



Podstawy Programowania

dr inż. Tomasz Marciniak

Wykład 3

- Tablice
- Struktury
- wskaźniki

Tablice

- Tablica to zbiór danych tego samego typu, do których można odwoływać się przez wspólną nazwę. Tablicę deklarujemy w taki o to sposób:

Tablica jednowymiarowa:

typ_danych nazwa_tablicy [rozmiar] = {lista_wartości}

Tablica wielowymiarowa:

typ_danych nazwa_tablicy [lista_rozmiarów] = {lista_wartości}

typ_danych jest typem podstawowym lub zdefiniowanym, **nazwa_tablicy** jest nazwą, taka sama jak dla zwykłych zmiennych, **rozmiar** określa rozmiar tablicy (liczbę elementów) oraz zakres indeksowania,

lista_rozmiarów określa rozmiar tablicy wielowymiarowej i zakresy indeksowania,

lista_wartości jest zbiorem danych początkowych tablicy (inicjalizacja tablicy).

Tablice

- Tablica to zbiór danych tego samego typu, do których można odwoływać się przez wspólną nazwę. Tablicę deklarujemy w taki o to sposób:

Tablica jednowymiarowa:

typ_danych nazwa_tablicy [rozmiar] = {lista_wartości}

Tablica wielowymiarowa:

typ_danych nazwa_tablicy [rozmiar1] [rozmiar2] ... [rozmiarN] = {lista_wartości}

typ_danych jest nazwą, która określa typ danych, które będą przechowywane w tablicy.
nazwa_tablicy jest nazwą, którą użyjemy do odwołania się do tablicy.
rozmiar1, rozmiar2, ..., rozmiarN są liczbami naturalnymi, które określają rozmiar tablicy w każdym wymiarze.
lista_wartości jest listą wartości, które zostaną przypisane do elementów tablicy.

lista_wartości jest zbiorem danych początkowych tablicy (inicjalizacja tablicy).

Inicjowanie tablic

- Mogą należeć do klasy **automatic**, **static** i **external**, ale nie do klasy register,
- za pomocą deklaracji definiującej:

```
int tablica[5] = {2, 3, 4, 5, 6 };
```

```
int tablica[ ] = {2,3,4}
```

jeżeli pominięto rozmiar tablicy, to kompilator obliczy rozmiar na podstawie ilości wprowadzonych danych, umieszczonych pomiędzy nawiasami klamrowymi,

Inicjowanie tablic

- deklarując tablicę jako **static**. Jeżeli nie poda się wartości inicjalizujących, kompilator zainicjalizuje tablice zerami.
- Deklarując tablicę globalną (umieszczoną na zewnątrz wszystkich funkcji). Jeżeli nie poda się wartości inicjalizujących, kompilator zainicjalizuje tablice zerami.

Inicjowanie tablic

- `int tablica[5] = {2, 3, 4, 5, 6 },`
- `int tablica[] = {2,3,4},`
- **`int t[3][4] = {{2, 3, 4}{3, 4, 5}{4, 5, 6}{6, 7, 8}},`**
- **`int t[3][4] = {2, 3, 4, 3, 4, 5, 4, 5, 6, 6, 7, 8};`**

Przykład

```
1  /* tablica jednowymiarowa */
2  #include <stdio.h>
3  #include <conio.h>
4  #define ROZ 20
5  #define KOL 5
6  int main()
7  {
8      int k, x[ROZ];
9      int suma = 0;
10     for (k = 0; k < ROZ; ++k)
11     {
12         x[k] = k;
13         suma += x[k];
14     }
15     printf("\n Lista elementow");
16     for (k=0; k < ROZ; ++k)
17     printf("%c%4d", (k % KOL == 0)? '\n' : '\xB0', x[k]);
18     printf("\nsuma elemntow tablicy = %d", suma);
19     getch();
20     return 0;
21 }
```


Przykład

```
#define ROZ 20  
int k, x[ROZ];
```

Dobrze

```
const int ROZ = 20  
int k, x[ROZ];
```

Źle

Własne typy

- Deklaracja własnych typów z użyciem deklaracji ***typedef***;
- `typedef std_nazwa_typu
własna_nazwa_typu`
- **`typedef unsigned long int uli;
uli liczba1, liczba2;`**


Struktury (structures)

- Struktura to zbiór danych różnych typów, dostępnych pod wspólną nazwą,
- Deklaracja struktury jest następująca:

```
struct nazwa {  
    typ1 nazwa1;  
    typ2 nazwa2;  
    ...  
    typN nazwaN;  
};
```

Deklaracja struktury i zmiennych | sposób

```
struct pacjent      //definicja struktury "pacjent"  
{int nr_badania;    //elementy struktury  
char *imie ;        //elementy struktury  
float koszt;        //elementy struktury  
} pt1,pt2;          //deklaracje zmiennych  
                    //strukturalnych
```



Dostęp do elementów struktury

```
typedef struct          //definiujemy typ pacjent
{ int nr_badania;      //elementy struktury
  char *imie ;         //elementy struktury
  float koszt;         //elementy struktury
} pacjent;
```

```
pacjent pt1,pt2;       //deklarujemy zmienne typu pacjent
```

```
pt1.nr_badania=1234;
pt1.imię=„Adam”;
pt1.koszt=100;
```

Struktury (structures) cd.

- Aby uzyskać dostęp do jednego z pól zadeklarowanej zmiennej, która jest strukturą piszemy po nazwie tej zmiennej kropkę i dalej nazwę pola.

Np.: *zmienna.pole l*;

- Struktury tych samych typów możemy wzajemnie przypisać. **struct** **{int w; int z}** *s1, s2; // deklarujemy zmienne s1 i s2 typu, któremu nie nadajemy nazwy*

s1.w = 5; s1.z = 6; *// przypisujemy nowe wartości polom w i z zmiennej s1*

s2 = s1; *// teraz pola w i z zmiennej s2 mają wartości odpowiednio 5 i 6*

- Można też deklarować tablice struktur co jest bardzo przydatne podczas tworzenia baz danych.

struct s{int l, char z};

struct s zm [100]; *// tworzymy stuelementową tablicę o nazwie zm struktur s.*

Struktury (structures) cd.

