nie ma problemu z Vendor lock-in
- » Można głęboko analizować kod kernela
- » Można modyfikować kod źródłowy (jeszcze lepsze jest \*BSD)
- » Przyjemna platforma developerska
- » Można dystrybuować zmieniony system
  
- » Rekomenduję Linuxa jako podstawowy OS na tych zajęciach

# Dystrybucja Linuxa

- » A complete operating system:
- » Linux kernel
- » GNU tools, libraries (eg. Glibc)
- » various "extras"
- » windows system, window manager, desktop (Gnome, KDE, ...)
- » **package manager** (like Android market/play)
  - repository of **tested and validated** package
  - signed, authenticated, repeatedly checked
  - installation/update/delete application with dependencies
  - 20000-30000 packages (from simple library to complex systems)
  - "Seamless" upgrade (typically 6 cycles MSC)

# Jak komunikować się z OS-em?

# SHELL

- » Program komputerowy będący pośrednikiem pomiędzy OS-em a aplikacjami lub użytkownikiem
- » Powłoka, terminal, konsola, CLI (ang. Command Line Interface)
- » Historycznie pierwszy sposób komunikacji z komputerem (potężne narzędzie więc powszechnie używane do dzisiaj)
- » **Program:** sh, bash, tcsh, **fish**
- » “standardowe wyjście” (stdio/stderr) dla programów graficznych (głównie błędy, informacje, etc...)
- » **Praca zdalna**
- » MS DOS to też była powłoka (XP+ udostępnia **powershell**)



