Methodology and Programming Techniques

Department of Telecommunications, IEiT

dr inż. Jarosław Bułat

kwant@agh.edu.pl

Outline

» Scope of variables + block of code

» Structures in C/C++

» Coding standards

» Conditional statements

» Switch statement

» Enum type

» Loop statements

» Computer architecture (continuation)

» Linux - operating system, basic information, ecosystem

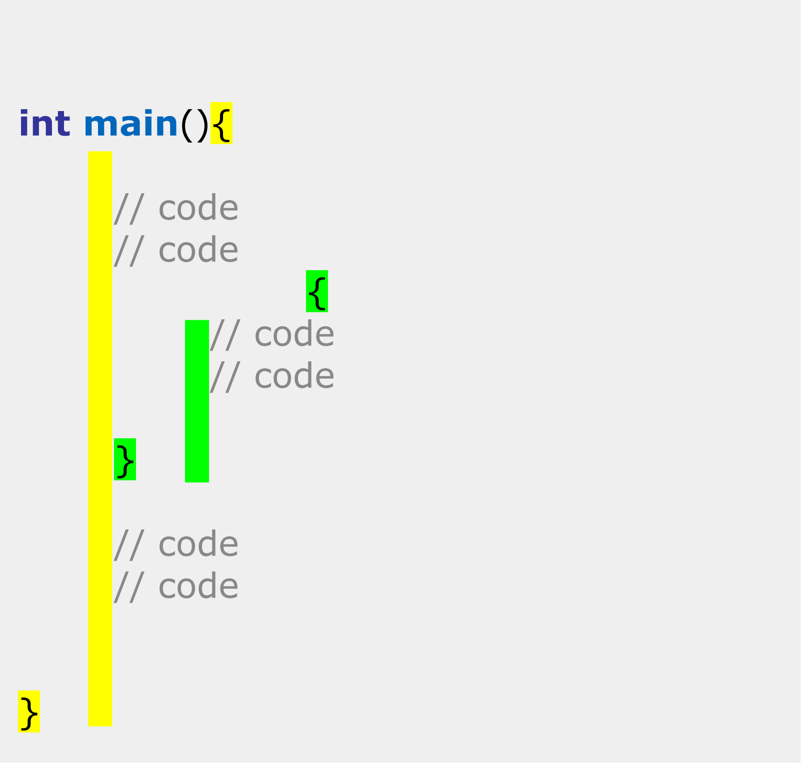
How long do variables live?

Scope of variables

```
int main(){

    // code
    // code








    // code
    // code


}
```

» Range is defined by block {...}

Scope of variables



```
int main(){

     // code
     // code
               {
          // code
          // code

     }

     // code
     // code


}
```

- » Range is defined by block {...}

Scope of variables

```
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
}
```

- Range is defined by block {...}
- The variable exists from the declaration to the end of the file or to the end the block
- Global variables exist from the declaration to the end of the file

Scope of variables

```
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
}
```

- Range is defined by block {...}
- The variable exists from the declaration to the end of the file or to the end the block
- Global variables exist from the declaration to the end of the file
- In a (nested) block, the variable covers but does not erase an already declared variable of the same name
- Error when re-declaring a variable in one block

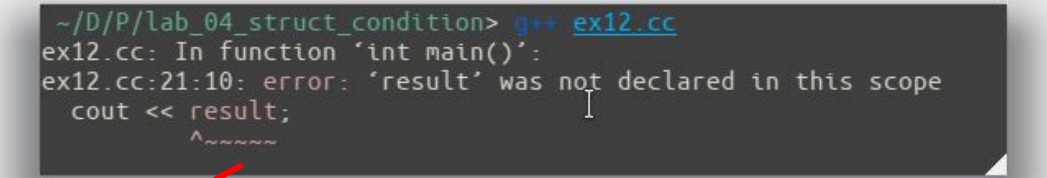
Scope of variables

```
#include <iostream>

using namespace std;

int main(){
	int x = 2;

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

	cout << result;
}
```

Range of variable from declaration to end of block (or file)

Scope of variables

```
#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;
         ^~~~~~
```

Range of variable from declaration to end of block (or file)

How to describe complex object?

many parameters

various type (int/float)

Struct Name{...};

```
struct Product{
        int weight;
        float price;
};

int main(){
        Product car;
        car.weight = 1e6;        // 1000000
        car.price = 100000;
        Product egg = {1, 0.5};

        cout << egg.weight <<"\n";
        cout << egg.price <<"\n";

        egg = car;                        // copy
        // egg.weight = car.weight;
        // egg.price = car.price;

        cout << egg.weight <<"\n";
        cout << egg.price <<"\n";
}
```

- Complex data type (aggregator of various types)
- **New, custom** data type
- 4-7: definition of typ

Struct Name{...};

```
struct Product{
        int weight;
        float price;
};

int main(){
        Product car;
        car.weight = 1e6;          // 1000000
        car.price = 100000;
        Product egg = {1, 0.5};

        cout << egg.weight <<"\n";
        cout << egg.price <<"\n";

        egg = car;                          // copy
        // egg.weight = car.weight;
        // egg.price = car.price;

        cout << egg.weight <<"\n";
        cout << egg.price <<"\n";
}
```

- Complex data type (aggregator of various types)
- **New, custom** data type
- 4-7: definition of type
- 10: declaration of the variable **car**, of the **Product** type

Struct Name{...};

```
struct Product{
        int weight;
        float price;
};

int main(){
        Product car;
        car.weight = 1e6;          // 1000000
        car.price = 100000;
        Product egg = {1, 0.5};

        cout << egg.weight <<"\n";
        cout << egg.price <<"\n";

        egg = car;                              // copy
        // egg.weight = car.weight;
        // egg.price = car.price;

        cout << egg.weight <<"\n";
        cout << egg.price <<"\n";
}
```

- Complex data type (aggregator of various types)
- **New, custom** data type
- 4-7: definition of type
- 10: declaration of the variable **car**, of the **Product** type
- Accessing elements (fields) with the operator "."

Struct Name{...};

```
struct Product{
        int weight;
        float price;
};

int main(){
        Product car;
        car.weight = 1e6;          // 1000000
        car.price = 100000;
        Product egg = {1, 0.5};

        cout << egg.weight <<"\n";
        cout << egg.price <<"\n";

        egg = car;                            // copy
        // egg.weight = car.weight;
        // egg.price = car.price;

        cout << egg.weight <<"\n";
        cout << egg.price <<"\n";
}
```

- » Complex data type (aggregator of various types)
- » **New, custom** data type
- » 4-7: definition of type
- » 10: declaration of the variable **car**, of the **Product** type
- » Accessing elements (fields) with the operator "."
- » 13: declaration + initialization
- » 18: copying
- » 19-20: copy element by element (**do not use**)

Declaration + definition

```
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";
}
```

- Declaration + definition
- **car** and **egg** have global scope
- Assign all the fields of the structure only during initialization or in **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
}
```

- In C, an explicit type definition is required
- An example of a custom **uint** in C/C++
- Scope of a new type: to end of file
- Typically defined globally, often for several *.cc files (#include <...>)

Structure size

```
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 ???
}
```

» **Size of the structure != Sum of the components**

Structure size

```
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 ???
}
```

» **Size of the structure != Sum of the components**

» Data structure padding:
  – start of variable
  – variable length

» CPU read/write memory in minimum 32-bit words

» Data compression in structures is not computationally efficient

» Package is possible: **#pragma pack(1)**

Nesting structure

```
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;
}
```

- The structure can contain almost any type of components
- Structure in structure: definition + declaration
- Declaration is essential **Price** range only within the **Product**

Nesting structure

```
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
}
```

- » The structure can contain almost any type of components
- » Structure in structure: definition + declaration
- » Declaration is essential
  Price range only within the **Product**

How to live without structures?

```
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;
}
```

- Old and low-level languages m have no structures:
  - Basic
  - Assembler
- Programming without structures is possible
  - difficult
  - must be very strong justification

How to live without structures?