- Możliwe jest przekazywanie struktur przez funkcję. Jeśli struktura jest zadeklarowana globalnie i ma przypisaną nazwę:
- **struct** nowa_struktura{**int** a; **int** b};
- **void** fun(**struct** nowa_struktura tutejsza_struktura) {
/* operacje na tutejsza_struktura */
int main(**void**)
{
 struct nowa_struktura c;
 c.a = 3; c.b = 4;
 fun(c); // wywołanie funkcji z parametrem, który jest strukturą
}
- Przy pisaniu programów w języku C++ lepiej w miejscu struktur stosować klasy, które mogą pełnić wszystkie funkcje struktury, dodatkowo dają inne możliwości.

Wskaźniki (*pointers*)

- Wskaźniki to inaczej adres obszaru pamięci komputera. Np.: wskaźnik **a** wskazuje na zmienną **b**, znajdującą się w komórce pamięci **1201**, wtedy, gdy ma przypisaną wartość **1201**. Deklaracja wskaźników jest następująca:

typ *nazwa;

Operatory wskaźnikowe

Istnieją operatory wskaźnikowe:

$w = \&z$ przypisuje wskaźnikowi **w** adres zmiennej **z** .

$z = *w$ przypisuje zmiennej **z** wartość zmiennej na jaką wskazuje **w** .

$w1 = w2$ przypisuje wskaźnikowi **$w1$** adres tej samej zmiennej, na jaką wskazuje wskaźnik **$w2$** .

$w++$ oraz ***wskaznik--*** odpowiednio zwiększa lub zmniejsza adres zmiennej, na jaką wskazuje **w** o 1.

$w+=n$ oraz ***wk-=n*** odpowiednio zwiększa lub zmniejsza adres zmiennej, na jaką wskazuje **w** o **n** bajtów.

$w1==w2$, $w1!=w2$, $w1<w2$, $w1>w2$, $w1<=w2$, $w1>=w2$ odpowiednio porównuje adresy wskazywane przez poszczególne wskaźniki **$w1$** i **$w2$** .

Wskaźniki (*pointers*) cd.

- Można też deklarować wskaźniki do wskaźników wskaźników i tak dalej, ale rzadko jest taka potrzeba. Należy pamiętać, że programy stają się wtedy bardzo nieczytelne, i często występują błędy, które mogą nawet zachwiać stabilność systemu.

Adres i rozmiar zmiennej

- ***Sizeof(*nazwa_typu*)*** – operator rozmiaru;
- ***&*** - operator adresu.

Przykład

```
1  #include <stdio.h>
2  #include <conio.h>
3  int main(){
4  char x1;
5  short int x2;
6  int x3;
7  float x4;
8  double x5;
9  x1 = 1;
10 x2 = 10;
11 x3 = 100;
12 x4 = 15.15;
13 x5 = 74.85;
14 printf("\nzmienna x1 = %6d, adres = %8u, %2i bajty",
15 x1,&x1,sizeof(x1));
16 printf("\nzmienna x2 = %6d, adres = %8u, %2i bajty",
17 x2,&x2,sizeof(x2));
18 printf("\nzmienna x3 = %6d, adres = %8u, %2i bajty",
19 x3,&x3,sizeof(x3));
20 printf("\nzmienna x4 = %6.2f, adres = %8u, %2i bajty",
21 x4,&x4, sizeof(x4));
22 printf("\nzmienna x5 = %6.2f, adres = %8u, %2i bajty",
23 x5,&x5, sizeof(x5));
24 getch();
25 return 0;
26 }
```

Przykład

```
1 #include <stdio.h>
2 #include <conio.h>
3 int main(){
4     char x1;
```

```
zmienna x1 =      1, adres = 2686791, 1 bajty
zmienna x2 =     10, adres = 2686788, 2 bajty
zmienna x3 =    100, adres = 2686784, 4 bajty
zmienna x4 =   15.15, adres = 2686780, 4 bajty
zmienna x5 =   74.85, adres = 2686768, 8 bajty
```

```
11     x3 = 100;
12     x4 = 15.15;
13     x5 = 74.85;
14     printf("\nzmienna x1 = %6d, adres = %8u, %2i bajty",
15     x1,&x1,sizeof(x1));
16     printf("\nzmienna x2 = %6d, adres = %8u, %2i bajty",
17     x2,&x2,sizeof(x2));
18     printf("\nzmienna x3 = %6d, adres = %8u, %2i bajty",
19     x3,&x3,sizeof(x3));
20     printf("\nzmienna x4 = %6.2f, adres = %8u, %2i bajty",
21     x4,&x4, sizeof(x4));
22     printf("\nzmienna x5 = %6.2f, adres = %8u, %2i bajty",
23     x5,&x5, sizeof(x5));
24     getch();
25     return 0;
26 }
```

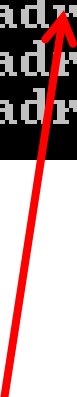
Przykład

```
zmienna x1 =      1, adres = 2686791,  1 bajty  
zmienna x2 =     10, adres = 2686788,  2 bajty  
zmienna x3 =    100, adres = 2686784,  4 bajty  
zmienna x4 =   15.15, adres = 2686780,  4 bajty  
zmienna x5 =   74.85, adres = 2686768,  8 bajty
```

- Zmienna x1 typu char
- Wartość 1
- Przechowywana w pamięci pod adresem 2686791
- Zajmuje w pamięci 1 bajt

Przykład

```
zmienna x1 =      1, adres = 2686791, 1 bajty  
zmienna x2 =     10, adres = 2686788, 2 bajty  
zmienna x3 =    100, adres = 2686784, 4 bajty  
zmienna x4 =   15.15, adres = 2686780, 4 bajty  
zmienna x5 =   74.85, adres = 2686768, 8 bajty
```