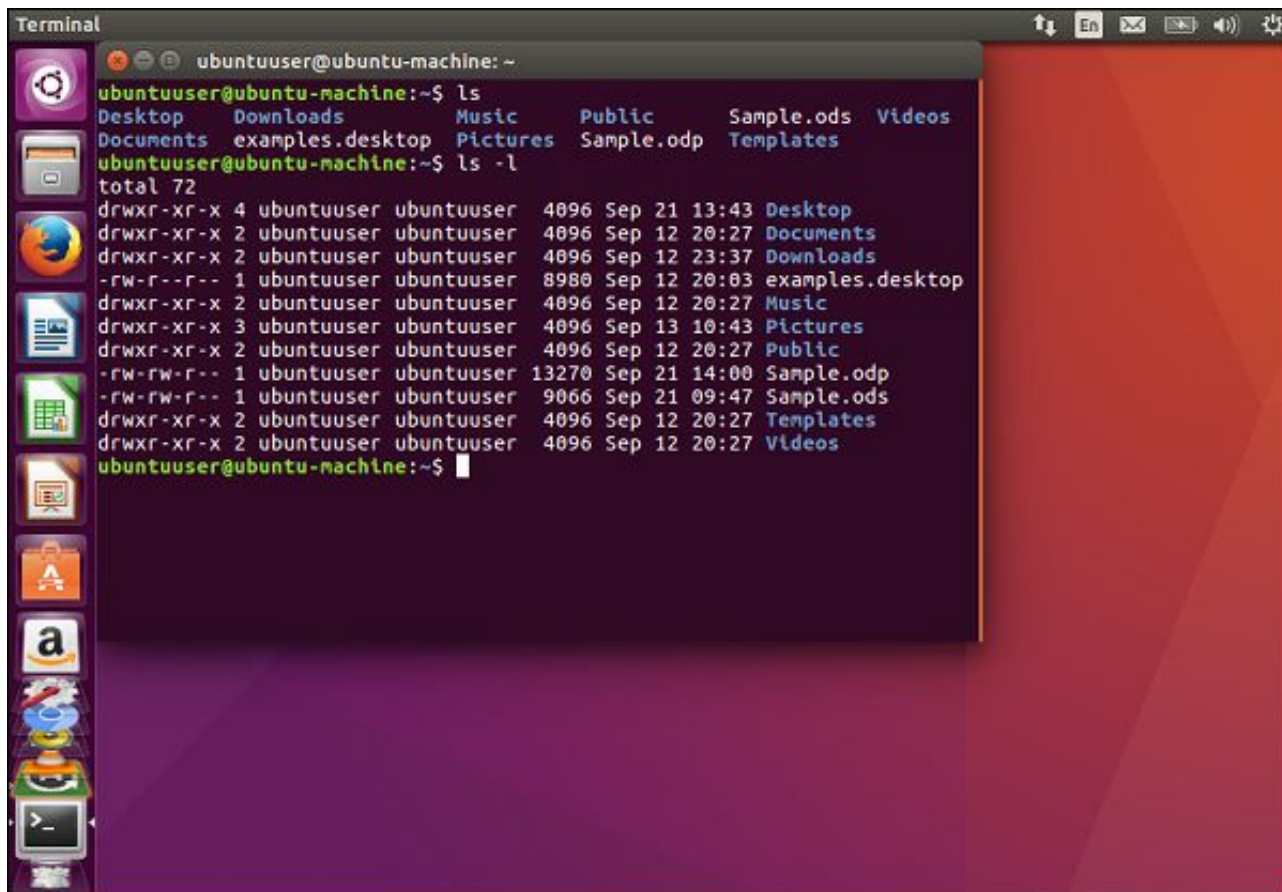
# SHELL

- » wbudowane polecenia np. **cd** (inaczej działają pod bash/tcsh)
- » zewnętrzne polecenia np. **cp**, **mv** (**whereis mv**)
- » wbudowane polecenia np. **pwd** przykrywają systemowe: **pwd --help** wyświetli co innego niż **/bin/pwd --help**)
- » sh/bash/fish to też język skryptowy, w którym można programować:

```
for var in LANG LANGUAGE LC_ALL LC_CTYPE; do
    value=`egrep "^${var}=" "$ENV_FILE" | tail -n1 | cut -d= -f2`
    [ -n "$value" ] && eval export $var=$value

    if [ -n "$value" ] && [ "$ENV_FILE" = /etc/environment ]; then
        log_warning_msg "/etc/environment has been deprecated for locale information; use /etc/default/locale"
    fi
done
```

# \*nix SHELL



The image shows a terminal window on an Ubuntu desktop. The terminal displays the output of the 'ls' and 'ls -l' commands. The desktop background is a red and purple gradient. The left sidebar contains icons for various applications including Dash, Home, Firefox, LibreOffice, and the Dash to Dock extension.

```
Terminal
ubuntuuser@ubuntu-machine: ~
ubuntuuser@ubuntu-machine:~$ ls
Desktop    Downloads  Music      Public    Sample.ods Videos
Documents  examples.desktop Pictures    Sample.odp Templates
ubuntuuser@ubuntu-machine:~$ ls -l
total 72
drwxr-xr-x 4 ubuntuuser ubuntuuser 4096 Sep 21 13:43 Desktop
drwxr-xr-x 2 ubuntuuser ubuntuuser 4096 Sep 12 20:27 Documents
drwxr-xr-x 2 ubuntuuser ubuntuuser 4096 Sep 12 23:37 Downloads
-rw-r--r-- 1 ubuntuuser ubuntuuser 8980 Sep 12 20:03 examples.desktop
drwxr-xr-x 2 ubuntuuser ubuntuuser 4096 Sep 12 20:27 Music
drwxr-xr-x 3 ubuntuuser ubuntuuser 4096 Sep 13 10:43 Pictures
drwxr-xr-x 2 ubuntuuser ubuntuuser 4096 Sep 12 20:27 Public
-rw-rw-r-- 1 ubuntuuser ubuntuuser 13270 Sep 21 14:00 Sample.odp
-rw-rw-r-- 1 ubuntuuser ubuntuuser 9066 Sep 21 09:47 Sample.ods
drwxr-xr-x 2 ubuntuuser ubuntuuser 4096 Sep 12 20:27 Templates
drwxr-xr-x 2 ubuntuuser ubuntuuser 4096 Sep 12 20:27 Videos
ubuntuuser@ubuntu-machine:~$
```

