Methodology and Programming Techniques

Department of Telecommunications, IEiT

dr inż. Jarosław Bułat

kwant@agh.edu.pl

www.agh.edu.pl

outline

- IDE
- Pointers
- Pointers and tables
- Pointers arithmetic
- Text in tables
- String type
- Pointer to structure
- Size of pointer
- Dynamic memory management (stos/heap)

www.agh.edu.pl

IDE

- **I**ntegrated **D**evelopment **E**nvironment
- System (program) to facilitate writing program
  - source file editor
  - project organization (complex program)
  - compilation
  - start-up
  - debugging
- CLion, Visual Studio, XCode, Eclipse, NetBeans IDE, Code :: Blocks, Qt Creator, Visual Studio Code, VIM + console, ~~devCPP~~

IDE

- Dedicated to one language or universal
  - Eclipse: Java, C/C ++, Python, PHP, etc ...
- Windows only (VC) or cross-platform
- Free - usually Open Source or Community Edition
- Commercial - CLion (jetbrains.com)
  - business: 200-100 EUR/year + VAT
  - individual: 90-50 EUR / year
  - students: free
- Simple: Atom, Visual Studio Code

IDE - editor

- It makes code writing easier
- Suggests variable names, arguments, ...
- Help - library documentation, features, environment, ...
- Refactoring
- Error parsing (+ compilation errors)
- Color syntax
- Helps code formating (indentation)
- Teamwork - git: diffs, versions, ...
- Debugger

IDE - Eclipse

- **Project:** File/New/C++ Project: Executable/Hello..., Toolchains: Linux GCC
  - lab: each program is a separate project !!!
- **Compilation**: Ctrl-b, toolbar: hammer
- **Run/Execute**: Ctrl-F11, toolbar: play
- Perspective: C/C++, Debug, Team
- Profile: Debug, Release
- save (ctr-s) -> compile (ctr-b) -> run (ctr-s)
- Project preferences
  - linking libm: /lib/x86_64-linux-gnu/libm.so.6

IDE - Visual Studio Code

- Simple environment
- Good for starting programming
- Hard to start project (need to configure in JSON file)
- Plugin Code Runner:
  - compile and run single-file C/C++ program
  - do not need to start project
  - hard to change compiler option
- License :-/

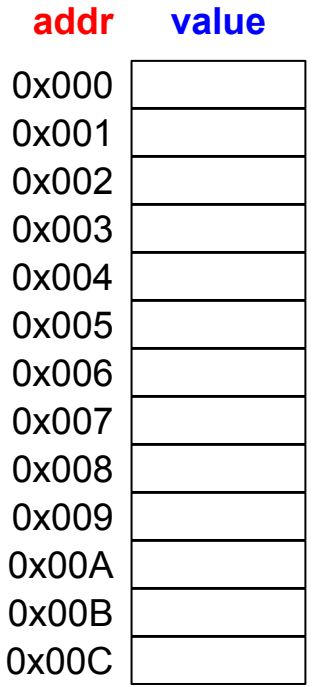
Pointers

You can like or hate it :-)

www.agh.edu.pl

Memory organization

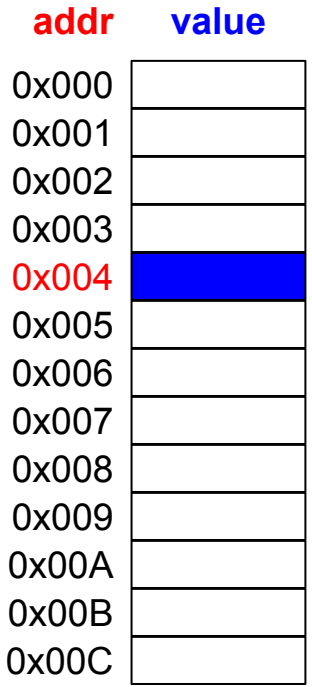
- Memory is continuous
- The single cell has a size of 8 bits
- Each cell has a unique address



