

Podstawy informatyki

Katedra Telekomunikacji, EiT

dr inż. Jarosław Bułat

kwant@agh.edu.pl

Plan prezentacji

- » Ankieta
- » IDE
- » Wskaźniki
- » Tablice i wskaźniki
- » Arytmetyka wskaźników
- » Napisy w tablicach - `char[]`
- » typ `String`
- » Wskaźnik na strukturę
- » Rozmiar wskaźnika
- » Dynamiczne zarządzanie pamięcią (stos/sterta)

Ankieta

- » Jakie są najtrudniejsze dla Was elementy labu/wykładu (teraz, nie te które już zostały rozwiązane) **przykład: wskaźniki, pętle, środowisko programistyczne, “lab na wygnaniu ;-”), zbyt małe litery na prezentacji, etc...**
- » Którego zagadnienie było zbyt trudne i należy je powtórzyć/rozszerzyć (jeżeli będzie taka możliwość)
- » student(ka) najbliższej drzwi wyjściowych “organizuje” kartkę A4, wpisuje datę, temat ankiety: “najtrudniejsze elementy przedmiotu” i rozpoczyna wypełniać

IDE

- » **I**ntegrated **D**evelopment **E**nvironment
- » Środowisko (program) ułatwiający programowanie:
 - edytor plików źródłowych
 - organizacja projektu (złożony program)
 - kompilacja
 - uruchomienie
 - debugowanie
- » CLion, Visual Studio, XCode, Eclipse, NetBeans IDE, Code::Blocks, Qt Creator, Visual Studio Code, VIM+konsola
~~devGPP~~

IDE

- » Dedykowane dla jednego języka lub **uniwersalne**
 - Eclipse: Java, C/C++, Python, PHP, etc...
- » Windows only (VC) albo **wieloplatformowe**
- » Darmowe - zazwyczaj Open Source albo Community Edition
- » Komercyjne - CLion (jetbrains.com)
 - biznes: **200-100 EUR/rok + VAT**
 - indywidualna: 90-50 EUR/rok
 - studencka: free
- » Proste: Atom, **Visual Studio Code**

IDE - edytor

- » Ułatwia pisanie kodu
- » Podpowiada nazwy zmiennych, argumenty, ...
- » Help - dokumentacja bibliotek, funkcji, środowiska, ...
- » Refactoring
- » Parser informujący o błędach (+ błędy z kompilacji)
- » Koloruje składnie
- » Pomaga formatować kod (indentation)
- » Praca zespołowa - git: diffy, wersje, ...
- » Debugger

IDE - Eclipse

- » **Projekt:** File/New/C++ Project: Executable/Hello... , Toolchains: Linux GCC
 - lab: **każdy program to osobny projekt !!!**
- » **Kompilacja:** Ctrl-b, toolbar: młoteczek
- » **Uruchomienie:** Ctrl-F11, toolbar: play
- » Perspektywy: C/C++, Debug, Team
- » Profile: Debug, Release
- » save (ctr-s) -> compile (ctr-b) -> run (ctr-F11)
- » Preferencje projektu
 - linkowanie **libm**: /lib/x86_64-linux-gnu/libm.so.6

Wskaźniki

można je lubić lub nienawidzić :-)

Organizacja pamięci

- » Pamięć jest ciągła
- » Pojedyncza komórka ma **wielkość 8b**
- » Każda komórka ma **unikalny adres**

addr **value**

0x000	
0x001	
0x002	
0x003	
0x004	
0x005	
0x006	
0x007	
0x008	
0x009	
0x00A	
0x00B	
0x00C	

Organizacja pamięci

- » Pamięć jest ciągła
- » Pojedyncza komórka ma **wielkość 8b**
- » Każda komórka ma **unikalny adres**
- » **Zawartość komórki** jest dostępna przez jej **adres**
- » **Dostęp przez adres to jedyny sposób na poziomie sprzętowym**

addr **value**

0x000	
0x001	
0x002	
0x003	
0x004	
0x005	
0x006	
0x007	
0x008	
0x009	
0x00A	
0x00B	
0x00C	

Organizacja pamięci

- » Pamięć jest ciągła
- » Pojedyncza komórka ma **wielkość 8b**
- » Każda komórka ma **unikalny adres**
- » **Zawartość komórki** jest dostępna przez jej **adres**
- » **Dostęp przez adres to jedyny sposób na poziomie sprzętowym**
- » Zmienna **char c=48;** znajduje się w jednej komórce pamięci (jej wartość)
 - program ma dostęp do niej przez **nazwę** albo **adres**
 - Nazwa zmiennej **“c”** istnieje tylko w programie !!!