```
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;
}
```

- Old and low-level languages m have no structures:
  - Basic
  - Assembler
- Programming without structures is possible
  - difficult
  - must be very strong justification
- What error did I make in the code?

How to live without structures?

```
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;
}
```

» Old and low-level languages m have no structures:
  - Basic
  - Assembler

» Programming without structures is possible
  - difficult
  - must be very strong justification

» What error did I make in the code?

What if ...

conditional instructions

Conditional statement - **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";
    }
}
```

- Control the flow of program
- The possibility of "branching"
- An arbitrary complex **logical expression**

- Standard input: **cin >> age;**
- If == an assembler (single) instruction
- **Indentation - code formatting (!!!)**

Conditional statement - **IF … else**

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

        if (x > 0) {
                //...
        }

        if (x > 0) {
                //...
        } else {
                //...
        }

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

- otherwise: *conditional expression*
- Control the flow of program execution
- Complex condition: **else**
- Multiple condition: **else if**
  - conditions checked successively
  - first fulfilled condition ends whole complex instruction
- Logical expressions should be disjoint
- **Most common conditions**

Conditional statement - **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";
}
```

- Nested condition means && and can be simplified
- **if(x)** means: all values except 0
- **if(! x)** denotes only when x==0

Conditional statement - **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 {
                //...
        }
```

- Examples of **how not to write conditions**
- **If (true)** should not be found in the production code
- Do not combine instructions with the condition in one line
- Always use curly braces, even with one instruction

Conditional operator (Ternary Operator)

```
int main(){
	int x = 4, y=0;

	int z = (x > 3) ? (3) : (y -= 1);

	if (x > 3) {
		z = 3;
	} else {
		y -= 1;
		z = y;
	}
}
```

- There are two expressions after “?”.
  - If first condition is fulfilled first is written as an output
  - In other case, second is thread as an output
- It have to be mathematical expression not “any code”

www.agh.edu.pl

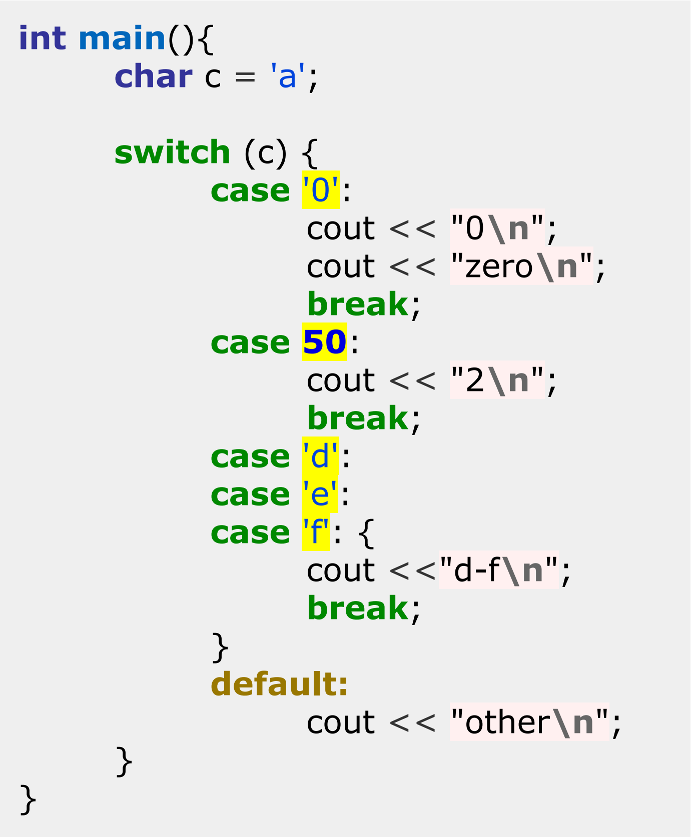
Conditional operator (Ternary Operator)

```
int main(){
        int x = 4, y=0;

        int z = (x > 3) ? (3) : (y -= 1);

        if (x > 3) {
                z = 3;
        } else {
                y -= 1;
                z = y;
        }
}
```

- There are two expressions after “?”.
  - If first condition is fulfilled first is written as an output
  - In other case, second is thread as an output
- It have to be mathematical expression not “any code”

www.agh.edu.pl

Switch statement



```
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";
    }
}
```

- Multiple choice statement
- Selection values must be known during compilation !!!

Switch statement

```
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";
        }
}
```

» Multiple choice statement

» Selection values must be known during compilation !!!

» case x: where x is a variable will not work!!!

Switch statement

```
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";
    }
}
```

- Multiple choice statement
- Selection values must be known during compilation !!!
- case x: where x is a variable will not work!!!
- Curly brackets are optional
- Pay attention to **break;**
- The first "successful case" ends the instruction - the break goes to the end
- "Default" is optional

AGH

I do not understand my own code ... **why?** !@#$%^

Coding standards

- **Rules for unifying the look** and behavior of the code
  - easier analysis of your code and others
  - less chance of making a mistake
  - makes possible a teamwork
- Formatting the code
- Naming conventions
- Commenting code
- **Design Patterns**

Coding standards

- Google C++ Style Guide
  https://google.github.io/styleguide/cppguide.html
- Formatting
- Comments
- Names (conventions)
- Functions
- Scoping
- Classes
- Header files
- Homework: Read the 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, union

enum - declaration

» Type of variable with values are restricted to range (set) of values (enumerators)

```
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 - declaration

- Type of variable with values are restricted to range (set) of values (enumerators)
- Excellent way to enumerate property of something
- **Improves code readability**

```cpp
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 - declaration

- Type of variable with values are restricted to range (set) of values (enumerators)
- Excellent way to enumerate property of something
- **Improves code readability**
- Often in m2m communication
- Implemented as **unsigned int**
- Substitute NAME with integer
- Enumerators (NAME) are counted from 0, unless you define other values
- Naming: capital letter

```
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 - declaration

- Example of use
- Often use together with the switch statement
- Enumerators (RED, BLUE, …) are known during compilation, could be use as a “case”
- Usually declared as global, when used as a part of communication protocol