- Zmienna x2 typu short int
- Wartość 10
- Przechowywana w pamięci pod adresem 2686788
- Zajmuje w pamięci 2 bajty

Przykład

```
zmienna x1 =      1, adres = 2686791,  1 bajty  
zmienna x2 =     10, adres = 2686788,  2 bajty  
zmienna x3 =    100, adres = 2686784,  4 bajty  
zmienna x4 =   15.15, adres = 2686780,  4 bajty  
zmienna x5 =   74.85, adres = 2686768,  8 bajty
```

- Zmienna x3 typu int
- Wartość 100
- Przechowywana w pamięci pod adresem 2686784
- Zajmuje w pamięci 4 bajty


Przykład

```
zmienna x1 =      1, adres = 2686791,  1 bajty  
zmienna x2 =     10, adres = 2686788,  2 bajty  
zmienna x3 =    100, adres = 2686784,  4 bajty  
zmienna x4 =  15.15, adres = 2686780,  4 bajty  
zmienna x5 =  74.85, adres = 2686768,  8 bajty
```

- Zmienna x4 typu float
- Wartość 15.15
- Przechowywana w pamięci pod adresem 2686780
- Zajmuje w pamięci 4 bajty

Przykład

```
zmienna x1 =      1, adres = 2686791,  1 bajty  
zmienna x2 =     10, adres = 2686788,  2 bajty  
zmienna x3 =    100, adres = 2686784,  4 bajty  
zmienna x4 =   15 15, adres = 2686780,  4 bajty  
zmienna x5 =  74.85, adres = 2686768,  8 bajty
```

- 
- Zmienna x5 typu double
 - Wartość 74.85
 - Przechowywana w pamięci pod adresem 2686768
 - Zajmuje w pamięci 8 bajtów

Odwołanie przez wskaźnik

- Deklaracja zmiennej

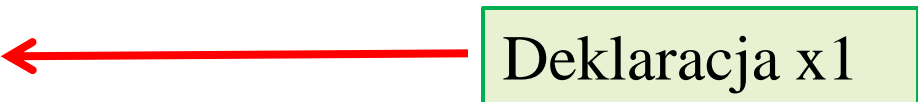
int x;

- Deklaracja wskaźnika

int *px;

Przykład (p29.c)

```
1  #include <stdio.h>
2  #include <conio.h>
3  int main()
4  { int x1;
5    int *px1;
6    x1 = 10;
7    px1 = &x1;
8    printf("\n(odwołanie przez nazwe) zmienna x1 = %4d",x1);
9    printf("\n(odwołanie przez adres) zmienna x1 = %4d",*px1);
10   getch();
11   return 0;
12 }
```



Deklaracja x1

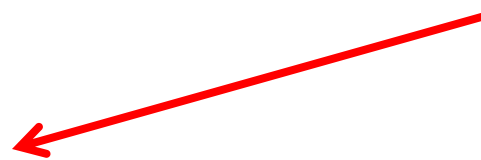
Przykład (p29.c)

```
1  #include <stdio.h>
2  #include <conio.h>
3  int main()
4  { int x1;
5    int *px1; ←
6    x1 = 10;
7    px1 = &x1;
8    printf("\n(odwołanie przez nazwe) zmienna x1 = %4d",x1);
9    printf("\n(odwołanie przez adres) zmienna x1 = %4d",*px1);
10   getch();
11   return 0;
12 }
```

Deklaracja
wskaźnika px1

Przykład (p29.c)


```
1  #include <stdio.h>
2  #include <conio.h>
3  int main()
4  { int x1;
5    int *px1;
6    x1 = 10;
7    px1 = &x1;
8    printf("\n(odwołanie przez nazwe) zmienna x1 = %4d",x1);
9    printf("\n(odwołanie przez adres) zmienna x1 = %4d",*px1);
10   getch();
11   return 0;
12 }
```



px1 wskazuje na x1

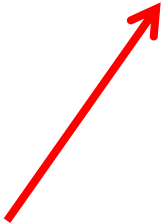
Przykład (p29.c)

```
1  #include <stdio.h>
2  #include <conio.h>
3  int main()
4  { int x1;
5    int *px1;
6    x1 = 10;
7    px1 = &x1;
8    printf("\n(odwołanie przez nazwe) zmienna x1 = %4d",x1);
9    printf("\n(odwołanie przez adres) zmienna x1 = %4d",*px1);
10   getch();
11   return 0;
12 }
```



Przykład (p29.c)

```
1  #include <stdio.h>
2  #include <conio.h>
3  int main()
4  { int x1;
5    int *px1;
6    x1 = 10;
7    px1 = &x1;
8    printf("\n(odwołanie przez nazwe) zmienna x1 = %4d",x1);
9    printf("\n(odwołanie przez adres) zmienna x1 = %4d",*px1);
10   getch();
11   return 0;
12 }
```



Przykład (p29.c)

```
1  #include <stdio.h>
2  #include <conio.h>
3  int main()
4  { int x1;
5    int *px1;
6    x1 = 10;
7    px1 = &x1;
8    printf("\n(odwołanie przez nazwe) zmienna x1 = %4d",x1);
9    printf("\n(odwołanie przez adres) zmienna x1 = %4d",*px1);
10   getch();
11   return 0;
12 }
```

```
(odwołanie przez nazwe) zmienna x1 =    10
(odwołanie przez adres) zmienna x1 =    10
```

Przypisanie wskaźnika

- px1, px2 jest wskaźnikiem do zmiennej typu int

int *px1, *px2;

- x1 jest zmienną typu int

int x1;

- Aby przypisać adres zmiennej x1 do wskaźnika px1

px1 = &x1;