www.agh.edu.pl

Memory organization

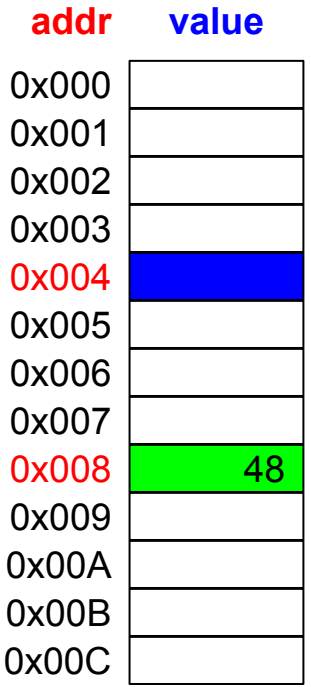
- Memory is continuous
- The single cell has a size of 8 bits
- Each cell has a unique address
- Cell content is available through its address
- **Access by address is the only way at the hardware level**



www.agh.edu.pl

Memory organization

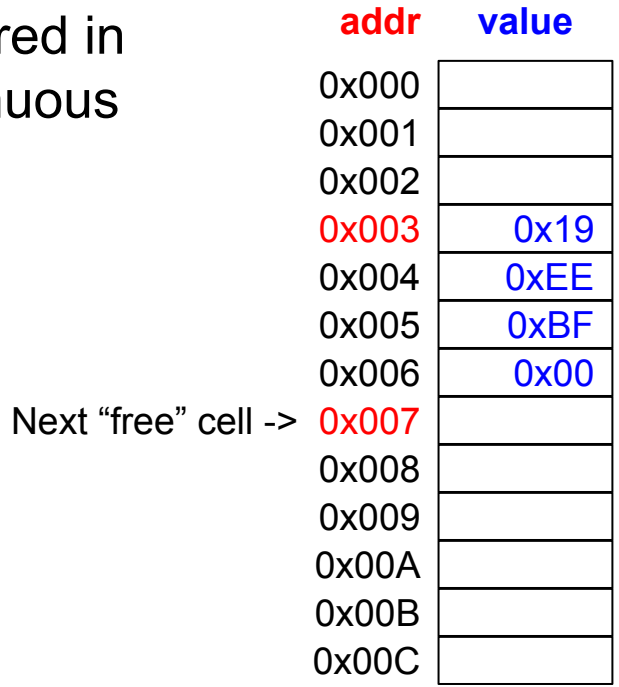
- Memory is continuous
- The single cell has a size of 8 bits
- Each cell has a unique address
- Cell content is available through its address
- **Access by address is the only way at the hardware level**
- Variable char c = 48; is in one memory cell (its value)
  - the program has access to it by name or address
  - the name of the variable "c" exists only in the program !!!



www.agh.edu.pl

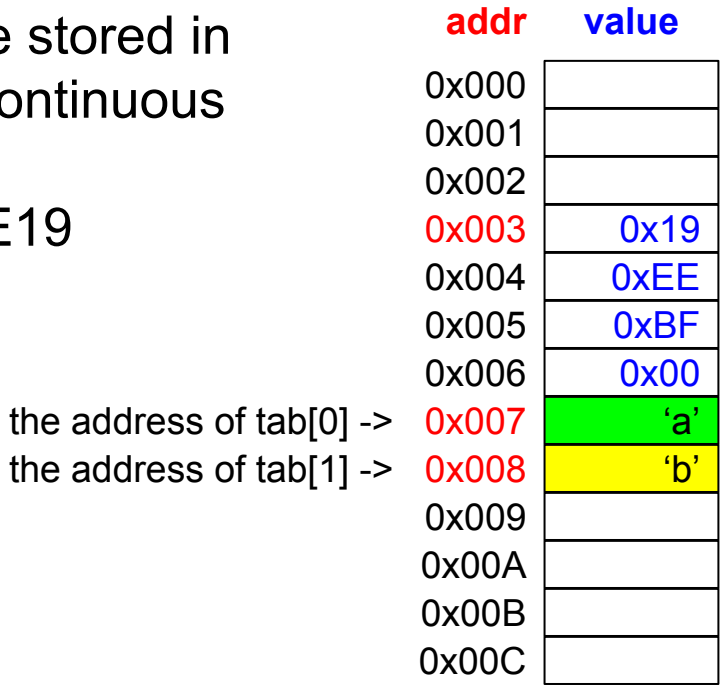
Memory organization

- Variables of size >1 Byte are stored in consecutive addresses (in continuous space)
- **int x** = **12578329**; // 0xBFEE19



Memory organization

- Variables of size >1 Byte are stored in consecutive addresses (in continuous space)
- **int x** = **12578329**; // 0xBFEE19
- **char tab[2];**
  - **tab[0] = ‘a’;**
  - **tab[1] = ‘b’;**



www.agh.edu.pl

Pointers

- This pointer is a variable whose value is the address of another variable (a variable that "points" another variable)
- A pointer that does not point another variable is uninitialized

`int *x;` declaration of pointer to int (to int type)

x is a type of "pointer to int"

`x = &y;` operator & (address-of) to obtain the address of the variable y

to x, the address of y is assigned (not its value!!!)