```
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;
}
```

union

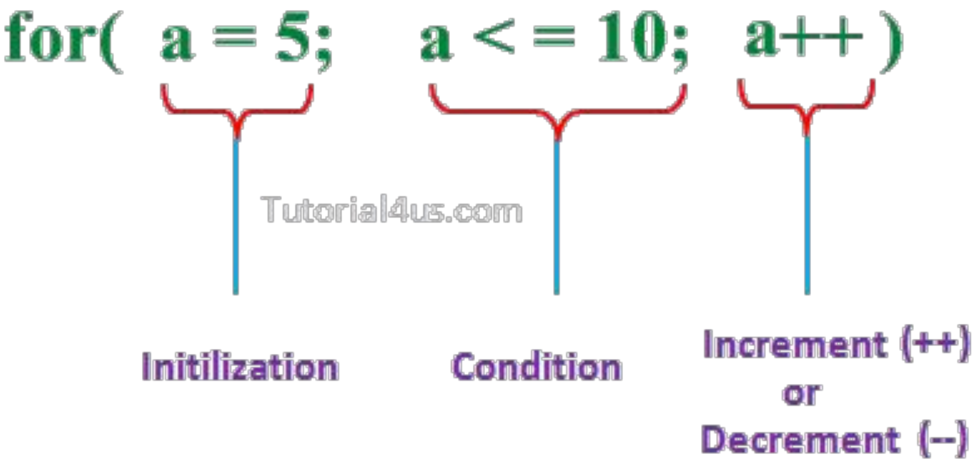
- Rarely used
- Designed to save resources (RAM)
- Mostly in low-level C
- Specific examples

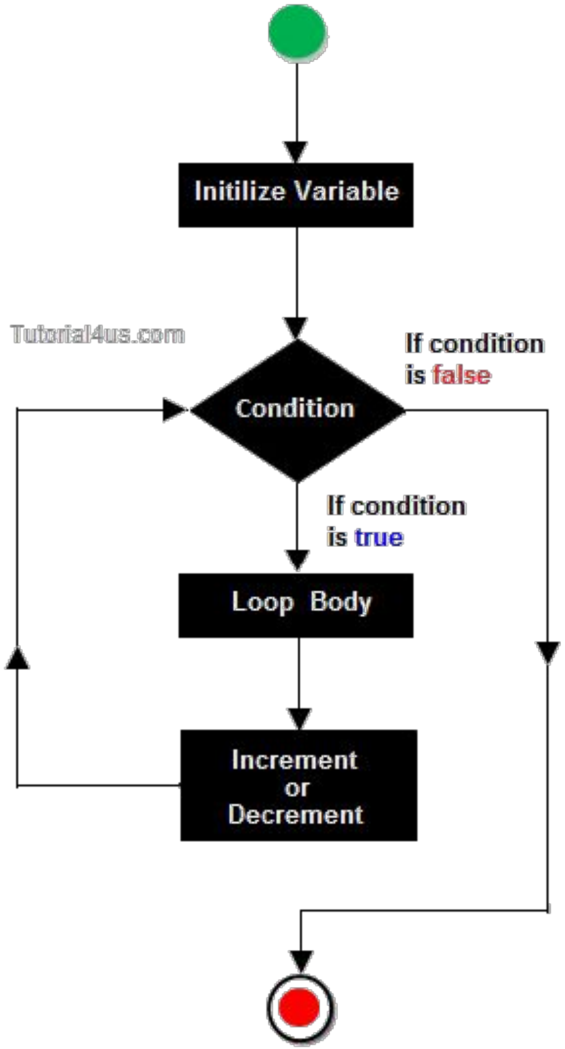
- Read about it if you need

I would like to print integers from 1 ... 1000

do I have to do write 1000 "cout" ???

Loop statement **for**





https://www.sitesbay.com/cpp/cpp-control-flow-statement

for statement

```
int main(){

    for (int i = 0; i < 100; ++i) {
        cout << "iterator: " << i << endl;
    }

    // cout << i;       // error: i out of scope
}
```

- There can be many **Iterators**
- The condition can be complex
- The third expression change iterator

- **Make 100 times the code**
- Perform the same operation on all elements of the set

while statement

```
int main(){

    int startCounter = 10;

    while (startCounter--) {
        cout << startCounter << ", ";
    }

    cout << "liftoff!!!" << endl;
}
```

- Result: **9, 8, 7, 6, 5, 4, 3, 2, 1, 0, liftoff !!!**
- First checked condition, then executed loop
- Instructions in the loop block may never be executed

do-while statement

```
int main(){

    int startCounter = 10;

    do {
        cout << startCounter << ", ";
    }while (startCounter--);