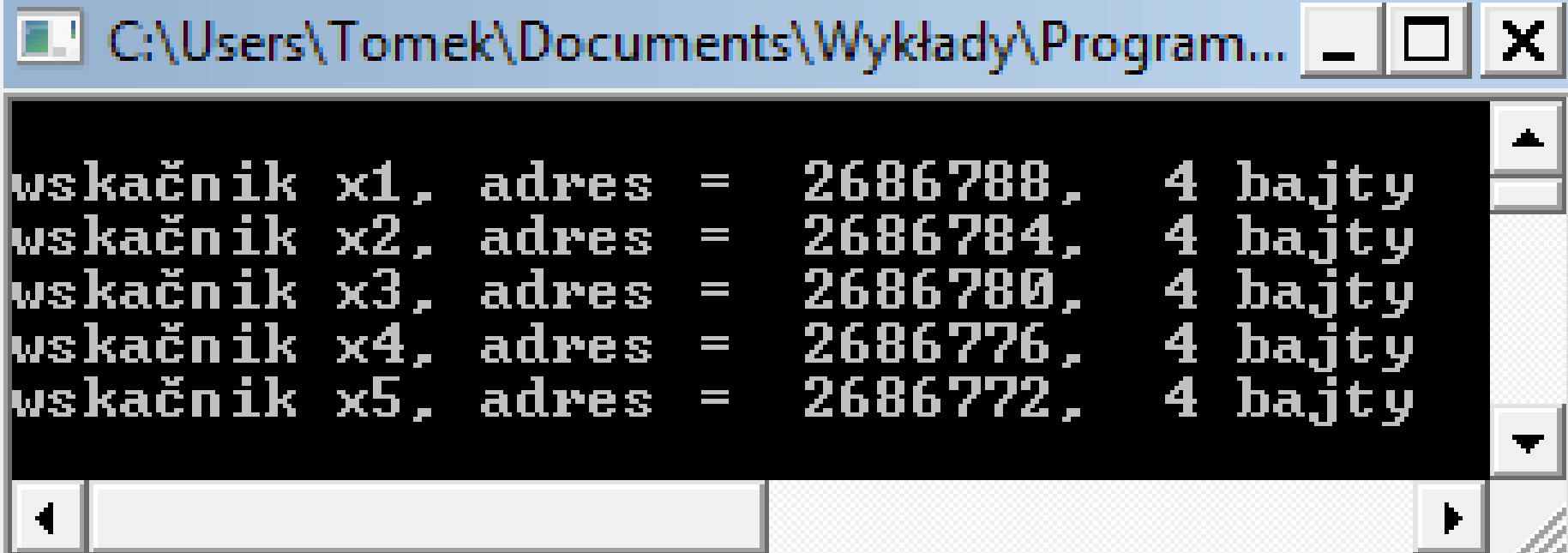
- Aby przypisać wskaźnik px2 do px1

px1=px2;

Ile miejsca zajmuje wskaźnik?

```
1  #include <stdio.h>
2  #include <conio.h>
3  □ int main(){
4      char *x1=0;
5      short int *x2=0;
6      int *x3=0;
7      float *x4=0;
8      double *x5=0;
9      printf("\nwskaźnik x1, adres = %8u, %2i bajty",&x1,sizeof(x1));
10     printf("\nwskaźnik x2, adres = %8u, %2i bajty",&x2,sizeof(x2));
11     printf("\nwskaźnik x3, adres = %8u, %2i bajty",&x3,sizeof(x3));
12     printf("\nwskaźnik x4, adres = %8u, %2i bajty",&x4, sizeof(x4));
13     printf("\nwskaźnik x5, adres = %8u, %2i bajty",&x5, sizeof(x5));
14     getch();
15     return 0;
16 }
```

Ile miejsca zajmuje wskaźnik?



A screenshot of a Windows command prompt window. The title bar shows the path "C:\Users\Tomek\Documents\Wykłady\Program...". The command prompt displays five lines of text, each showing a pointer variable, its address, and its size in bytes.

Variable	Address	Size
wskaźnik x1	2686788	4 bajty
wskaźnik x2	2686784	4 bajty
wskaźnik x3	2686780	4 bajty
wskaźnik x4	2686776	4 bajty
wskaźnik x5	2686772	4 bajty

Operacje arytmetyczne na wskaźnikach

- Jeśli p jest wskaźnikiem do konkretnego typu to dopuszczalne są operacje:

