

# Wykład 7: Tablice jedno i wielowymiarowe. Wskaźniki, arytmetyka wskaźników.

dr inż. Andrzej Stafiniak

*Wrocław 2023*



HR EXCELLENCE IN RESEARCH



Politechnika Wroclawska

# Podstawowe informacje o tablicach

- **Tablica** (ang. array) jest to pochodna (nie prosta/fundamentalna), złożona struktura danych.
- Stanowi ona ciąg wartości, ułożonych w sposób liniowy w pamięci komputera (jedna po drugiej) oraz posiadających wspólny identyfikator (**nazwę tablicy**).
- Jest to ciąg wartości/zmiennych tego **samego typu**.
- Nazwa tablicy, jej identyfikator, jest **wskaźnikiem adresu** na jej pierwszy element.
- Tablice mogą być jedno- lub wielowymiarowe.
- Pamięć pod tablice może być alokowana statycznie (**tablica statyczna**) lub dynamicznie (**tablica dynamiczna**) - wiąże się z tym różny obszar zajmowanej przez tablice pamięci.

# Tablica jednowymiarowa

- W celu dokonania deklaracji tablicy musimy użyć **typu**, jaki będzie przechowywać tablica, **nazwy** tablicy oraz operatora indeksu **[]** wraz z **rozmiarem** tablicy, czyli wartością całkowitą odpowiadającą liczbie elementów tablicy. Np.: `char tab[9];`
- Składnia deklaracji statycznej tablicy jednowymiarowej jest następująca:

`typ nazwaTablicy[rozmiar];`

- W przypadku tablic statycznych ich rozmiar musi być wartością stałą, znaną na etapie kompilacji programu.

# Tablica jednowymiarowa

```
#define NR 150
```

```
char tab[9]; //deklaracja 9-cio elementowej tablicy char'ów
```

```
int studIndex[6];
```

```
float ocena[NR];
```

Inicjalizacja tablicy - przypisanie wartości do poszczególnych komórek tablicy:

```
int nr=5;
```

```
scanf("%i", &nr);
```

```
float ocena[nr];
```

```
const int NR =5;
```

```
float ocena [NR];
```

```
tab[0] = 'S';
```

```
tab[1] = 't';
```

```
...
```

```
tab[8] = '!' ;
```

```
for(int i=0;i<6;++i){  
    studIndex[i]=i;
```

```
}
```

```
scanf("%i", &studIndex[i]);
```

komórka

S

t

o

p

w

a

r

!

indeks

0

1

2

3

4

5

6

7

8

```

#ifdef  _STDC_NO_VLA_
    printf("VLA not supported");
#else
    printf("VLA supported");
#endif

```

inicjalizacja tablicy - przypisanie wartości do poszczególnych komórek tablicy:

# C89 v. C11 ?

komórka	S	t	o	p		w	a	r	!
indeks	0	1	2	3	4	5	6	7	8

# Tablica jednowymiarowa - inicjalizacja

Deklaracja tablicy z jednoczesną inicjalizacją - definicja

```
#define NR 150
```

```
char tab[9]={'S','t','o','p',' ','w','a','r','!'}
```

```
int studIndex[6]= {1, 2, 3, 4, 5, 6}
```

Postacie równoważne, automatycznie zostanie dobrany rozmiar tablicy przez kompilator, w zależności od ilości oddzielnych elementów inicjujących.

```
char tab[]={ 'S','t','o','p',' ','w','a','r','!' }
```

```
int studIndex[]= {1, 2, 3, 4, 5, 6}
```

komórka

<b>S</b>	<b>t</b>	<b>o</b>	<b>p</b>		<b>w</b>	<b>a</b>	<b>r</b>	<b>!</b>
----------	----------	----------	----------	--	----------	----------	----------	----------

indeks

0	1	2	3	4	5	6	7	8
---	---	---	---	---	---	---	---	---

# Tablica jednowymiarowa - inicjalizacja

- Komórki, które nie będą zainicjalizowane jawnie otrzymają wartość zero.

```
float ocena[NR] = {4.5, 5.0} // ( 4.5, 5.0, 0, 0, 0 ... )
```