    cout << "liftoff!!!" << endl;
}
```

» Result: 10, 9, 8, 7, 6, 5, 4, 3, 2, 1, 0, liftoff !!!

» First the instructions in the loop, then the condition

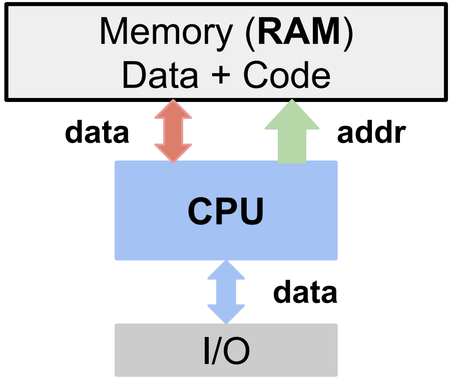
» The instructions in the block will execute at least once

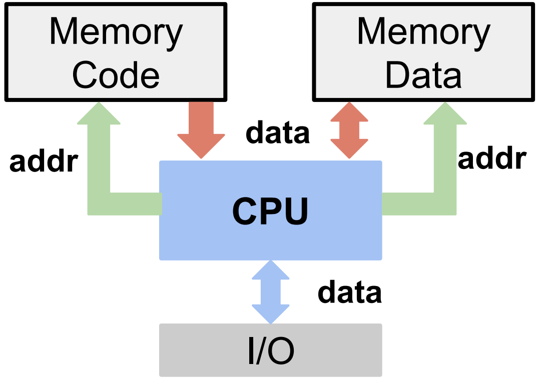
Computer architecture

continuation

www.agh.edu.pl

von Neumann vs Harvard



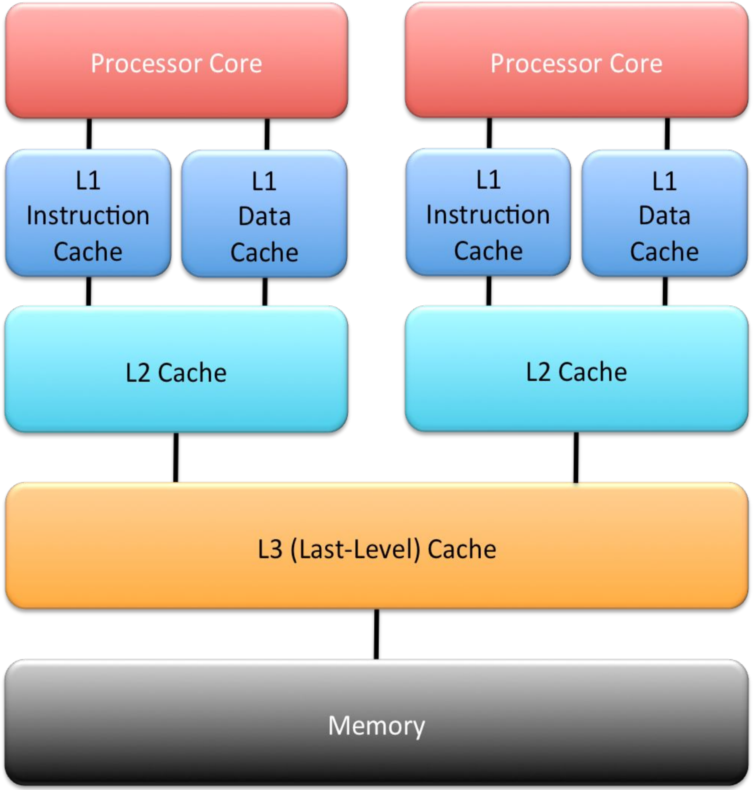


- Single RAM, one buse - **cheap**
- PC, servers, general computing

- Two memory, two busses: parallel access to data and instruction (**faster**)
- Program code is protected against changes
- DSP, uC (short programs)

theory vs reality :-)

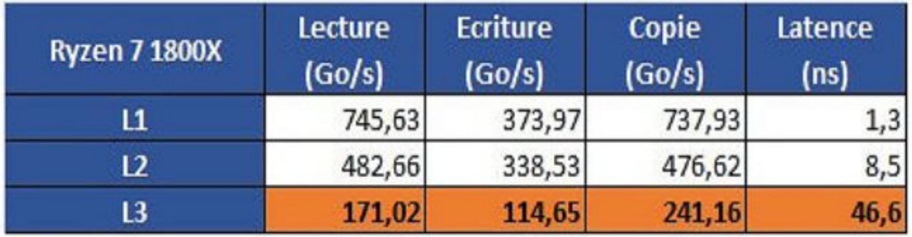
- outside: von Neumann
- inside: "it's complicated"
- RAM (ang. Random Access Memory) DDR4-2400, CL15 to: 2.4 GT/s (x64bits), **1 byte after 50 ns, next after 15 ns**.
- Zen (AMD-Ryzen):
  - L1: 64 KiB instruction + 32 KiB data
  - L2: 512 KiB (per core)
  - L3: 8 MiB (per CXX quad-core)



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

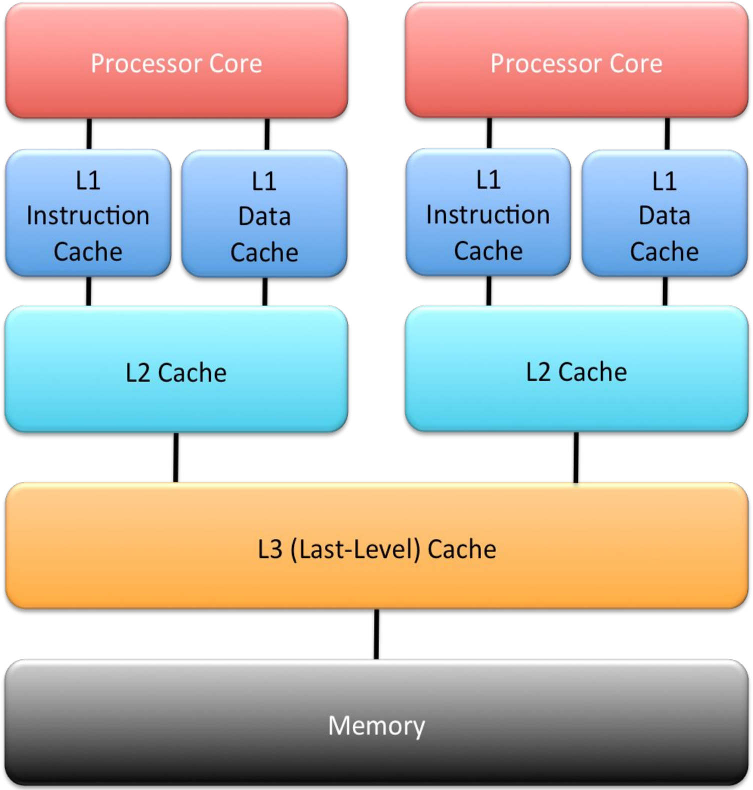
theory vs reality :-)

- outside: von Neumann
- inside: “it’s complicated”
- RAM (ang. Random Access Memory) DDR4-2400, CL15 to: 2.4 GT/s (x64bits), **1 byte after 50 ns, next after 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

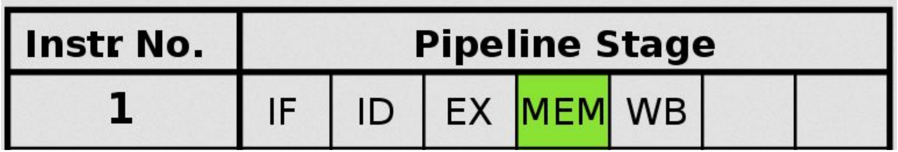
https://www.techpowerup.com/



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

instruction cycle

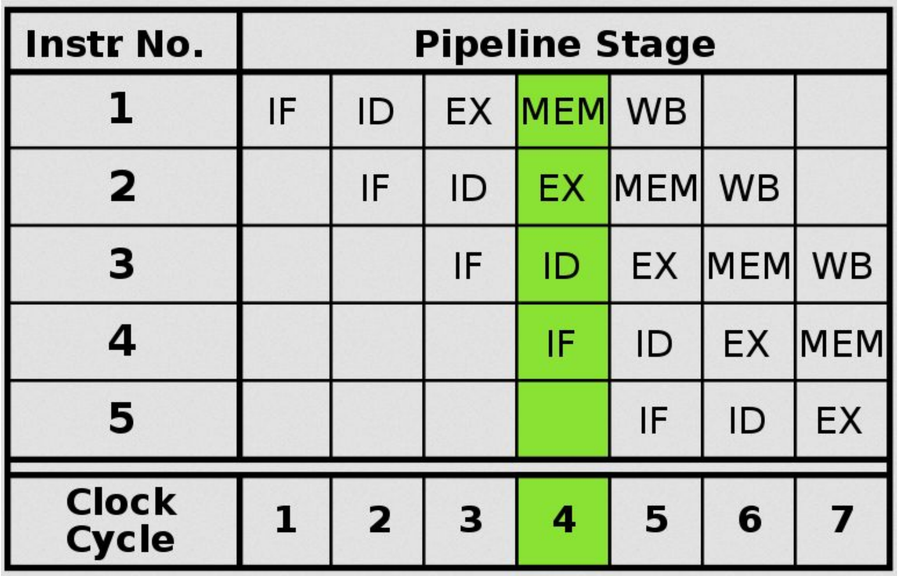
»	**IF**	Instruction Fetch
»	**ID**	Instruction Decode
»	**EX**	Execute
»	**MEM**	Memory access
»	**WB**	Register write back



Instr No.	**Pipeline Stage**						
1	IF	ID	EX	🟩 MEM	WB		

instruction cycle

»	**IF**	Instruction Fetch
»	**ID**	Instruction Decode
»	**EX**	Execute
»	**MEM**	Memory access
»	**WB**	Register write back



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

https://en.wikipedia.org/wiki/Instruction_pipelining

Do I have to manage the whole computer system by myself?

OS