# SHELL

```

Applications  Places  Thu 18:33  13.5 °C

0 [||| 3.4% 4 [ 0.0%]
1 [| 1.0% 5 [| 2.4%]
2 [| 0.5% 6 [ 0.0%]
3 [ 0.0% 7 [ 0.0%]

Mem[||||||||||||||||| 3839/16000MB] Tasks: 105, 302 thr; 1 running
Load average: 0.14 0.15 0.14
Uptime: 2 days, 06:48:04

PID USER PRI NI VIRT RES SHR S CPU% MEM% TIME+ Command
14627 bean 20 0 966M 115M 72596 S 0.0 0.7 0:00.00 gnome-control-center --overview
14626 bean 20 0 966M 115M 72596 S 0.0 0.7 0:00.01 gnome-control-center --overview
14600 bean 20 0 862M 86652 45316 S 0.0 0.5 0:06.23 python2 -m guake.main
14794 bean 20 0 36676 5188 3680 S 0.0 0.0 0:00.01 /bin/zsh
14614 bean 20 0 862M 86652 45316 S 0.0 0.5 0:00.00 python2 -m guake.main
14606 bean 20 0 862M 86652 45316 S 0.0 0.5 0:00.00 python2 -m guake.main
14605 bean 20 0 36676 5104 3584 S 0.0 0.0 0:00.02 /bin/zsh
14706 bean 20 0 19776 2996 2700 S 0.0 0.0 0:00.00 tmux

F1Help F2Setup F3Search F4Filter F5Tree F6SortBy F7Nice -F8Nice +F9Kill F10Quit

[124308.723318] ata4.00: configured for UDMA/133
[124308.894640] wlp10s0u1u4: send auth to d8:50:e6:93:b2:78 (try 1/3)
[124308.896433] wlp10s0u1u4: authenticated
[124308.896526] ath9k_htc 5-1.4:1.0 wlp10s0u1u4: disabling HT as WMM/QoS is not supported by the AP
[124308.896528] ath9k_htc 5-1.4:1.0 wlp10s0u1u4: disabling VHT as WMM/QoS is not supported by the AP
[124308.899240] wlp10s0u1u4: associate with d8:50:e6:93:b2:78 (try 1/3)
[124308.901660] wlp10s0u1u4: RX AssocResp from d8:50:e6:93:b2:78 (capab=0x411 status=0 aid=8)
[124308.908363] wlp10s0u1u4: associated
[124308.908386] IPv6: ADDRCONF(NETDEV_CHANGE): wlp10s0u1u4: link becomes ready
[126535.858744] snd_hda_codec_hdmi hdaudioC1D0: HDMI: invalid ELD data byte 1

bean@bean-desktop ~
[18:33:36]

GNU nano 2.4.2 File: ...iliora-Secunda/gnome-shell/gnome-shell.css

===== */
.extension-dialog .modal-dialog-button:focus {
border-image: url("button-assets/button-violet.svg") 10;
}
.extension-dialog .modal-dialog-button:focus:active {
border-image: url("button-assets/button-violet-hover.svg") 10;
}
.extension-dialog .modal-dialog-button:focus:pressed {
}

^G Get Help ^O Write Out ^W Where Is ^K Cut Text ^J Justify
^X Exit ^R Read File ^\ Replace ^U Uncut Text ^T To Spell

[2] 0:htop* "screen" 18:33 10-Sep-15
  
```

# SHELL - how to...

## Ogólny schemat poleceń:

```
user@host: dir $ nazwa_polecenia [opcje, argumenty, ...] <enter>
```

Opcje literowe poprzedza się znakiem „-”, a słowne znakami „--”: np.: `ls -al`

Każde polecenie posiada zwykle pomoc: „-h” lub „--help”