- Inicjalizacji następują kolejne, począwszy od pierwszej, komórki tablicy. Istnieje możliwość podania wartości konkretnej komórki po przez wykorzystanie tak zwanej oznaczonej inicjalizacji.

```
float ocena[NR] = {4.5, [3]=5.0} // ( 4.5, 0, 0, 5.0, 0, 0 ... )
```

# Tablica jednowymiarowa - inicjalizacja

```
#include <stdio.h>
```

```
int main(void)
{
    int tab[10];
    int tab1[10] = {[7] = 10};
    float tab2[100] = {5, 9, [7] = 10, 44, 66};

    for (int i = 0; i < (sizeof(tab)/sizeof(tab[0])); i++)
        printf("%i ", tab[i]);

    printf("\n\n");
    for (int i = 0; i < (sizeof(tab1)/sizeof(tab1[0])); i++)
        printf("%i ", tab1[i]);

    printf("\n\n");

    for (int i = 0; i < (sizeof(tab2)/sizeof(tab2[0])); i++)
        printf("%.0f ", tab2[i]);

    return 0;
}
```

W taki sposób można szybko sprawdzić ile elementów posiada dana tablica.

[illegible]



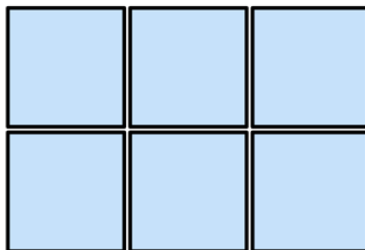
# Tablice wielowymiarowe

- W języku C/C++ istnieje możliwość tworzenia tablic wielowymiarowych.
- Deklaracja tablicy wielowymiarowych odbywa się za pomocą wykorzystania kolejnych par nawiasów []

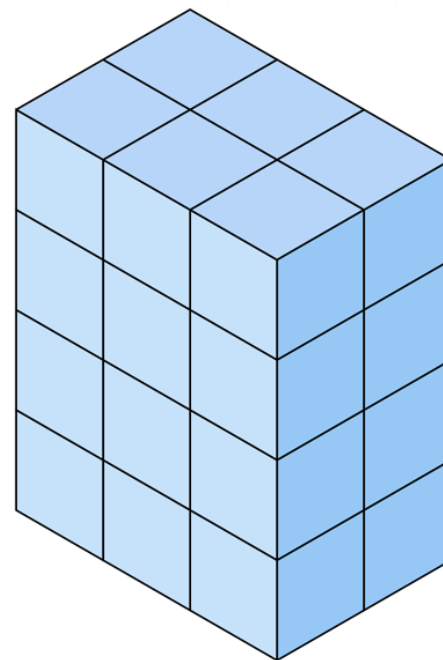
Tablice 1-wymiarowa – 1D



Tablice 2-wymiarowa – 2D



Tablice 3-wymiarowa – 3D



# Tablice wielowymiarowe - inicjalizacja

- Deklaracja **tablicy wielowymiarowych** odbywa się za pomocą wykorzystania dodatkowo kolejnych pary nawiasów []. Natomiast inicjalizacja takich tablic może odbywać się również za pomocą nawiasów klamrowych lub pętli.

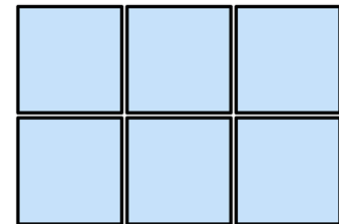
```
int array2D[2][3]={  
    {11, 12, 13},  
    {21, 22, 23}  
}
```

```
int array2D[2][3];  
for(int i=0, i<2;++i)  
    for (int j=0; j<3;++j)  
        tab2D[i][j] = i+j;
```

- Automatyczne dedukcje rozmiaru tylko dla pierwszego wymiaru:

```
int array2D[][3]={  
    {11, 12, 13},  
    {21, 22, 23}  
}
```