- It is not necessary, you can do it by your own - **bare metal** (Bare machine): Arudino, IoT, automation
- **It is the interface between man and machine**
- (wiki) computer management software, creates an environment to run and control the user's tasks:
  - allocation of CPU time
  - allocating memory (RAM)
  - synchronization between tasks (IPC)
  - other resources management (eg. to HDD access by two processes)

OS features

- Manage tasks/process (create, ends/kill)
- Manage actions like: IPC, interrupt, events, ...
- Manage resources (access rights, access time, limits, conflicts)
- Enables communication with the user

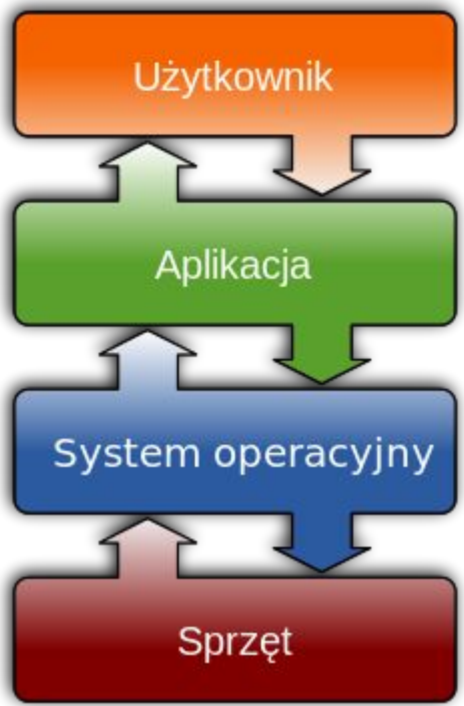
www.agh.edu.pl

Process

- » Instance of executed/started computer program
- » An application can have multiple processes (parallel computing)
- » The process can have multiple threads
- » Each process has an identifier (PID)
- » OS admits resources (RAM, CPU, …) to the process
- » The process may create a new process

Human machine interface

- User tasks (applications) do not communicate directly with the hardware
- OS is an interface between human and machine
- OS controls access to resources
  - access rights
  - access time
- OS queue tasks
  - priorities
  - optimization
- ;-) OS is God, has power over life and death of any process/application



Polska wikipedia, 2009-2015

OS feature

- Multiuser
- Multitasking
- Multiprocessing
- Multithreading
- Preemption
- ...

Multi-user

- Sharing one computer for multiple users
- Remote login - mainframe
- Access rights, ensuring the separation:
  - memory
  - storage

Multitasking

- Feature of the system allows "simultaneous" operation of many processes
- I/O (disk, keyboard, network) is much slower than the CPU
  - run two programs simultaneously,
    - # 1 is waiting for I/O
    - # 2 uses the CPU
  - OS must ensure that are no conflicts of resources (eg. #1, #2 want 100% CPU)
- Scheduler, Planning, priorities, multiprocessing
- RAM - rare good
- Many CPUs, one kernel (Linux: up to 4096 cores)