`int z = *x;` dereference the value of the variable

addr	value
0x000	
0x001	
0x002	
0x003	0x19
0x004	0xEE
0x005	0xBF
0x006	0x00
0x007	
0x008	
0x009	
0x00A	
0x00B	
0x00C	

www.agh.edu.pl

Pointers

- int x = 12578329; // 0xBFEE19
- int *y = **&x**; // initialization

- x == 12578329 type: int
- &x == 0x003 type: pointer to int
- y == 0x003 type: pointer to int
- *y == 12578329 type: int

addr	value
0x000	
0x001	
0x002	
0x003	0x19
0x004	0xEE
0x005	0xBF
0x006	0x00
0x007	
0x008	
0x009	
0x00A	
0x00B	
0x00C	

www.agh.edu.pl

Pointers

- int x;
- int *y = &x;
- char z;

x = y;	error, attempt to assign an address to int
y = x;	error, attempt to assign an int value to an address
cout << *x;	error, attempt to dereference a variable instead of a pointer
y = &z;	error, types do not match int* != char*

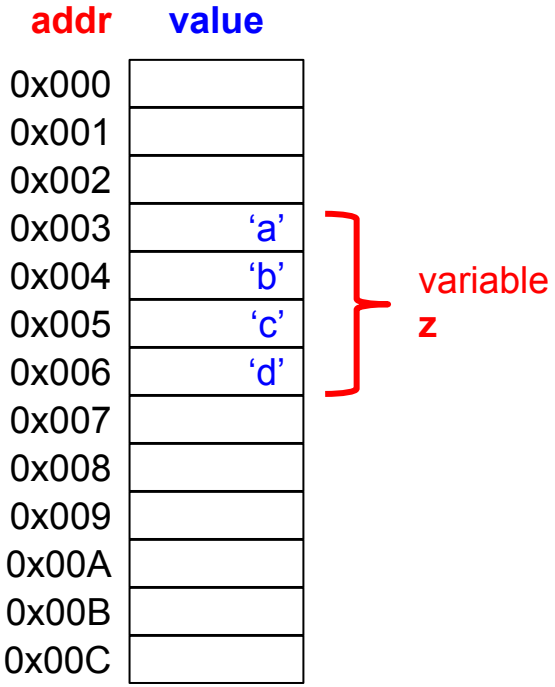
www.agh.edu.pl

Pointers

- char x0 = ‘a’; // &x0 == 0x003
- char x1 = ‘b’; // &x1 == 0x004
- char x2 = ‘c’; // &x2 == 0x005
- char x3 = ‘d’; // &x3 == 0x006

- char *y = &x0; // ok
- cout << *y; // ok

- int *z = &x0;// not ok!!!
- cout << *z; // not ok!!!



www.agh.edu.pl

Namespace

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

int main() {

    std::cout << "ala ma kota" << std::endl;
    cout << "ala ma kota" << endl;

}
```

- Allows the same names of variable/functions in one program:
  - x::test = 7;
  - y::test = 8;

Pointers: obtain address of variable

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

int main() {

      int x = 4;
      int *y;     // pointer to int

      y = &x;    // address-of operator

      cout << y << endl; // 0x7fffbe6781bc
      cout << *y << endl;      // 4

}
```

» "Cout" will understand that the int* must be typed as an address not value

www.agh.edu.pl

Pointers: dereference

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

int main() {

    int x = 4;
    int *y;          // pointer to int

    y = &x;          // address-of operator
    *y = 6;          // dereference

    cout << x << endl; // 6

}
```

- The dereference operation is read/write:
  - you can read the value of pointed variable
  - you can also change it

Pointers: obtain address

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

int main() {

        int x0 = 4;
        int x1 = 7;
        int *y;

        y = &x0;
        cout << *y << endl;      // 4

        y = &x1;
        cout << *y << endl;      // 7

}
```

- The pointer is a variable so you can change it :)
  - once points to x0
  - another time to x1
- It is possible to print values of different variables with the help of one pointer y!!!