Dodatkowe informacje dostarcza manual: `$ man nazwa`

## Praca zdalna:

```
ssh moj_nick@student.agh.edu.pl
```

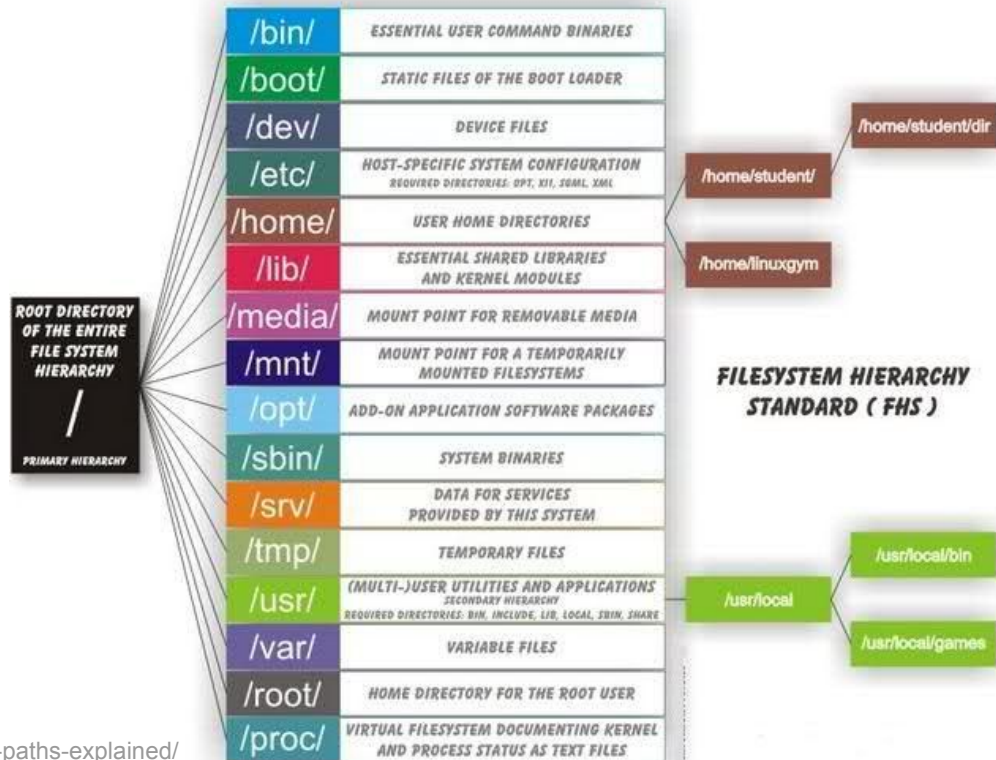
i tutaj mamy konsolę tylko na zdalnej maszynie