www.agh.edu.pl

Protection and storage management

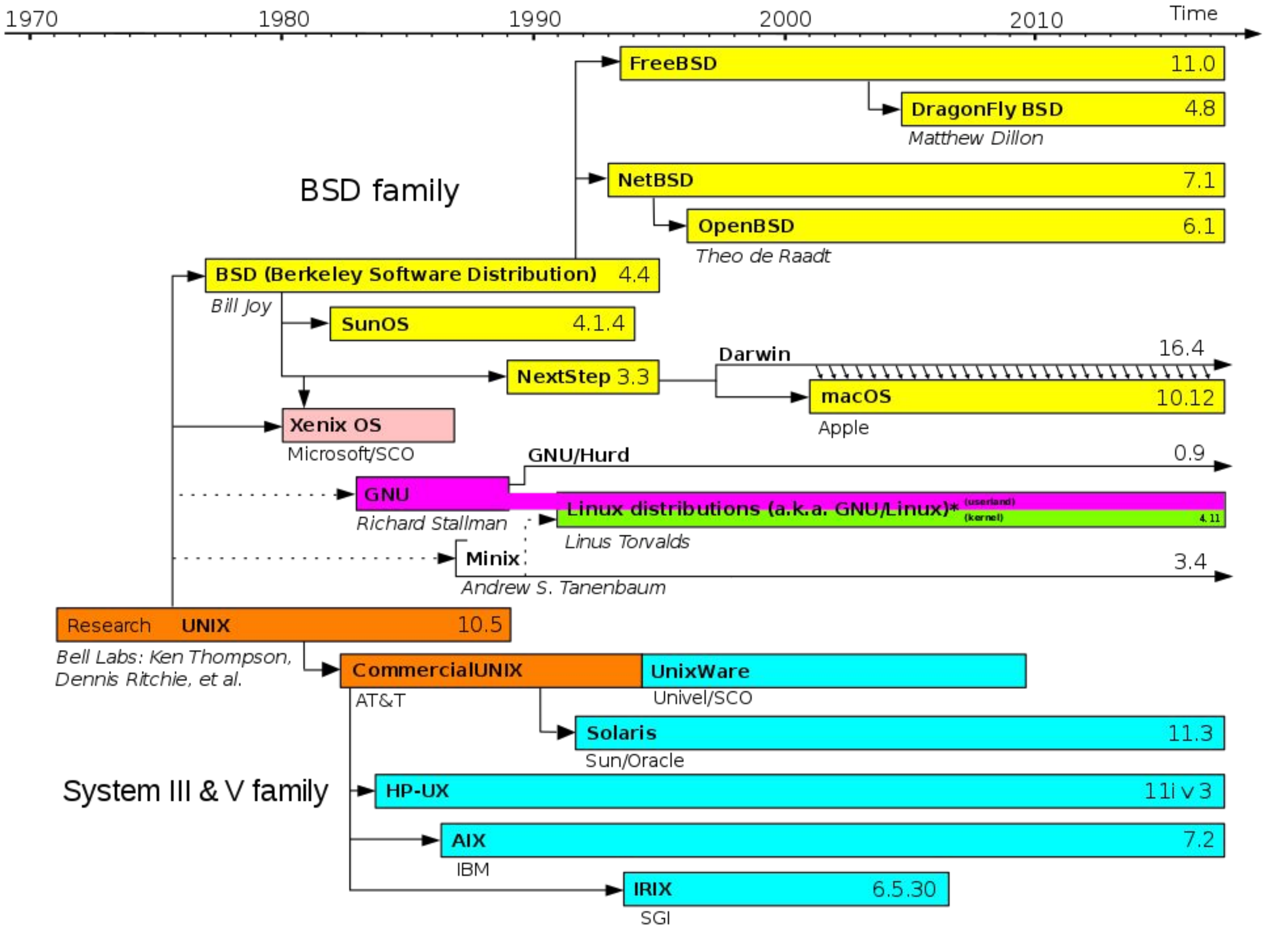
- » Separation of the processes in memory (RAM)
  - – attempt to “illegal” access ends with the interruption of the process (OS kill process)
- » Hardware support - MMU
- » Memory protection prevents the read/overwrite protected memory of another process
- » Why protect RAM?
  - – password
  - – keys
  - – interception process (identity theft)
  - – the possibility of system failure (virus)

Linux

- **1964** - the beginnings of the system **Multics** (**multi**plexed **I**nformation and Computing **S**ervice)
- **1969** - the first system **unix** written in assembler in the center Bell Labs, AT&T
- **1973** - rewrite the code Unix into C (D. B. Kernighan and Ritchie)
- **The 80's** - development of technology: **TCP/IP** (Transmission Control Protocol / Internet Protocol)
- **GNU** OS (**G**NU is **N**ot **U**nix), **POSIX** (Portable Operating System Interface)
- **1991** - Linus Torvalds, a Finnish student, creates operating system kernel Linux
- **1991** -... - establishment and development of many varieties of Linux, Open Source community
- **2008** - Android: (Linux as "firmware")

The main features of the system:

- Multitasking, multiuser
- Scales (SmartWatch ... TOP500)
- Stability
- Openness (the ability to analyze / change)


1970 1980 1990 2000 2010 Time

BSD family

FreeBSD 11.0

DragonFly BSD 4.8

Matthew Dillon

NetBSD 7.1

OpenBSD 6.1

Theo de Raadt

BSD (Berkeley Software Distribution) 4.4

Bill Joy

SunOS 4.1.4

NextStep 3.3

Darwin 16.4

macOS 10.12

Apple

Xenix OS

Microsoft/SCO

GNU/Hurd 0.9

GNU

Richard Stallman

Linux distributions (a.k.a. GNU/Linux)* (userland) (kernel) 4.11

Linus Torvalds

Minix 3.4

Andrew S. Tanenbaum

Research UNIX 10.5

Bell Labs: Ken Thompson, Dennis Ritchie, et al.

CommercialUNIX

AT&T

UnixWare

Univel/SCO

Solaris 11.3

Sun/Oracle

System III & V family

HP-UX 11i v3

AIX 7.2

IBM

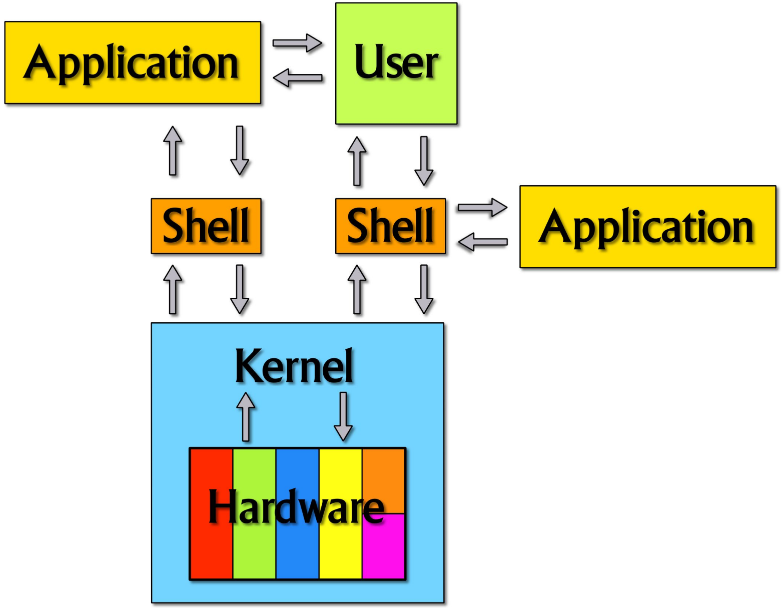