www.agh.edu.pl

Pointers: obtain address

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

int main() {

      int x = 4;
      int *y0, *y1;

      y0 = &x;
      y1 = y0;

      cout << *y1 << endl;     // 4

}
```

» The pointer is a variable so I can assign its value (address of variable x) to another pointer

– &x is in y0 and y1

www.agh.edu.pl

Examples of use of (pointers)

www.agh.edu.pl

Pointers and tables

```
char tab[]={'a','b','c','d'};
char *c;

c = tab;                    // ok!
cout << *tab << endl;       // a
cout << c[3] << endl;       // d

// c = tab[0];              // not ok!!!
c = &tab[3];
*c = 'a';

cout << tab[3] << endl; // ???
```

- tab variable is of type char*
- tab points the first element of the array

www.agh.edu.pl

Table of characters

```
char tab[] = "Hello World!!!";

cout << tab[0] << endl;   // H
cout << tab[1] << endl;   // e
cout << tab << endl     // Hello World!!!
cout << sizeof(tab) << endl;    // 15

char last = tab[sizeof(tab)-1];
cout << int(last) << endl;       // 0 (end of line)
```

- Array of characters is a "string"
- It is a convention in C language
- The last character is the end of the string '\0'.
- Deprecated in C++

www.agh.edu.pl

String type

```
char *tab = "Hello World!!!";
string str = "Hello World!!!";

cout << "C:  " << tab << endl;
cout << "C++:" << str << endl;

str = "My Longer Hello World!!!";
cout << "C++:" << str << endl;
```

- Text stored in the table is a problem:
  - can not change length
  - difficult to maintain the end of the text - security vulnerability
- "xxxx" in the source code is a constant string
- In C++ text should be stored in string type
- String type is flexible and safer
- String is part of the standard library

www.agh.edu.pl

Pointer to the structure

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

int main() {
        Product p = {1, .5};
        Product *x = &p;

        p.weight = 2;
        x->weight = 4;
        float my_price = x->price;

        cout << p.weight << endl;     // ??
        cout << my_price << endl;     // ??
}
```

- The pointer to the structure acts just like the pointer on the variable
- Structure Addressing:
  - operator . for variables
  - operator -> for the pointers

www.agh.edu.pl

Pointer to the structure in the array

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

int main() {
        Product p[10];
        Product *prod;
        float weight;

        weight = p[4].weight;
        prod = p[4];    // błąd !!!
        prod = &p[4];
        weight = prod->weight;
        weight = (&p[4])->weight;
}
```

» Obtain the address of a single array element is the same as the address of a variable

www.agh.edu.pl

Pointer size

```
struct Product {
        int shape[20];
        float price;
}prod;

int main() {
        char *pc;
        int *pi;
        Product *pp = &prod;

        cout << sizeof(prod) << endl;      // 84
        cout << sizeof(pc) << endl;        // ??
        cout << sizeof(pi) << endl;        // ??
        cout << sizeof(pp) << endl;        // ??
        cout << sizeof(*pp) << endl; // ??
}
```

- Variable pp is not a copy of the prod variable but it is they address
- The size of the pointer is constant, independent of the size of the variable it points to

Pointer to non-existent object

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

int main() {
      int *x;
      int y = 10;

      if (y > 5) {
            int z = 2*y;
            x = &z;
      }

      cout << x << endl; // ok (but pointless)
      cout << *x << endl;      // Error !!!
}
```

- In condition: the pointer receives the address of the variable that will no longer exist
- Outside condition: x points to memory, which **was** occupied by a variable but **now** is not valid (variable no longer exist!)
- If y <= 5, the pointer is uninitialized !!!

www.agh.edu.pl

Pointers inception ;-)

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

int main() {
        int x;
        int *y;
        int **z;

        y = &x;
        z = &y;
}
```

- The pointer is a variable
- I can get the address of the pointer
- I can get pointer to a pointer to a variable.

What does it mean to increment the pointer?

ie arithmetic of indicators

www.agh.edu.pl

Pointers arithmetic

- The pointer is the memory address
- Incrementing pointer increases address by sizeof(type)
- `int *p = &x; p++;` will increase the value of p by sizeof(int)
- All operations on the pointer are changes by sizeof(type)
- It is used almost exclusively in conjunction with arrays

www.agh.edu.pl

The array name is the address

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

int main(){
   int tab[] = {4, 3, 2, 1, 0};
   int *p = tab;

   cout << tab[0] << endl;     // 4
   cout << *tab << endl;// 4
   cout << *p << endl;         // 4
}
```