$p - 1$

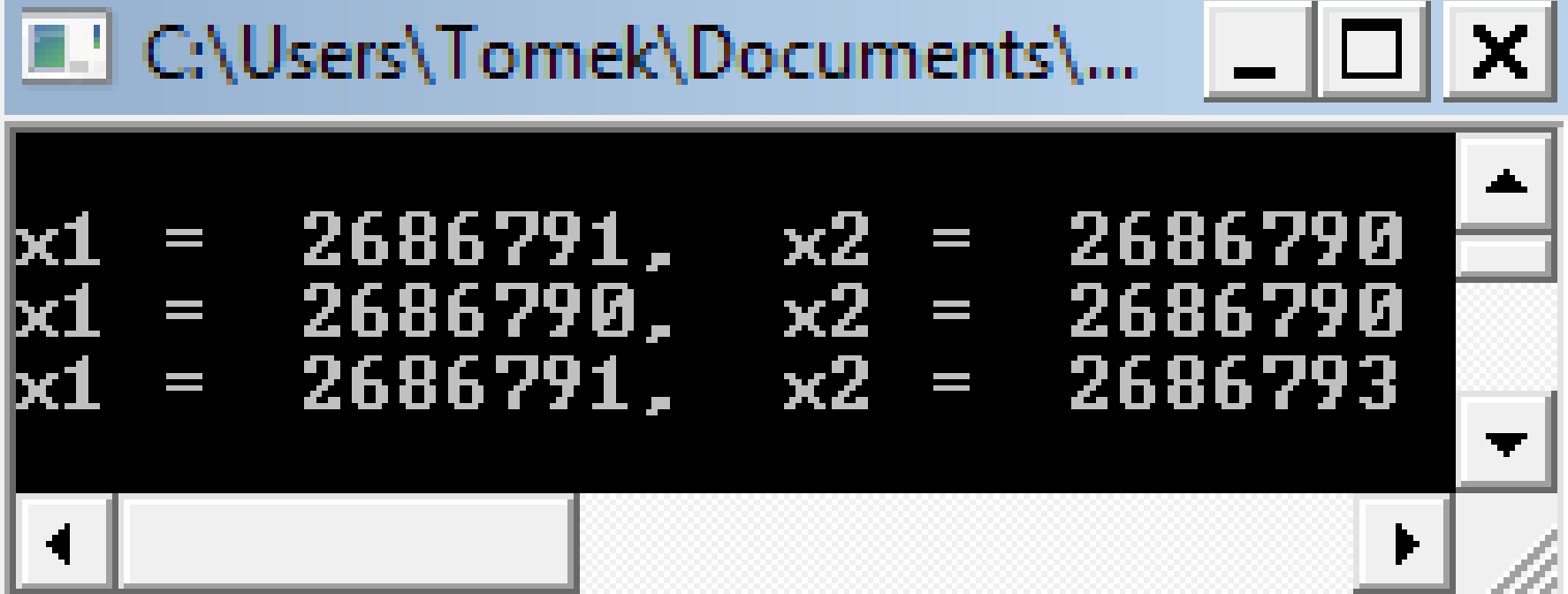
$p += i$

$++p$

Operacje na wskaźnikach - char

```
1  #include <stdio.h>
2  #include <conio.h>
3  ☐ int main(){
4      char xx1,xx2;
5      char *x1=&xx1;
6      char *x2=&xx2;
7      printf("\nx1 = %8u,  x2 = %8u",x1,x2);
8      x1=x2;
9      printf("\nx1 = %8u,  x2 = %8u",x1,x2);
10     x1++;x2+=3;
11     printf("\nx1 = %8u,  x2 = %8u",x1,x2);
12     getch();
13     return 0;
14 }
```

Operacje na wskaźnikach- char



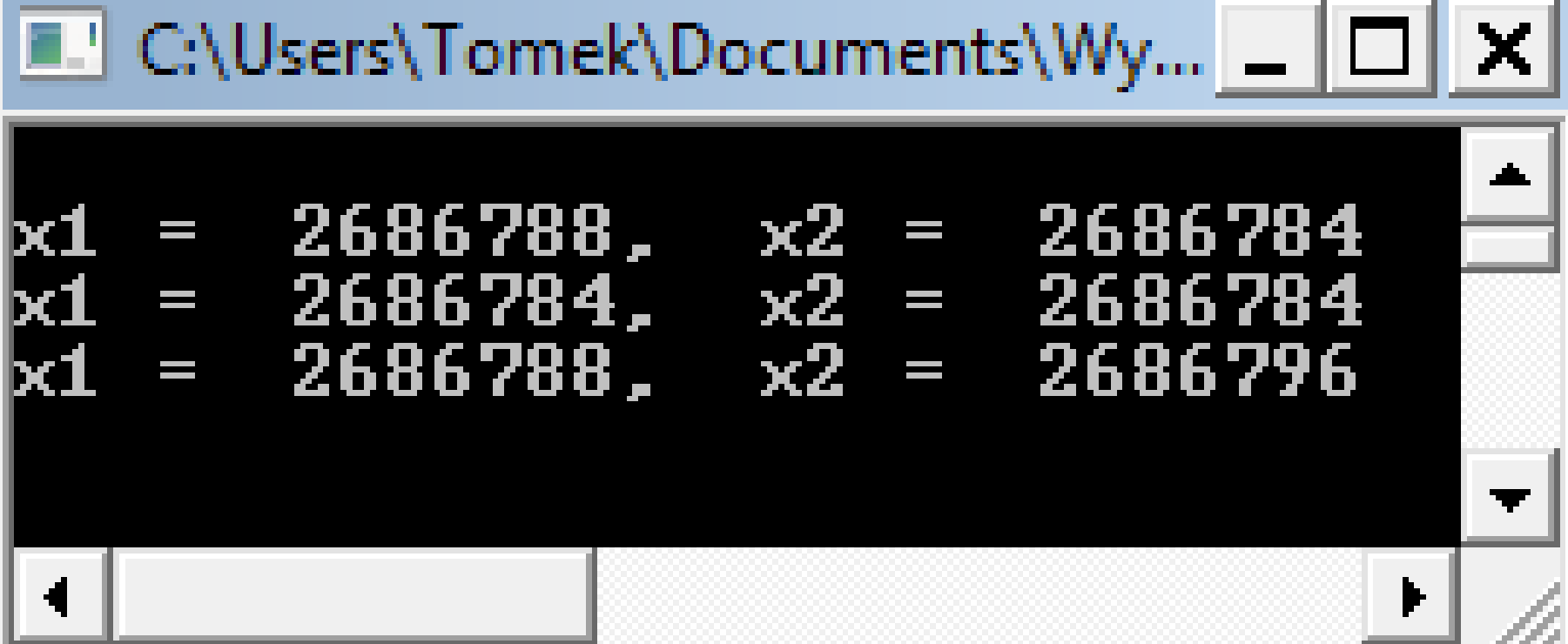
A screenshot of a Windows command prompt window. The title bar shows the path "C:\Users\Tomek\Documents\...". The command prompt displays three lines of pointer arithmetic for two character arrays, x1 and x2. The first line shows x1 at address 2686791 and x2 at 2686790. The second line shows x1 at address 2686790 and x2 at 2686790. The third line shows x1 at address 2686791 and x2 at 2686793. The output is displayed in a monospaced font on a black background.

```
x1 = 2686791,    x2 = 2686790  
x1 = 2686790,    x2 = 2686790  
x1 = 2686791,    x2 = 2686793
```

Operacje na wskaźnikach- int

```
1  #include <stdio.h>
2  #include <conio.h>
3  □ int main(){
4      int xx1,xx2;
5      int *x1=&xx1;
6      int *x2=&xx2;
7      printf("\nx1 = %8u,   x2 = %8u",x1,x2);
8      x1=x2;
9      printf("\nx1 = %8u,   x2 = %8u",x1,x2);
10     x1++;x2+=3;
11     printf("\nx1 = %8u,   x2 = %8u",x1,x2);
12     getch();
13     return 0;
14 }
```


Operacje na wskaźnikach- int



A screenshot of a Windows Notepad window. The title bar shows the file path "C:\Users\Tomek\Documents\Wy...". The text inside the window is as follows:

x1	=	2686788,	x2	=	2686784
x1	=	2686784,	x2	=	2686784
x1	=	2686788,	x2	=	2686796

The window includes standard Windows controls: minimize, maximize, and close buttons in the title bar; scroll up and scroll down buttons on the right side; and a horizontal scrollbar at the bottom.