addr	value
0x000	
0x001	
0x002	
0x003	
0x004	
0x005	
0x006	
0x007	
0x008	48
0x009	
0x00A	
0x00B	
0x00C	

Organizacja pamięci

- » Zmienne o rozmiarze $>1\text{B}$ są przechowywane w kolejnych adresach (w ciągłej przestrzeni)
- » **int x = 12578329; // 0xBFEE19**

Następny wolny adres ->

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

Organizacja pamięci

- » Zmienne o rozmiarze >1B są przechowywane w kolejnych adresach (w ciągłej przestrzeni)
- » **int x = 12578329;** // 0xBFEE19
- » **char tab[2];**
 - **tab[0] = 'a';**
 - **tab[1] = 'b';**

addr	value
0x000	
0x001	
0x002	
0x003	0x19
0x004	0xEE
0x005	0xBF
0x006	0x00
adres tab[0] -> 0x007	'a'
adres tab[1] -> 0x008	'b'
0x009	
0x00A	
0x00B	
0x00C	



quiz

PI06_mem

socrative.com

- login
- student login

Room name:

KWANTAGH

Wskaźniki

- » Wskaźnik to jest zmienna której wartością jest adres innej zmiennej (**zmienna która “wskazuje” inną zmienną**)
- » Wskaźnik który nie wskazuje innej zmiennej jest *niezainicjalizowany*

int *x; deklaracja **wskaźnika** **do int** (do typu int)

x jest typu **“wskaźnik do int”**

x = &y; operator **&** to **pobranie adresu** zmiennej y
do x przypisuje się **adres a nie jej wartość!!!**

int z = *x; **wyłuskanie** (ang. dereference) wartości zmiennej

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

Wskaźniki

```

» int x = 12578329;    // 0xBFEE19
» int *y = &x;         // inicjalizacja

» x == 12578329        typ: int
» &x == 0x003           typ: wskaźnik do int
» y == 0x003           typ: wskaźnik do int
» *y == 12578329       typ: int

```

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

Wskaźniki

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

`x = y;` błąd, próba przypisania adresu do zmiennej `int`
`y = x;` błąd, próba przypisania wartości `int` do adresu
`cout << *x;` błąd, próba wyluskania na zmiennej a nie na wsk.
`y = &z;` błąd, typy się nie zgadzają `int* != char*`

Wskaźniki

```

» 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!!!

```

addr	value	
0x000		
0x001		
0x002		
0x003	'a'	} zmienna z
0x004	'b'	
0x005	'c'	
0x006	'd'	
0x007		
0x008		
0x009		
0x00A		
0x00B		
0x00C		



quiz

PI06_point1

socrative.com

- login
- student login

Room name:

KWANTAGH

Wskaźniki: pobranie adresu

```
#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” zrozumie, że typ `int*` należy wypisać jako adres

Wskaźniki: wyłuskanie

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

```
int main() {
```

```
    int x = 4;
```

```
    int *y;           // pointer to int
```

```
    y = &x;           // address-of operator
```

```
    *y = 6;            // wyłuskanie
```

```
    cout << x << endl; // 6
```

```
}
```

» Operacja wyłuskania jest read/write:

- można przeczytać wartość wskazywanej zmiennej
- można też zmienić taką zmienną

Wskaźniki: pobranie adresu

```
#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
```

```
}
```

» Wskaźnik jest zmienną więc można ją zmieniać :)

- raz wskazuje na **x0**
- innym razem na **x1**

» Wypisuję wartości różnych zmiennych za pomocą jednego y !!!

Wskaźniki: pobranie adresu

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

```
int main() {
```

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

```
    y0 = &x;
    y1 = y0;
```

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

```
}
```

» **Wskaźnik jest** zmienną więc mogę przypisać jej wartość (adres zmiennej x) do innego wskaźnika

– &x jest w y0 oraz y1

Wskaźniki: zmiana wartości

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

```
int main() {
```

```
    int x = 4;  
    int *p = &x;
```

```
    cout << x << endl; // 4  
    cout << *p << endl; // 4
```

```
    (*p)++;  
    // *p++; // error
```

```
    cout << x << endl; // 5  
}
```

» Wyluskanie pozwala zmienić wartość zmiennej

Namespace

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

int main() {

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

}
```

- » Namespace to przestrzeń nazw
- » Pozwala mieć takie same nazwy zmiennych/funkcji w jednym programie:
 - `x::test = 7;`
 - `y::test = 8;`



quiz

PI06_point2

socrative.com

- login
- student login

Room name:

KWANTAGH

Przykłady użycia wskaźników

Wskaźniki i tablice

```
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; // ???
```