IRIX 6.5.30

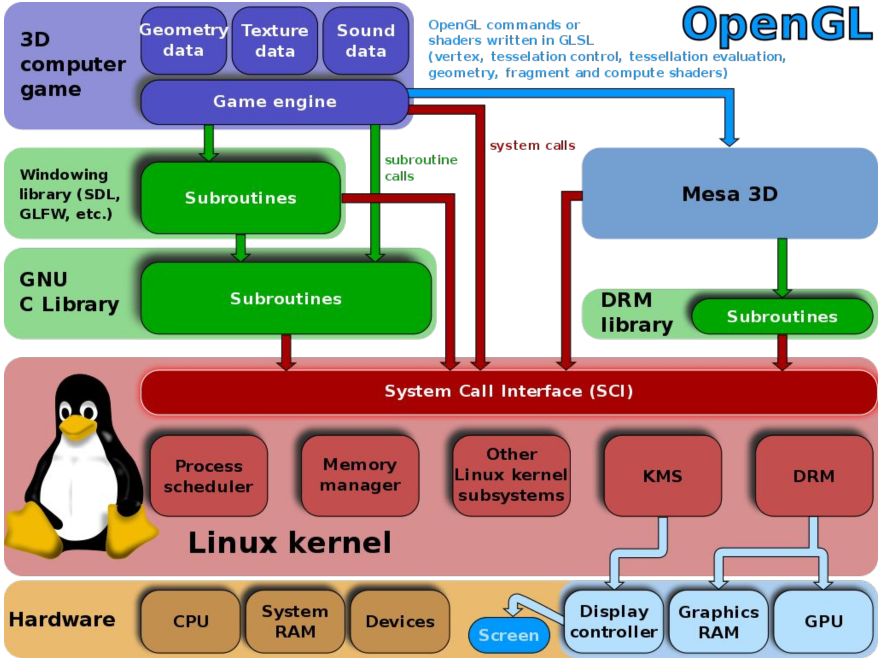
SGI

*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.

https://en.wikipedia.org/wiki/Linux

Linux - architecture





https://en.opensuse.org/images/e/e2/Flow1.jpg

https://en.wikipedia.org/wiki/Linux

Why Linux?

- It is everywhere (especially in network devices)
- It becomes an industry standard
- There is no problem of vendor lock-in
- You can deeply analyze kernel code and system
- You can modify the source code (even better is *BSD)
- Nice development platform
- You can change system and still legally distribute (sell) it
-
- I recommend Linux as the primary OS on the classes

Linux distribution (distro)

- A complete operating system:
- Linux kernel
- GNU tools, libraries (eg. Glibc)
- Various "extras"
- Windows system, window manager, desktop (Gnome, KDE, ...)
- Package manager
  - 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)

How to interact with OS

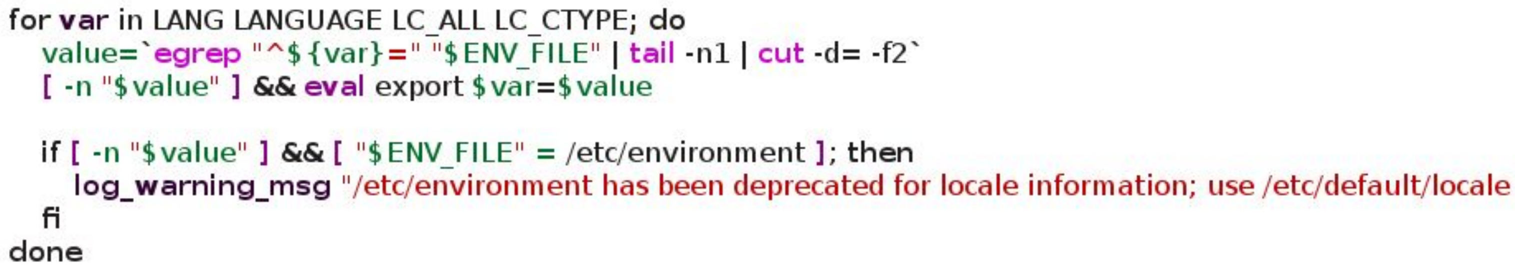
www.agh.edu.pl

SHELL

- User interface for access to an operating system's services
- shell, terminal, console, CLI (Command Line Interface)
- Historically, the first way of communicating with a computer (a powerful tool so commonly used today by power user :))
- Program: sh, bash, tcsh, fish
- "Standard output" (stdio/stderr) for graphic programs (mostly errors, information, etc ...)
- Allows for seamless remote work
- MS-DOS was also a kind of shell (XP + provides powershell)

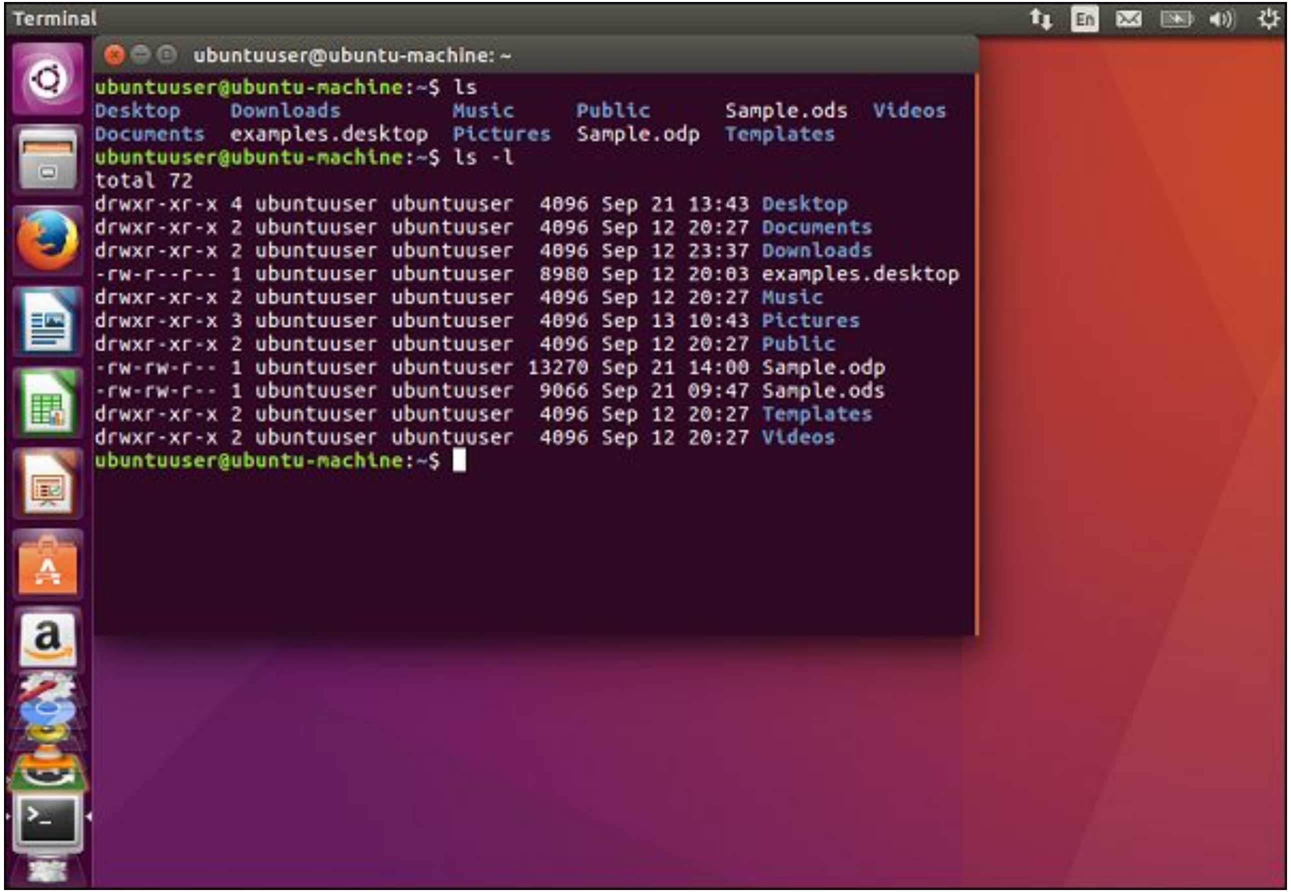
SHELL

- » built-in commands eg. cd (could be different on bash/tcsh)
- » external commands, eg. cp, mv ('whereis mv)
- » built-in commands eg. pwd cover system: 'pwd --help' which displays other than '/bin/pwd --help')
- » sh/bash/fish is also a scripting language, which can be used to write a program:



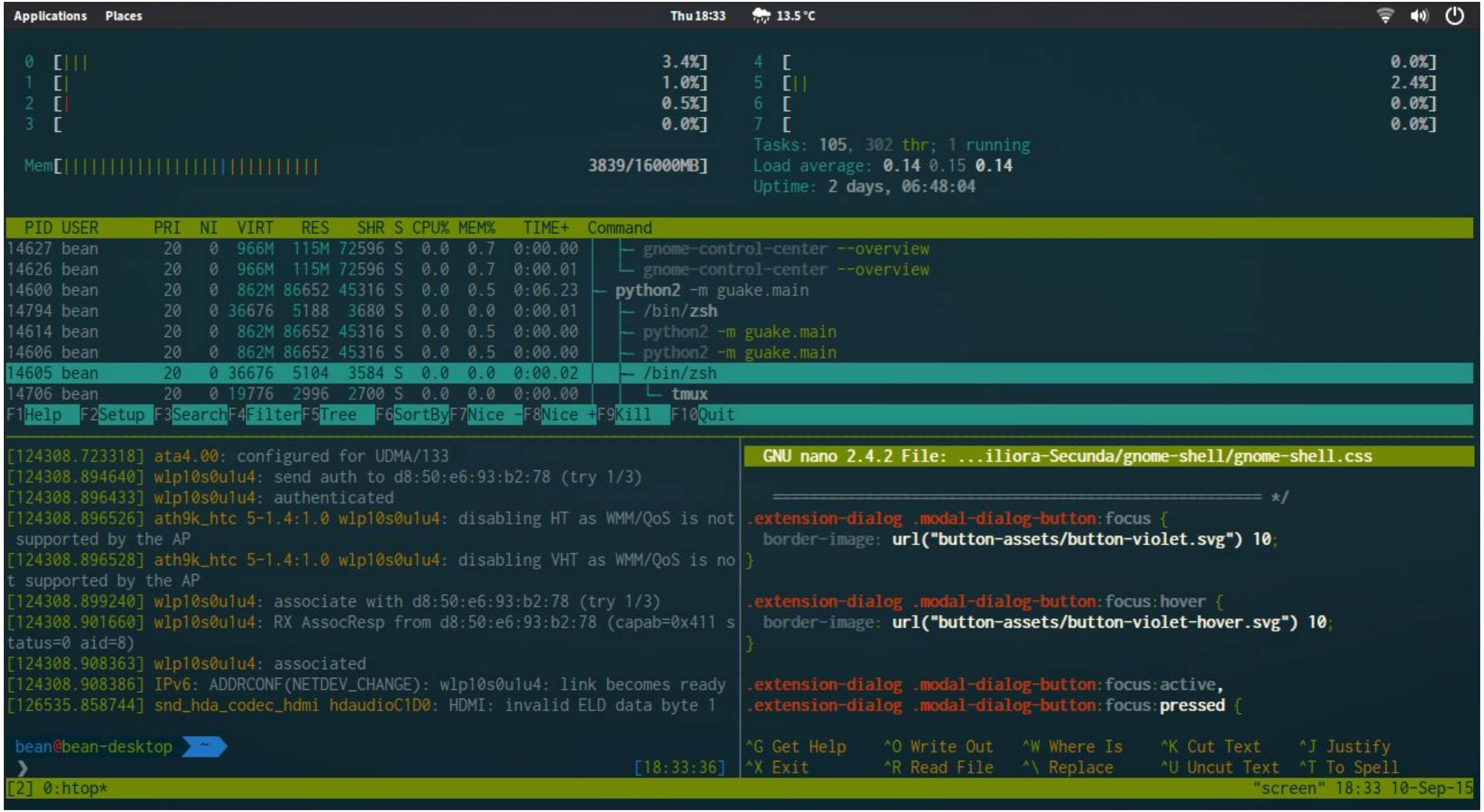
```
for var in LANG LANGUAGE LC_ALL LC_CTYPE; do
  value=`egrep "^${var}=" "$ENV_FILE" | tail -n1 | cut -d= -f2`
  [ -n "$value" ] && eval export $var=$value