» The array name is the address to its first element

Addressing array by pointer

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

int main(){
   int tab[] = {4, 3, 2, 1, 0};
   int *p = tab;

   cout << *p << ": " << p << endl;          // 4: 0x7fff138587d0
   p++;
   cout << *p << ": " << p << endl;          // 3: 0x7fff138587d4
   cout << *(p+1) << ": " << p+1 << endl;    // 2: 0x7fff138587d8
   cout << *p+1 << ": " << p+1 << endl;      // 4: 0x7fff138587d8
}
```

- Array of the type int means it's addresses change +=4 bytes
- *p+1, operators priority !!!

www.agh.edu.pl

Addressing array by pointer

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

int main(){
   char tab[] = "ala ma kota";
   char *p = tab;

   for (size_t i = 0; i < 11; ++i) {
      cout << *(p++);
   }
   cout << endl;
}
```

- Change pointer to the next element (cell)
- Dereference pointer (value of next cell in array)
- Results:
  ala ma kota

www.agh.edu.pl

Addressing array by pointer

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

int main(){
   char tab[] = "ala ma kota";
   char *p = &tab[3];

   for (size_t i = 0; i < 11; ++i) {
      cout << *(p++);
   }
   cout << endl;
}
```

- You can start "viewing" the array from any location
- From which character did I start printing?
- What error is in the loop?

www.agh.edu.pl

Addressing array by pointer

```
int main(){
   char tab[] = "ala ma kota";
   char *p;
   p = &tab[0];  // p = tab;

   for (size_t i = 0; i < 11; i+=2) {
      cout << *p;
      p+=2;
      // cout << *(p + i);
      // cout << *(tab + i);
   }
   cout << endl;
}
```

- &tab[0] == tab
- You can jump every few items
- You can use pointers in an arithmetic expression

www.agh.edu.pl

Addressing array by pointer

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

int main(){
   char tab[] = "ala ma kota";
   char *p = tab;

   while ( *p ) {
      cout << *p++;   // *(p++)
   }
   cout << endl;
}
// while( p ) {}  what kind of error?
```

- In "C" text is a sequence of characters + the end of the sequence '\0' (means 0)
- The loop uses '\0' as an exit condition (pointer dereference not pointer itself)
- Incrementation has higher priority than dereference so it will affect the pointer but will do not change value !!!
- Not recommended (will fail if no '\0' at the and)

www.agh.edu.pl

Addressing array by pointer

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

int main(){
   char tab[] = "ala ma kota";

   for( size_t i = 0; i < 11; ++i ) {
      cout << *(tab+i);
      // tab++;
   }
   cout << endl;
}
```

- The tab is the address, so you can use it to index the array
- The tab is const, so you can not change it !!!
- ex26.cc:9:12: error: lvalue required as increment operand

www.agh.edu.pl

Pointers subtraction

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

int main(){
   int tab[] = {4, 3, 2, 1, 0};
   int *p0 = tab;
   int *p1 = &tab[2];

   cout << p0 << endl;       // 0x7ffdfaaa6050
   cout << p1 << endl;       // 0x7ffdfaaa6058
   cout << p1-p0 << endl;    // 2 (not 2*sizeof(int))
}
```

» Subtraction of pointers gives result in:
multiplications sizeof(type)

www.agh.edu.pl

Pointers comparison

```
int tab[11];
int *start = tab;
int *end = &tab[10];
// init tab and print

while (end > start) {
    int tmp = *end;
    *end = *start;
    *start = tmp;
    end--;
    start++;
}
// print tab
```

» Loop changes the order of items in the array

www.agh.edu.pl

Table of pointers

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

int main(){
   int tab[] = {4, 3, 2, 1, 0};
   int *p[2] = {&tab[0], &tab[4]};
   // int *p[2] = {tab, tab+4};

   cout << *p[0] << endl;     // 4
   cout << *p[1] << endl;     // 0
}
```

- Declare an array whose element is pointer to int
- For example, a 2D table, the first column contains pointers to the beginning of each row
  - each line can be of different size

Pointers arithmetic only for array

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

int main(){
   int x0 = 0;
   int x1 = 1;
   int x2 = 2;

   int *p = &x1;
   cout << *p << endl;     // 1
   p++;                     // !@#$%^&
   cout << *p << endl;     // 0
}
```

- Pointers arithmetic can be use only with continuous memory
- What do I expect from this code? Do I jump to the next variable?