- » zmienna **tab** jest typu **char***
- » **tab** wskazuje na pierwszy element tablicy

Tablice znaków

```
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)
```

- » Tablica znaków to “napis”
- » Konwencja w języku C
- » Ostatni znak to znak końca napisu ‘\0’
- » Przestarzałe w C++
- » “Hello World!!!” zostanie zapisane w pamięci programu

Zmienna string

```
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;
```

- » Tekst w tablicach to problemy:
 - nie można zmienić długości
 - trudności z “opanowaniem” końca tekstu - **częste ataki**
- » “xxxx” w kodzie źródłowym to **stała napisowa**
- » W C++ tekst w typie **string**
- » Typ string jest elastyczny i bezpieczniejszy
- » String to część biblioteki standardowej

Wskaźnik na struktury

```
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;    // ??
```

```
}
```

- » Wskaźnik na strukturę działa tak samo jak wskaźnik na zmiennej
- » Adresowanie struktur:
 - operator . dla zmiennych
 - operator -> dla wskaźników

Wskaźnik do struktury w tablicy

```
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;  
}
```

» Pobranie adresu pojedynczego elementu tablicy tak samo jak pobranie adresu zmiennej

Rozmiar wskaźnika

```
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;    // ??
```

```
}
```

- » Wskaźnik `pp` to nie jest kopia zmiennej `prod` tylko jej adres
- » Rozmiar wskaźnika jest stały, nie zależy od rozmiaru zmiennej na którą wskazuje

Wskaźnik na nieistniejący obiekt

```
#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 (ale bez sensu)
    cout << *x << endl; // błąd !!!
```

```
}
```

- » Warunek: wskaźnik otrzymuje adres zmiennej, która przestanie istnieć
- » *x to dostęp do pamięci, którą zajmowała zmienna, która przestała istnieć!
- » Jeśli $y \leq 5$, wskaźnik niezainicjalizowany i próba użycia !!!

Wskaźnikowa Incepcja

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

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

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

```
}
```

- » Wskaźnik jest zmienną
- » Mogę pobrać adres wskaźnika
- » Uzyskam wskaźnik, na wskaźnik na zmienną.



quiz

PI06_point3

socrative.com

- login
- student login

Room name:

KWANTAGH

Co oznacza inkrementacja wskaźnika

czyli arytmetyka wskaźników

Arytmetyka na wskaźnikach

- » Wskaźnik to adres pamięci
- » Inkrementacja wskaźnika zwiększa adres o `sizeof(typ)`
- » `int *p = &x; p++;` zwiększy wartość `p` o `sizeof(int)`
- » Wszystkie operacje arytmetyczne na wskaźnikach są zmianami o `sizeof(typ)`
- » Używa się prawie wyłącznie w połączeniu z tablicami

Nazwa tablicy to adres

```
#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
}
```

» Nazwa tablicy to wskaźnik na pierwszy element tablicy

Adresowanie tablicy wskaźnikiem

```
#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
}
```

- » kolejne adresy tablicy typu **int** **+=4 bajty**
- » ***p+1**, kolejność !!!

Adresowanie tablicy wskaźnikiem

```
#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;
}
```

- » Zmiana wskaźnika na kolejne elementy
- » Wyłuskanie zawartości (kolejnej komórki tablicy)
- » Rezultat:
ala ma kota

Adresowanie tablicy wskaźnikiem

```
#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;
}
```

- » Można rozpocząć “przeoglądanie” tablicy od dowolnego miejsca
- » Od którego znaku rozpocząłem wypisywanie?
- » Jaki błąd jest w pętli?

Adresowanie tablicy wskaźnikiem

```
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`
- » Wskaźnikiem można przeskakiwać co kilka elementów
- » Wskaźnik w wyrażeniu arytmetycznym

Adresowanie tablicy wskaźnikiem

```
#include <iostream>
```

```
using namespace std;
```

```
int main(){
```

```
    char tab[] = "ala ma kota";
```

```
    char *p = tab;
```

```
    while ( *p ) {
```

```
        cout << *p++; // *(p++)
```

```
    }
```

```
    cout << endl;
```

```
}
```

```
// while( p ) {} jaki błąd?
```

- » Napis w konwencji "C" to sekwencja znaków + znak końca sekwencji '\0' czyli 0
- » Pętla wykorzystuje '\0' jako warunek wyjścia (testowana wartość wyłuskania a nie wsk.)
- » Inkrementacja ma wyższy priorytet niż wyłuskanie więc będzie dotyczyć wskaźnika ale wykona się po wyłuskaniu !!!
- » **Niezalecane**

Adresowanie tablicy wskaźnikiem