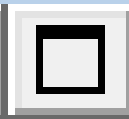
Operacje na wskaźnikach- double

```
1  #include <stdio.h>
2  #include <conio.h>
3  int main(){
4      double xx1,xx2;
5      double *x1=&xx1;
6      double *x2=&xx2;
7      printf("\nx1 = %8u,   x2 = %8u",x1,x2);
8      x1=x2;
9      printf("\nx1 = %8u,   x2 = %8u",x1,x2);
10     x1++;x2+=3;
11     printf("\nx1 = %8u,   x2 = %8u",x1,x2);
12     getch();
13     return 0;
14 }
```

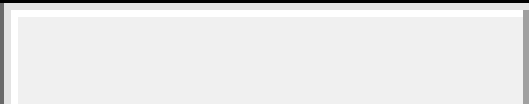
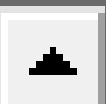
Operacje na wskaźnikach- double



C:\Users\Tomek\Documents\...



```
x1 = 2686784,    x2 = 2686776  
x1 = 2686776,    x2 = 2686776  
x1 = 2686784,    x2 = 2686800
```



Dopuszczalne zapisy

Jeśli `int *wsk;`

to możemy mieć następujące zapisy:

1. `wsk = 0;`
2. `wsk = NULL;` //równoważne z (1)
3. `wsk = &k;`
4. `wsk = (int *) 25550;` // absolutny adres w pamięci

Niedopuszczalne zapisy

Jeśli

```
int *wsk;
```

```
double a = 99.99;
```

to zapis:

```
wsk = &a;
```

jest błędny, szczególnie kiedy będziemy wykonywali operacje arytmetyczne na wskaźnikach ;

Ograniczenia

- nie wskazywać na stałą
(np. **&7** jest nielegalne)
- nie wskazywać na zwykłe wyrażenia
(np. **&(x - 100)** jest nielegalne)
- nie wskazywać na zmienną rejestrową
(np. **register x; &x** jest nielegalne)

Funkcje i wskaźniki

Przekazywanie parametrów do funkcji:

- Przekazywanie przez wartość,
- Przekazywanie przez wskaźnik.

Przekazywanie przez wskaźnik

Deklaracja funkcji ma postać:

```
void max(int,int,int*);
```

Definicja funkcji ma postać:

```
void max(int a,int b,int *v)  
{ *v = (a > b) ? a : b; }
```

Wywołanie tej funkcji ma postać:

```
int x,y,z;  
max(x,y,&z);
```


Dereferencja

```
fun(int *a, int *b)
{
    int temp;
    if (*a > *b)
    {
        temp = *a;
        *a = *b;
        *b = temp;
    }
}
```

Odwołanie przez wskaźnik

Aby stosować odwołanie przez wskaźnik należy postępować następująco:

- parametr funkcji musi być zadeklarowany, jako wskaźnik,
- wewnątrz ciała funkcji musi nastąpić dereferencja wskaźnika,
- należy przekazać aktualny adres jako parametr funkcji.

Odwołanie przez wskaźnik

```
1  #include <stdio.h>
2  #include <conio.h>
3  void suma(int *a,int *b){
4      int temp= *a + *b;
5      *b=temp;
6  }
7  int main()
8  {
9      int x1=5;
10     int x2=10;
11     suma(&x1,&x2);
12     printf("\n zmienna x1 = %4d",x1);
13     printf("\n zmienna x2 = %4d",x2);
14     suma(&x1,&x2);
15     printf("\n zmienna x2 = %4d",x2);
16     getch();
17     return 0;
18 }
```

Odwołanie przez wskaźnik

```
1  #include <stdio.h>
2  #include <conio.h>
3  void suma(int *a,int *b){
4      int temp= *a + *b;
5      *b=temp;
6  }
7  int main()
8  {
9      int x1=5;
10     int x2=10;
11     suma(&x1,&x2);
12     printf("\n zmienna x1 = %4d",x1);
13     printf("\n zmienna x2 = %4d",x2);
14     suma(&x1,&x2);
15     printf("\n zmienna x2 = %4d",x2);
16     getch();
17     return 0;
18 }
```

```
3  void suma(int *a,int *b){
4      *b += *a;
5  }
```

```
zmienna x1 =      5
zmienna x2 =     15
zmienna x2 =     20
```

Korzyści ze stosowania wskaźników

- funkcja może zwrócić do środowiska wywołującego więcej niż jedną wartość.
- za pomocą wskaźników przekazywane są tablice i łańcuchy
- za pomocą wskaźników budowane są skomplikowane struktury danych
- za pomocą wskaźników można uzyskać użyteczne informacje techniczne (np. ilość wolnej pamięci).