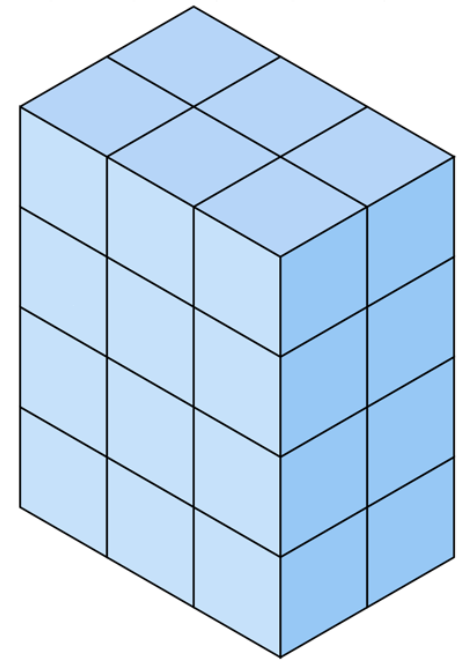
Tablice 2-wymiarowa – 2D



# Tablice wielowymiarowe - inicjalizacja

```
int array2D[][3][4]={  
    {  
        {000, 001, 002, 003},  
        {010, 011, 012, 013},  
        {020, 021, 022, 023}  
    },  
    {  
        {100, 101, 102, 103},  
        {110, 111, 112, 113},  
        {120, 121, 122, 123}  
    }  
}
```

Tablice 3-wymiarowa – 3D



# Tablice wielowymiarowe – wyświetlanie

```
#include <stdio.h>

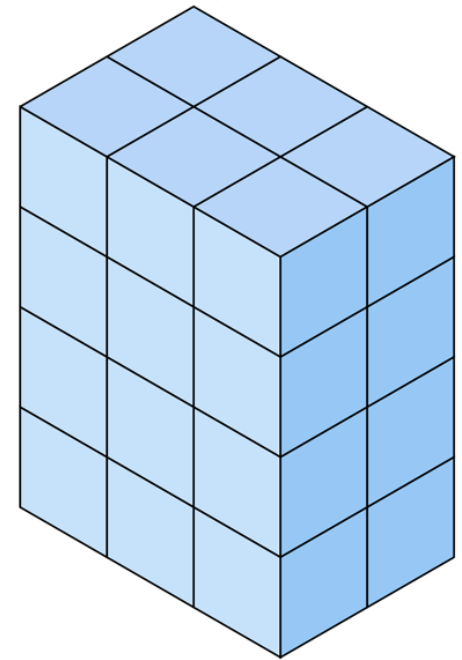
int main(void)
{
    int tab3D[3][4][5];    // [i][j][k]
    int i,j,k;

    for (i =0; i<3;i++){
        for (j=0; j<4;j++){
            for (k=0; k<5;k++)
                tab3D[i][j][k] = i+j+k;
        }
    }

    for (i =0; i<3;i++){
        printf("\n");
        for (j=0; j<4;j++){
            printf("\n");
            for (k=0; k<5;k++)
                printf("%i ", tab3D[i][j][k]);
        }
    }

    printf("\n");
    return 0;
}
```

Tablice 3-wymiarowa – 3D



```
0 1 2 3 4
1 2 3 4 5
2 3 4 5 6
3 4 5 6 7

1 2 3 4 5
2 3 4 5 6
3 4 5 6 7
4 5 6 7 8

2 3 4 5 6
3 4 5 6 7
4 5 6 7 8
5 6 7 8 9
```

# Tablice wielowymiarowe – przeszukiwanie

```
#include <stdio.h>

int main() {
    int buff=0, n=0;
    int array[]={9, 15, 12, 33, 20, 7};
    n=sizeof(array)/sizeof(array[0]);

    for(int i=0; i<n; ++i){
        printf("%d, ", array[i]);
    }

    for(int i=0; i<n; i++){
        if (buff<array[i])
            buff = array[i];
    }

    printf("\nMax=%d", buff);
    return 0;
}
```

```
9, 15, 12, 33, 20, 7,
Max=33
```