  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



```
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



```
  0  [|||                                              3.4%]    4  [                                              0.0%]
  1  [|                                                1.0%]    5  [||                                            2.4%]
  2  [|                                                0.5%]    6  [                                              0.0%]
  3  [                                                 0.0%]    7  [                                              0.0%]
                                                                Tasks: 105, 302 thr; 1 running
  Mem[||||||||||||||||||||||||||               3839/16000MB]    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 F3SearchF4FilterF5Tree  F6SortByF7Nice -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 no
t 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 s
tatus=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:hover {
  border-image: url("button-assets/button-violet-hover.svg") 10;
}

.extension-dialog .modal-dialog-button:focus:active,
.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
```

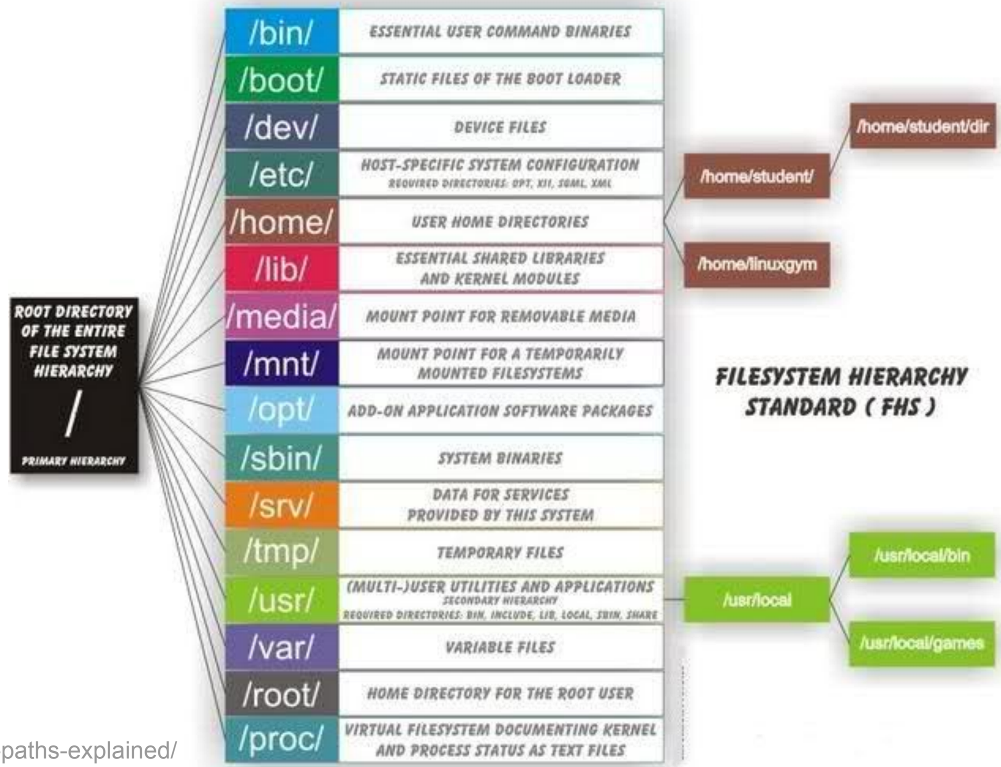
SHELL - how to...

- **General scheme of commands**:

  user@host:dir$ command_name [options, arguments ...] <enter>
- Options letters preceded by the sign "-"And the verbal signs "--": Eg .: ls -al
- Each command has usually help "-h" or "--help"
- Additional information manual provides: $man name
- **Remote work**:
  - ssh moj_nick@student.agh.edu.pl
  - and here we have console on the remote machine (looks and behave same like local)
- **Interactive tutorial**: https://www.learnshell.org/

The file system structure

- FHS (Filesystem Hierarchy Standard)
- Mount point - any empty directory
- "File extension" is not decoration information



ROOT DIRECTORY OF THE ENTIRE FILE SYSTEM HIERARCHY

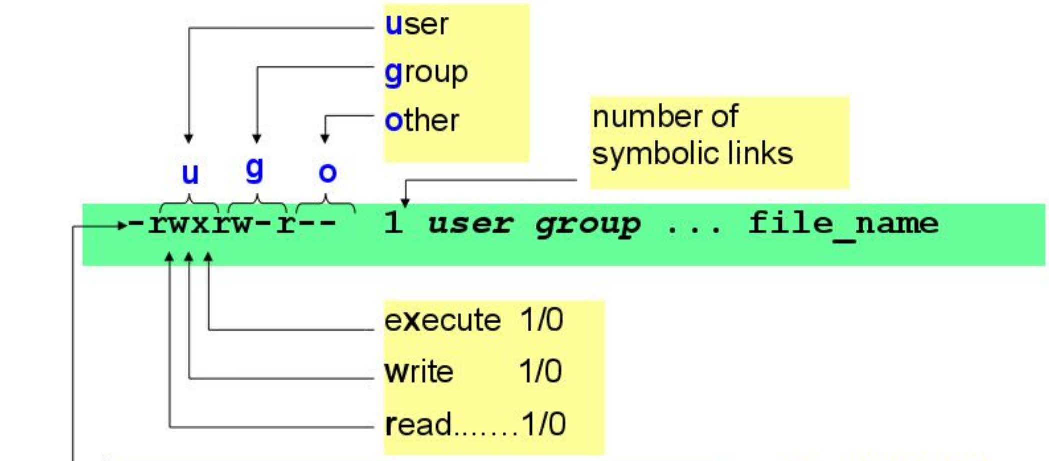
/

PRIMARY HIERARCHY

/bin/	ESSENTIAL USER COMMAND BINARIES
/boot/	STATIC FILES OF THE BOOT LOADER
/dev/	DEVICE FILES
/etc/	HOST-SPECIFIC SYSTEM CONFIGURATION REQUIRED DIRECTORIES: OPT, X11, SGML, XML
/home/	USER HOME DIRECTORIES
/lib/	ESSENTIAL SHARED LIBRARIES AND KERNEL MODULES
/media/	MOUNT POINT FOR REMOVABLE MEDIA
/mnt/	MOUNT POINT FOR A TEMPORARILY MOUNTED FILESYSTEMS
/opt/	ADD-ON APPLICATION SOFTWARE PACKAGES
/sbin/	SYSTEM BINARIES
/srv/	DATA FOR SERVICES PROVIDED BY THIS SYSTEM
/tmp/	TEMPORARY FILES
/usr/	(MULTI-)USER UTILITIES AND APPLICATIONS SECONDARY HIERARCHY REQUIRED DIRECTORIES: BIN, INCLUDE, LIB, LOCAL, SBIN, SHARE
/var/	VARIABLE FILES
/root/	HOME DIRECTORY FOR THE ROOT USER
/proc/	VIRTUAL FILESYSTEM DOCUMENTING KERNEL AND PROCESS STATUS AS TEXT FILES

https://www.tecmint.com/linux-directory-structure-and-important-files-paths-explained/

File attributes (metadata)



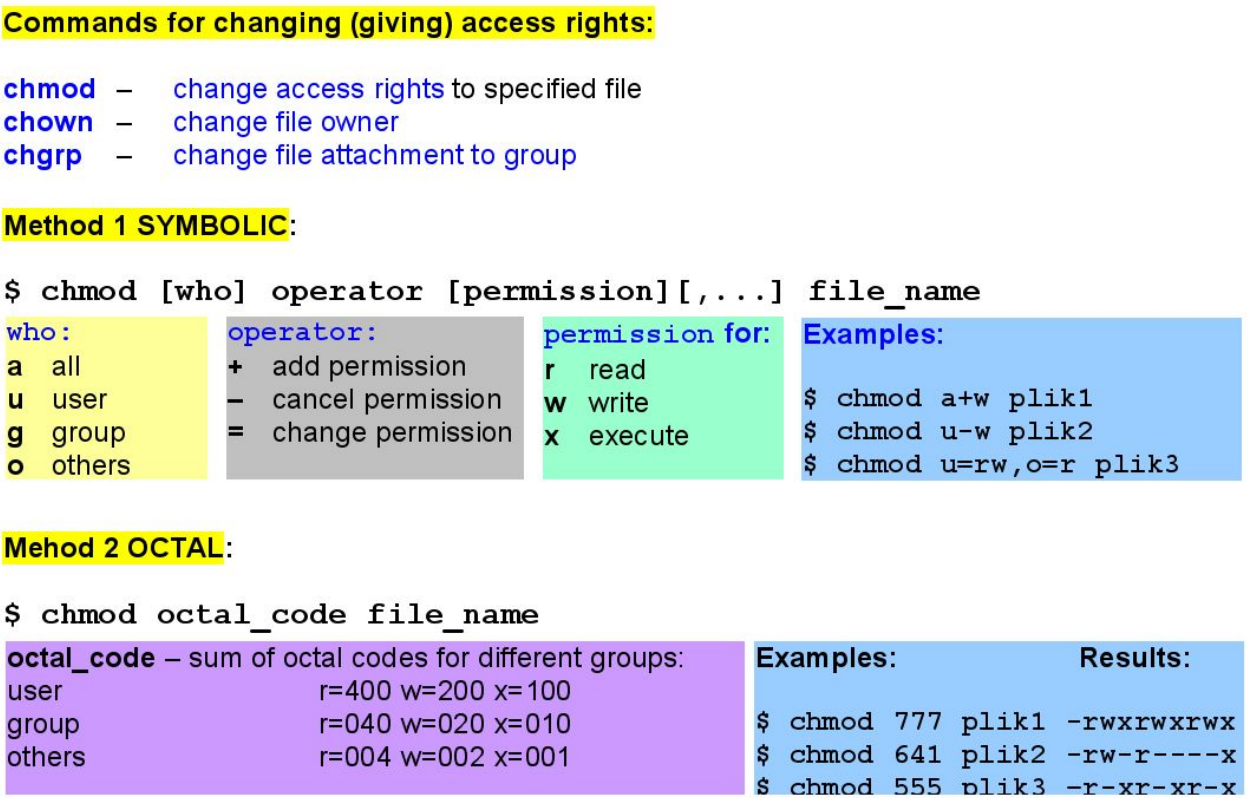
-	ordinary file
d	directory = list of files
l	**symbolic link**
c	urządzenie znakowe (np. modem)
b	block device (e.g. disc)
p	named pipe
s	socket

Symbolic link

Special file type that is pointing to other file or directory. Link can be „soft” or „hard”.

In Microsoft Windows similar role plays shortcut.

Atrybuty plików (metadane)



Commands for changing (giving) access rights:

chmod	–	change access rights to specified file
chown	–	change file owner
chgrp	–	change file attachment to group

Method 1 SYMBOLIC:

```
$ chmod [who] operator [permission][,...] file_name
```

who:	
a	all
u	user
g	group
o	others

operator:	
+	add permission
–	cancel permission
=	change permission

permission for:	
r	read
w	write
x	execute

Examples:

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

Mehod 2 OCTAL:

```
$ chmod octal_code file_name
```

octal_code – sum of octal codes for different groups:

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

Examples: **Results:**

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

SHELL - ToDo

- Environment Variables
- Remote work
- Manipulating files
- Text manipulation (read/modify)
- Streams I/O (in particular stdio/stderr)
- scripts
  - conditional statements
  - loops
  - manipulation of files

Thank you!