Tablice i wskaźniki

```
int tab[10];
```

```
tab[0]=0;
```

```
tab[1]=1;
```

```
tab[2]=2;
```

```
.
```

```
.
```

```
.
```

```
int tab[10];
```

```
int *wsk=&tab[0];
```

```
*wsk++=0;
```

```
*wsk++=1;
```

```
*wsk++=2;
```

```
.
```

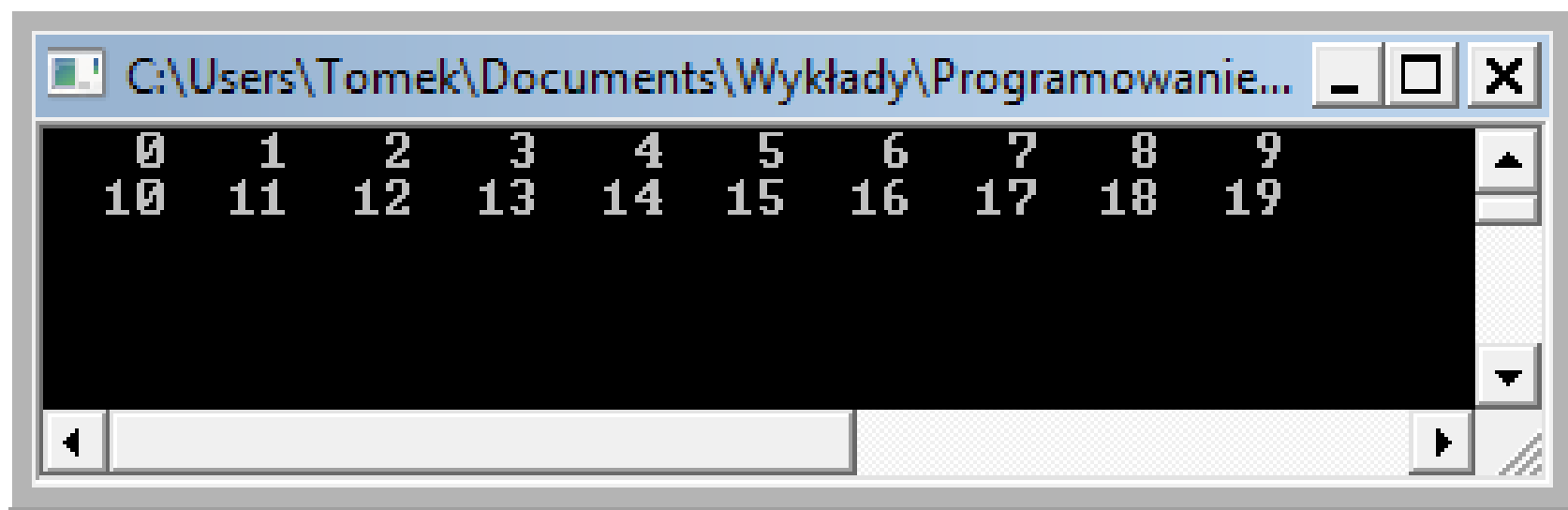
```
.
```

```
.
```

Tablice i wskaźniki

```
1  #include <stdio.h>
2  #include <conio.h>
3  ☐ int main(){
4      int i;
5      int tab[10];
6      int *wsk=&tab[0];           6 | int *wsk=tab;
7  ☐ for (i=0;i<10;i++){
8          tab[i]=i;
9          printf("%4d",tab[i]);
10     }
11     printf("\n");
12 ☐ for (i=0;i<10;i++){
13         *wsk+=i+10;
14         printf("%4d",tab[i]);
15     }
16     getch();
17     return 0;
18 }
```

Tablice i wskaźniki

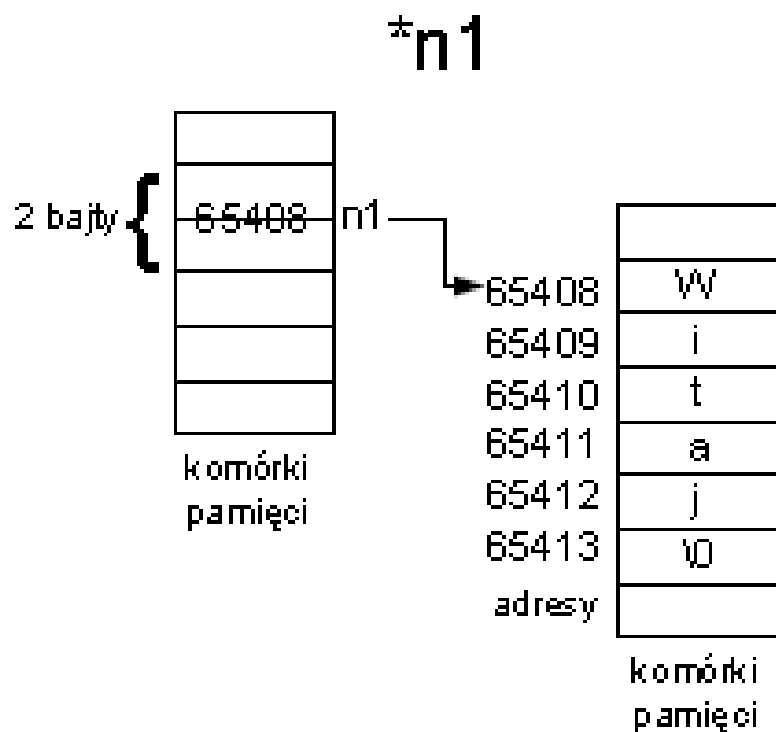


A screenshot of a Windows command prompt window. The title bar shows the path "C:\Users\Tomek\Documents\Wykłady\Programowanie...". The command prompt displays a 2x10 grid of numbers. The first row contains numbers 0 through 9, and the second row contains numbers 10 through 19. The window has standard Windows controls (minimize, maximize, close) and a scrollbar on the right side.