# Wskaźniki

- **Wskaźnik** (*pointer*) to nazwa typu danych pochodnych (nie prostych/wbudowanych). Jako wartość, **zmienne wskaźnikowe** przechowują **adresy pamięci** innych stałych, zmiennych lub funkcji.
- **Adres pamięci** to pewien unikalny identyfikator, pozwalający na odnalezienie lokalizacji każdego elementu/zasobu w obszarze pamięci komputera.
- **Zmienne wskaźnikowe** również posiadają swój adres, a rozmiar przestani w pamięci komputera zajmowany przez typ wskaźnikowego zależy od architektury komputera, dla systemów 32-bitowych są to 4B, dla 64-bitowych – 8B.

# Wskaźniki

Wykorzystanie **zmiennych wskaźnikowych** w języku C/C++ umożliwia:

- pracę na oryginałach zmiennych (np. argumenty funkcji)
- dynamiczną alokację pamięci
- optymalizację czasu wykonania programu.

# Wskaźniki

- Deklaracja **zmiennej wskaźnikowej** odbywa się przez podanie **typu** zasobu, na który wiązuje wskaźnik, operatora dereferencji **\*** oraz **nazwy** zmiennej wskaźnikowej.

```
type * namePointer;
```

- Aby zainicjalizować zmienną wskaźnikową (zdefiniować), należy przypisać jej adres innej zmiennej. Możemy wykonać to za pomocą operatora **pobrania adresu &**.

```
type someVar = value;  
type * namePointer = &someVarr;
```

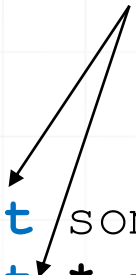


# Wskaźniki

➤ Przykład:

Typ **zmiennnej wskaźnikowej** jest taki sam jak zmiennnej, na którą wskazuje wskaźnik.

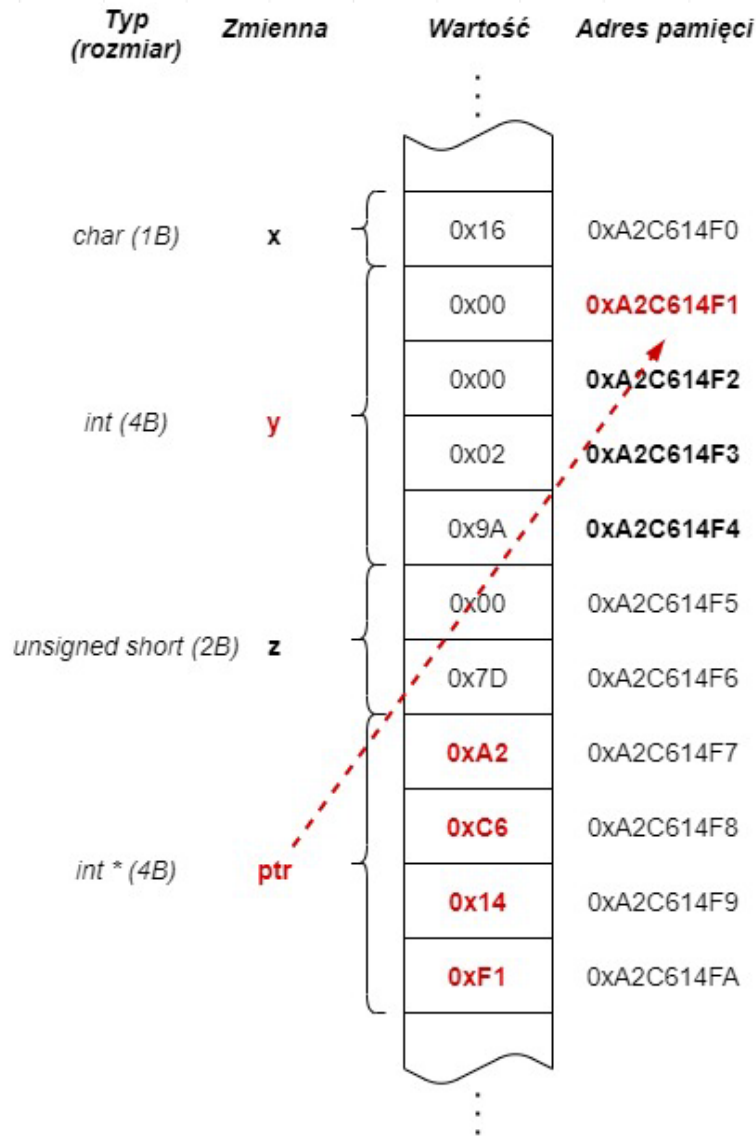
```
int someVar = 303;  
int * ptr1 = &someVar;
```



```
int anotherVar = 304;  
int * ptr2;  
ptr2 = &anotherVar;
```

```
char tab[]={ "Stop war!!!" };  
char ptr3 = tab;
```