- Never ever!!!

www.agh.edu.pl

Pointers arithmetic only for array

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

int main(){
   int x0 = 0;
   int x1 = 1;
   int x2 = 2;

   int *p;
   p = &x1 + &x2;  // ???
   // ex30.cc:10:16: error: invalid operands of types 'int*' and 'int*' to binary
'operator+'
}
```

- I'll add two addresses
- It is so stupid that my compiler will laugh ;-)

www.agh.edu.pl

Arrays implementation

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

int main(){
   char tab[] = "It's Magic!!!";

   char c;
   c = tab[10];
   c = *(tab+10);

   // *(tab+10) = *(10+tab) = 10[tab]
   cout << c << endl;          // !
   cout << 10[tab] << endl;    // !
}
```

» Indexing arrays is implemented by means of pointer arithmetic:

$$tab[10] = *(tab+10)$$

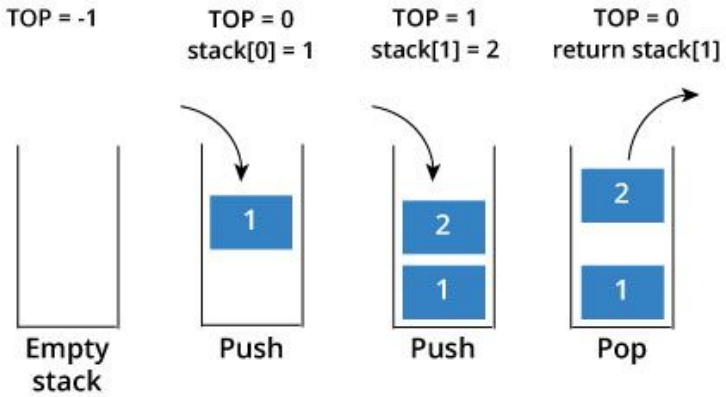
www.agh.edu.pl

Stos/Heap

dynamic memory management

www.agh.edu.pl

Stos



https://www.programiz.com/dsa/stack

- Linear data structure
- **LIFO** type buffer (Last In, First Out)
- push(), pop(), isEmpty()
- Patented in 1957

- Memory is common to all programs
- Memory reservation requires* multitasking suspension
- Stack is implemented in hardware (CPU)
- Reserved for the program at start
- Store local variables (auto), function arguments
- stacksize: 8192 kbytes

www.agh.edu.pl

Heap

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

int main(){
   int *p = new int;
   *p = 10;

   cout << *p << endl;

   delete p;
}
```

- Dynamic memory management
- new - memory allocation
- delete - deallocation (release memory)
- new returns pointer of the reserved type
- C++ do not have a garbage collector, need to explicite free resources (RAM)
- OS automatically frees memory after program finishes
- No release unused memory is a mistake !!! Leads to **memory leak**

New operator

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

int main(){
   int x = 10;
   int *p;

   if (x > 5) {
      p = new int;
      *p = x*10;
   }

   cout << *p << endl;

   delete p;
}
```

- Reserved memory does not automatically release after leaving the block
- Failure to take care of memory is causing the so-called "Memory leak"
- Remember, deallocation memory is the responsibility of developer!!!

New operator

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

int main(){
   int *p;

   p = new int;
   if (p==NULL) {
      cout << "no memory!!";
   }

   delete p;   // p == NULL
}
```

- In older systems (OS), the new operator returns NULL if memory could not be reserved
- In modern systems, OS throw exception instead returns NULL, so checking it is pointless
- If p == NULL, you can safely delete p;

www.agh.edu.pl

Operator new - arrays

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

int main(){
   int size = 100;
   int *p = new int[size];

   for (size_t i = 0; i < size; ++i) {
       p[i] = i;
   }

   delete [] p;
}
```

- Dynamic declaration of the array
- Addressing as in the table
- Releasing an array

www.agh.edu.pl

Memory allocation in "C"

```c
#include <stdlib.h>

int main(){
   int size = 100;
   int *p = (int *)malloc(size, sizeof(int));

   for (size_t i = 0; i < size; ++i) {
      p[i] = i;
   }

   free(p);
}
```

- In C there is a pair of functions:
  - malloc(...)
  - free(...)
- The malloc function returns the type *void
- It should be cast to the right type

www.agh.edu.pl

Thank you