```
#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;
}
```

- » `tab` to adres, więc można nią indeksować tablicę
- » `tab` to stała, więc nie można jej zmieniać!!!
- » `ex26.cc:9:12: error: lvalue required as increment operand`



quiz

PI06_point4

socrative.com

- login
- student login

Room name:

KWANTAGH

Odejmowanie wskaźników

```
#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 (a nie 2*sizeof(int))
```

```
}
```

» Odejmowanie wskaźników daje
wynik w:

krotnościach sizeof(typ)

Porównywanie wskaźników

```
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
```

» Pętla zamienia kolejność elementów w tablicy

Tablica wskaźników

```
#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
```

```
}
```

» Deklaracja tablicy, której każdy element jest:

wskaźnikiem na typ `int`

» np: tablica 2D, pierwsza kolumna zawiera wskaźniki na początek każdego wiersza

– każdy wiersz może być różnej długości

Arytmetyka wskaźników tylko na tablicy

```
#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
}
```

- » Arytmetyka wskaźników tylko w odniesieniu do tablic
- » Czego ja oczekuję po tym kodzie? że “przeskoczę” na następną zmienną?
- » **Never ever!!!**

Arytmetyka wskaźników tylko na tablicy

```
#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+'  
}
```

» Dodaję dwa adresy

» To jest tak głupie, że mnie kompilator wyśmieję ;-)

Implementacja tablic

```
#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;   // !
```

```
}
```

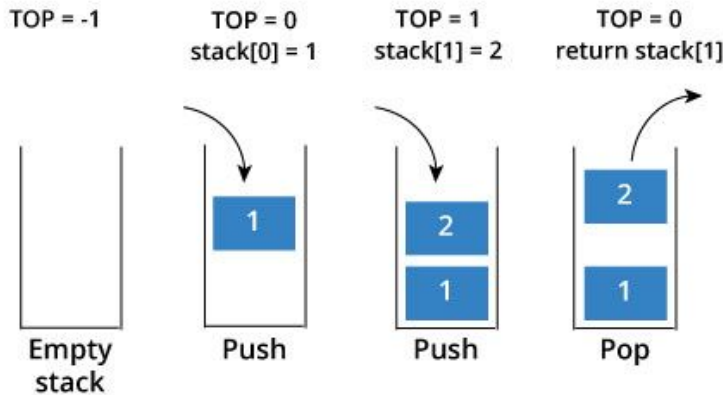
- » Indeksowanie tablic jest zaimplementowane za pomocą arytmetyki na wskaźnikach

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

Stos/Sterta

dynamiczne zarządzanie pamięcią

Stos (*ang.* Stack)



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

- » Liniowa struktura danych
- » Bufor typu **LIFO** (Last In, First Out)
- » `push()`, `pop()`, `isEmpty()`
- » opetentowany w 1957

- » Pamięć jest wspólna dla wszystkich programów
- » Rezerwacja pamięci wymaga* wstrzymania multitaskingu
- » Stos jest implementowany sprzętowo w CPU
- » Zarezerwowany dla programu
- » Przechowuje zmienne lokalne (automatyczne), argumenty funkcji
- » `stacksize: 8192 kbytes`

Sberta (*ang.* heap)

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

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

    cout << *p << endl;

    delete p;
}
```

- » Dynamiczne zarządzanie pamięcią
- » **new** - rezerwacja pamięci
- » **delete** - zwolnienie pamięci
- » **new zwraca wskaźnik na rezerwowany typ**
- » C++ nie ma garbage collector, należy explicitie zwolnić zasoby (RAM)
- » OS zwolni automatycznie pamięć po zakończeniu programu
- » **Brak zwolnienia nieużywanej pamięci jest błędem !!!**

Operator new

```
#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;
}
```

- » Zarezerwowana pamięć **nie zwalnia się “sama”** po wyjściu z bloku
- » Brak dbania o zwalnianie pamięci powoduje tzw. “wyciek pamięci”
- » **Pamięć należy zwalniać !!!**

Operator new

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

int main(){
    int *p;

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

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

- » W starszych systemach **operator new** zwraca NULL jeżeli nie udało się zarezerwować pamięci
- » We współczesnych systemach będzie wyjątek więc sprawdzanie jest bezcelowe
- » jeżeli **p == NULL**, można bezpiecznie wykonać **delete p**;

Operator new - tablice

```
#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;
}
```

- » Dynamiczna deklaracja tablicy
- » Adresowanie jak w tablicy
- » Zwalnianie tablicy

Rezerwowanie pamięci w “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);  
}
```

» W języku C jest para funkcji:

– malloc(...)

– free(...)

» Funkcja malloc zwraca typ
*void

» Należy rzutować na właściwy
typ

Dziękuję