# Wskaźniki



```
int y = 1;
```

```
int * ptr;
```

```
ptr = &y;
```

Wizualizacja fragmentu pamięci operacyjnej komputera (np. stos) w **32-bitowym** systemie operacyjnym.

Elementarna komórka pamięci którą można **adresować** posiada rozmiar czyli 1 bajt (1B) czyli 8 bitów (8b).

Wskaźnik **prt** wskazuje na adres pierwszej z czterech komórek pamięci, na których zapisana jest zmienna **y** typu **int** (4B). Dlatego ważne jest, aby **typ** wskaźnika odpowiadał **typowi** zmiennej, ponieważ poza **adresem** należy przekazać również informację **ile bajtów** należy pobrać.

Adresy i zapisane dane w pamięci wyraża się w systemie szesnastkowym

# Wskaźniki

```
#include <stdio.h>

int main () {
    int var1 = 11;
    double var2 = 1.2;
    char tab[] = "Stop war!!!";

    char * ptrChr = tab;    ptrChr = &tab[3];
    int * ptrInt = &var1;

    printf("var1 - %d \n", var1);
    printf("var2 - %.2f \n", var2);
    printf("*ptrInt - %d \n", *ptrInt);

    double * ptrDou;
    ptrDou = &var2;
    *ptrDou += 1.3;

    printf("\n");
    printf("tab - %c \n", tab[0]);
    printf("tab /s - %s \n", tab);
    printf("tab /c - %c \n", tab);
    printf("tab - %d \n", tab);
    printf("\n");

    printf("*ptrChr - %c \n", *ptrChr);
    printf("ptrChr /s - %s \n", ptrChr);
    printf("ptrChr /c - %c \n", ptrChr);
    printf("ptrChr - %d \n", ptrChr);
    printf("\n");
    printf("** (ptrChr+5) - %c \n", *(ptrChr+5));

    return 0;
}
```

- Aby odwołać się do **wartości zmiennej** wskazywanej przez **adres** zapisany jako **wartość zmiennej wskaźnikowej** należy posłużyć się **operatorem dreferencji**, nazywanym również **operatorem wyłuskania** czyli **\***.

```
var1 - 11
var2 - 1.20
*ptrInt - 11

tab - S
tab /s - Stop war!!!
tab /c - Ě
tab - 6422268

*ptrChr - S
ptrChr /s - Stop war!!!
ptrChr /c - Ě
ptrChr - 6422268

*(ptrChr+5) - w
```

# Wskaźniki

## Główna zasada:

**Nie rób dereferencji na niezainicjalizowanym wskaźniku, jeśli życie Ci miłe!**

```
int main () {  
  
    int var1 = 11;  
    int * ptrInt1 = &var1;  
    int * ptrInt2;  
        // *ptrInt2 = 31415 nie rób tak!  
  
    printf("Wartosc var1 - %d \n", var1);  
    printf("Adres &var1 - %d \n", &var1);  
    printf("Wartosc wyluskana *ptrInt1 - %d \n", *ptrInt1);  
    printf("Wartosc wskaznika ptrInt1 - %d \n", ptrInt1);  
    printf("Adres wskaznika &ptrInt1 - %d \n", &ptrInt1);  
  
    printf("\n");  
    printf("Wartosc wskaznika ptrInt2 - %d \n", ptrInt2);  
    printf("Adres wskaznika &ptrInt2 - %d \n", &ptrInt2);  
    printf("Wartosc wyluskana *ptrInt2 - %d \n", *ptrInt2);  
  
    return 0;  
}
```

```
Wartosc var1 - 11  
Adres &var1 - 6422300  
Wartosc wyluskana *ptrInt1 - 11  
Wartosc wskaznika ptrInt1 - 6422300  
Adres wskaznika &ptrInt1 - 6422296  
  
Wartosc wskaznika ptrInt2 - 13044984  
Adres wskaznika &ptrInt2 - 6422292  
Wartosc wyluskana *ptrInt2 - 13045240
```