## Interaktywny samouczek:

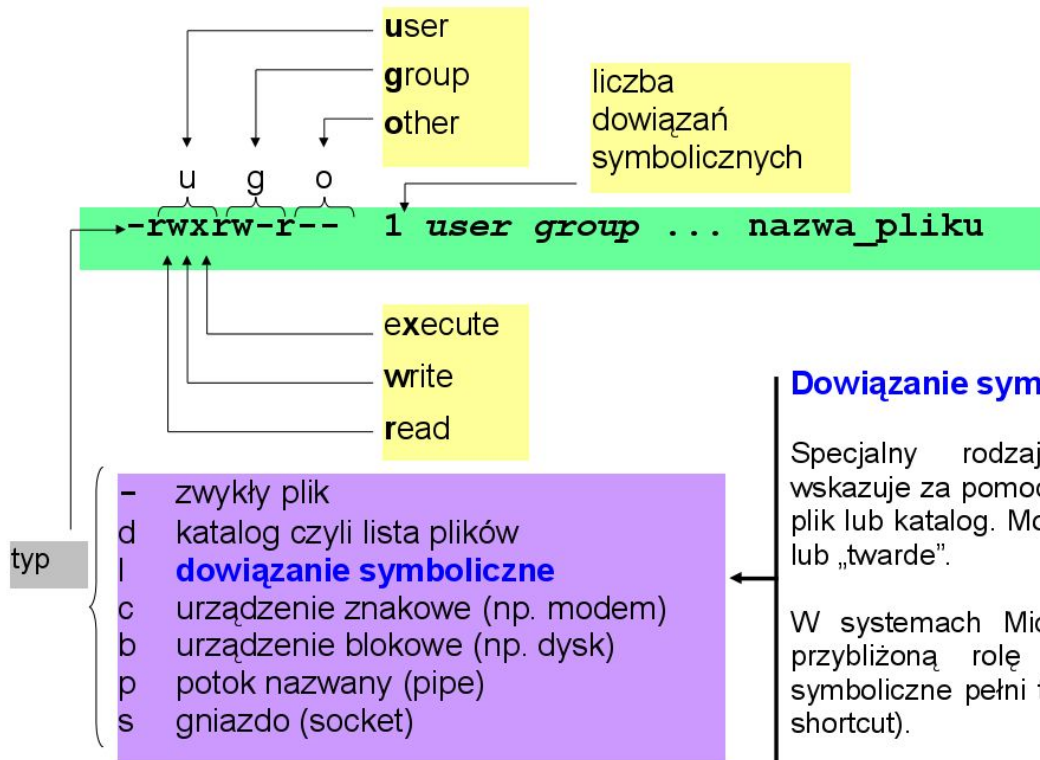
<https://www.learnshell.org/>

# Struktura systemu plików

- » FHS (Filesystem Hierarchy Standard)
- » Punkt montowania - dowolny pusty katalog
- » “rozszerzenie pliku” to dekoracja nie informacja



# Atrybuty plików (metadane)



## Dowiązanie symboliczne

Specjalny rodzaj pliku, który wskazuje za pomocą nazwy na inny plik lub katalog. Może być „miękkie” lub „twarde”.

W systemach Microsoft Windows przybliżoną rolę co dowiązanie symboliczne pełni tzw. „skrót” (ang. shortcut).



# Atrybuty plików (metadane)

Polecenia do zmiany praw dostępu:

- chmod – Polecenie zmiany zezwoleń systemowych do pliku
- chown – Polecenie zmiany właściciela pliku
- chgrp – Polecenie zmiany przypisania pliku do grupy

Sposób 1 nadawania praw (symboliczny):

`$ chmod [kto] operator [zezwolenie][,...] nazwa_pliku`

**kto:**

a wszyscy  
u użytkownik  
g grupa  
o inni

**operator:**

+ dodaj zezwolenie  
- odbierz zezwolenie  
= zastąp zezwolenie

**zezwolenie:**

r odczyt  
w zapis  
x wykonanie

**Przykłady:**

```
$ chmod a+w plik1
$ chmod u-w plik2
$ chmod u=rw,o=r plik3
```

Sposób 2 nadawania praw (ósemkowy):

`$ chmod kod_ósemkowy nazwa_pliku`

**kod\_ósemkowy** – suma kodów ósemkowych w grupach:

user	r=400 w=200 x=100
group	r=040 w=020 x=010
others	r=004 w=002 x=001

**Przykłady:**

**Wyniki:**

```
$ chmod 777 plik1 -rwrwrw
$ chmod 641 plik2 -rw-r---x
$ chmod 555 plik3 -r-xr-x-x
```

# SHELL - ToDo

- » Zmienne środowiskowe
- » Praca zdalna
- » Manipulacja plikami
- » Manipulacja tekstem
- » Strumienie we/wy (w szczególności stdio/stderr)
- » Skrypty
  - instrukcje warunkowe
  - pętle
  - manipulacja plikami



Dziękuję