# Wskaźniki

## Główna zasada:

- nie rób dereferencji na niezainicjalizowanym wskaźniku, jeśli życie Ci miłe!

**W celu sprawdzenia czy wskaźnik jest zainicjalizowany adresem, a nie losową wartością, wprowadzono:**

- makro **NULL** (o wartości 0, typu int) do inicjalizacji początkowej wskaźnika, nie wskazujące na żaden adres (języku C).
- słowo kluczowe **nullptr** zdefiniowane jako wartość wskaźnika niewskazującego na żaden adres zasobu w pamięci (język C++).

```
int * ptrInt = NULL;  
if (ptrInt != NULL)
```

# Wskaźniki

## Główna zasada:

- nie rób dereferencji na niezainicjalizowanym wskaźniku, jeśli życie Ci miłe!

Istnieje również możliwość zastosowanie „stałej wskaźnikowej” z wykorzystaniem słowa kluczowego **const**:

- w zależności od konstrukcji definicji wskaźnika, od usytuowania słowa **const**, mamy możliwość zapobiegania nieintencjonalnym modyfikacjom:

- zmiennej wskazywanej przez wskaźnik
- adresu przypisanego do wskaźnika podczas inicjalizacji

```
#include <stdio.h>
```

```
int main() {
```

```
float var = 3.33;
const float * constPtr = &var;
printf("Var: %g", *constPtr);
// Var: 3.33
*constPtr = 7.77;
//błąd kompilacji
```

```
return 0;
```

```
}
```

```
constPtr.c: In function 'main':
constPtr.c:9:12: error: assignment of read-only location '*constPtr'
 *constPtr = 7.77;
   ^
```

```
#include <stdio.h>
```

```
int main() {
```

```
float var = 3.33;
float var2 = 7.77;
float * const constPtr = &var;
```

```
constPtr = &var2;
//błąd kompilacji
```

```
return 0;
```

```
}
```

```
constPtr.c: In function 'main':
constPtr.c:9:11: error: assignment of read-only variable 'constPtr'
 constPtr = &var2;
   ^
```

# Wskaźniki

**„Wskaźniki stanowią jeden z najważniejszych mechanizmów języka C/C++, którego zrozumienie jest niezbędne do wykorzystania pełni możliwości i pojęcia „filozofii” programowania w języku C/C++.”**

# Wskaźniki – arytmetyka wskaźników

- Dodawani oraz odejmowanie liczb całkowitych od wskaźników.

```
char tab[] = "Stop war!!!";  
char * ptrChr1, * ptrChr2;
```

```
ptrChr1 = tab; //ptrChr1 = adres 1 elementu tablicy tab  
ptrChr2 = ptrChr1 + 3; //ptrChr2 ustawiony na 4 element tablicy
```

Należy pamiętać że liczba 3 zostanie wymnożona przez odpowiednią ilość bajtów w zależności od typu tablicy, a następnie wartość ta zostanie dodana do adresu.

wartość	S	t	o	p		w	a	r	!	\0
indeks	0	1	2	3	4	5	6	7	8	9
	↑			↑						
	ptrChr1			ptrChr2						



# Wskaźniki – arytmetyka wskaźników

```
#include <stdio.h>

int main () {

    char tabChr[] = "Stop war!!!";
    char * ptrChr1, * ptrChr2;

    int tabInt[] = {0,1,2,3,4,5,6,7,8,9};
    int * ptrInt1, * ptrInt2;

    ptrChr1 = tabChr;
    ptrChr2 = ptrChr1 + 3;

    ptrInt1 = tabInt;
    ptrInt2 = ptrInt1 + 3;

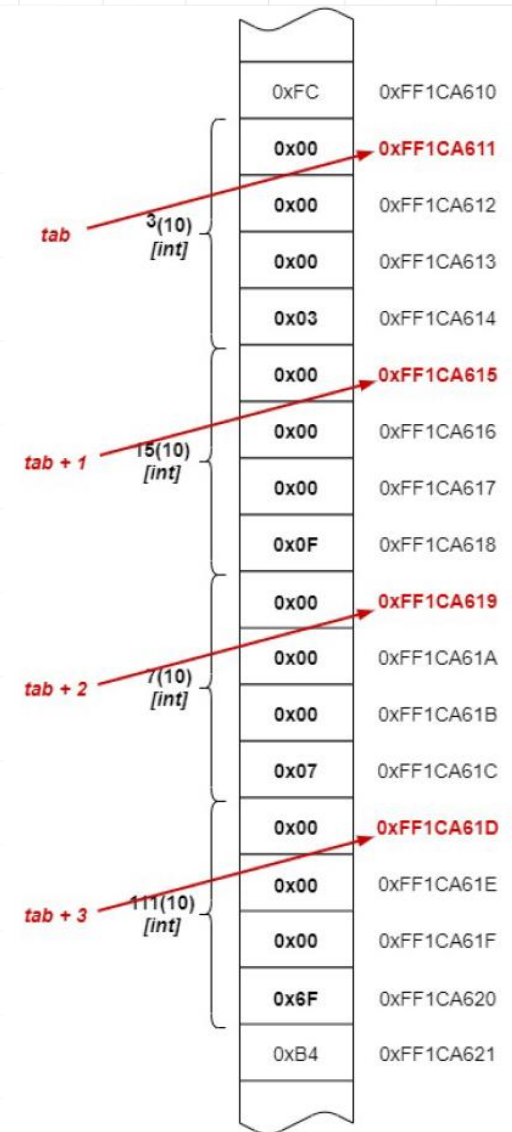
    printf("*ptrChr1 - %c \n", *ptrChr1);
    printf("*ptrChr2 - %c \n", *ptrChr2);
    printf("ptrChr1 - %d \n", ptrChr1);
    printf("ptrChr2 - %d \n", ptrChr2);

    printf("\n");
    printf("*ptrInt1 - %d \n", *ptrInt1);
    printf("*ptrInt2 - %d \n", *ptrInt2);
    printf("ptrInt1 - %d \n", ptrInt1);
    printf("ptrInt2 - %d \n", ptrInt2);

    return 0;
}
```

```
*ptrChr1 - S
*ptrChr2 - p
ptrChr1 - 6422276
ptrChr2 - 6422279

*ptrInt1 - 0
*ptrInt2 - 3
ptrInt1 - 6422236
ptrInt2 - 6422248
```



M. Stępiak, Laboratorium Informatyki



Politechnika Wroclawska

# Wskaźniki – arytmetyka wskaźników

Sortowanie tablicy –  
sortowanie bąbelkowe

```
#include <stdio.h>

int main() {
    int buff, n;
    int array[]={9, 15, 12, 20, 7};
    n=sizeof(array)/sizeof(array[0]);
    int *ptr = array;

    for(int i=0; i<n; ++i){
        printf("%d, ", *(ptr+i));
    }

    for(int i=0; i<n; i++){
        for (int j=i+1; j<n;j++){
            if (*(ptr+j)<*(ptr+i)){
                buff = *(ptr + i);
                *(ptr+i)=*(ptr+j);
                *(ptr+j) = buff;
            }
        }
    }

    printf("\n");
    for(int i=0; i<n; ++i){
        printf("%d, ", array[i]);
    }
    return 0;
}
```

```
9, 15, 12, 20, 7,
7, 9, 12, 15, 20,
```