0	1	2	3	4	5	6	7	8	9
10	11	12	13	14	15	16	17	18	19

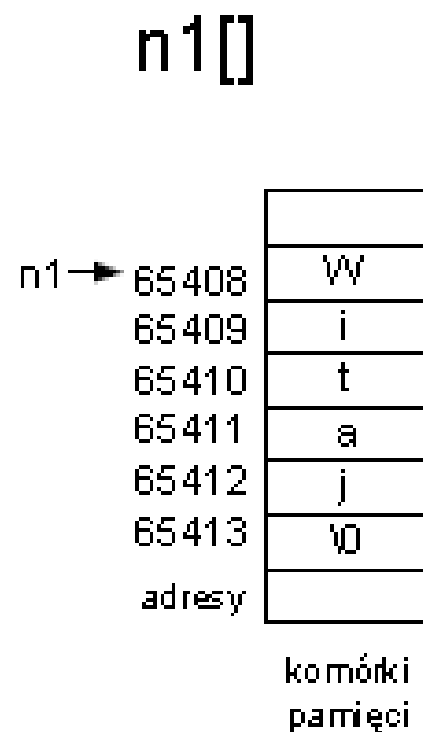
Łańcuchy i wskaźniki

`char *n1 = "Witaj";`



`char *n1 = "Witaj";`
n1 jest zmienną wskaźnikową

`char n1[] = "Witaj";`



`char n1[] = "Witaj";`
n1 jest stałą wskaźnikową

Podsumowanie

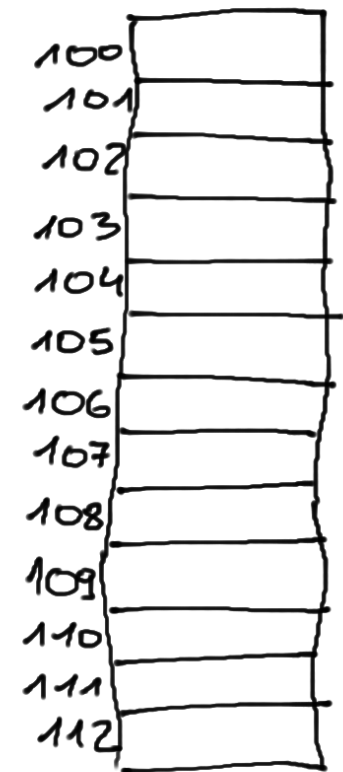
PAMIĘĆ

100	
101	
102	
103	
104	
105	
106	
107	
108	
109	
110	
111	
112	

Podsumowanie

int a;

PAMIĘĆ



Podsumowanie

int a;

PAMIĘĆ

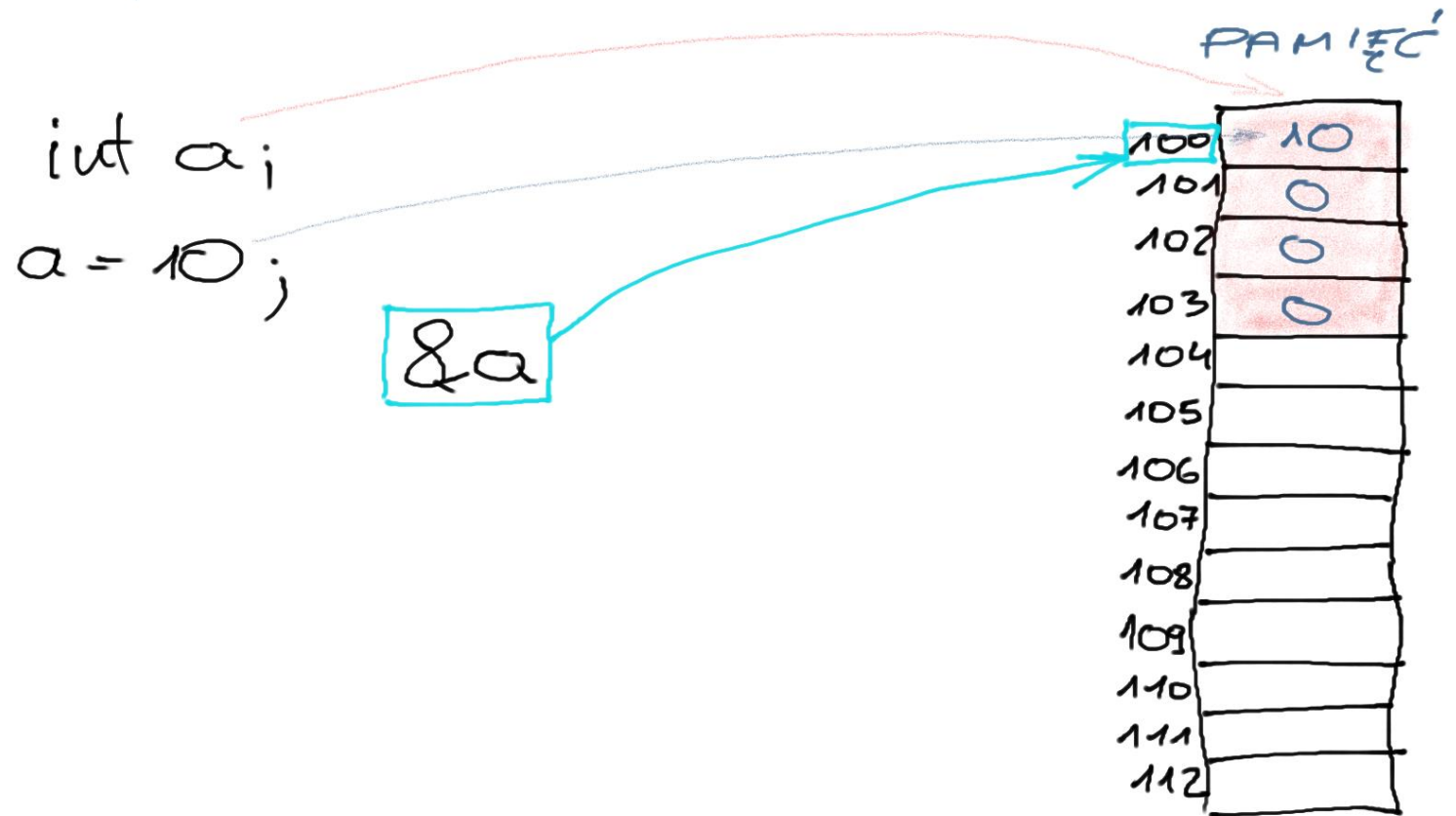


Podsumowanie

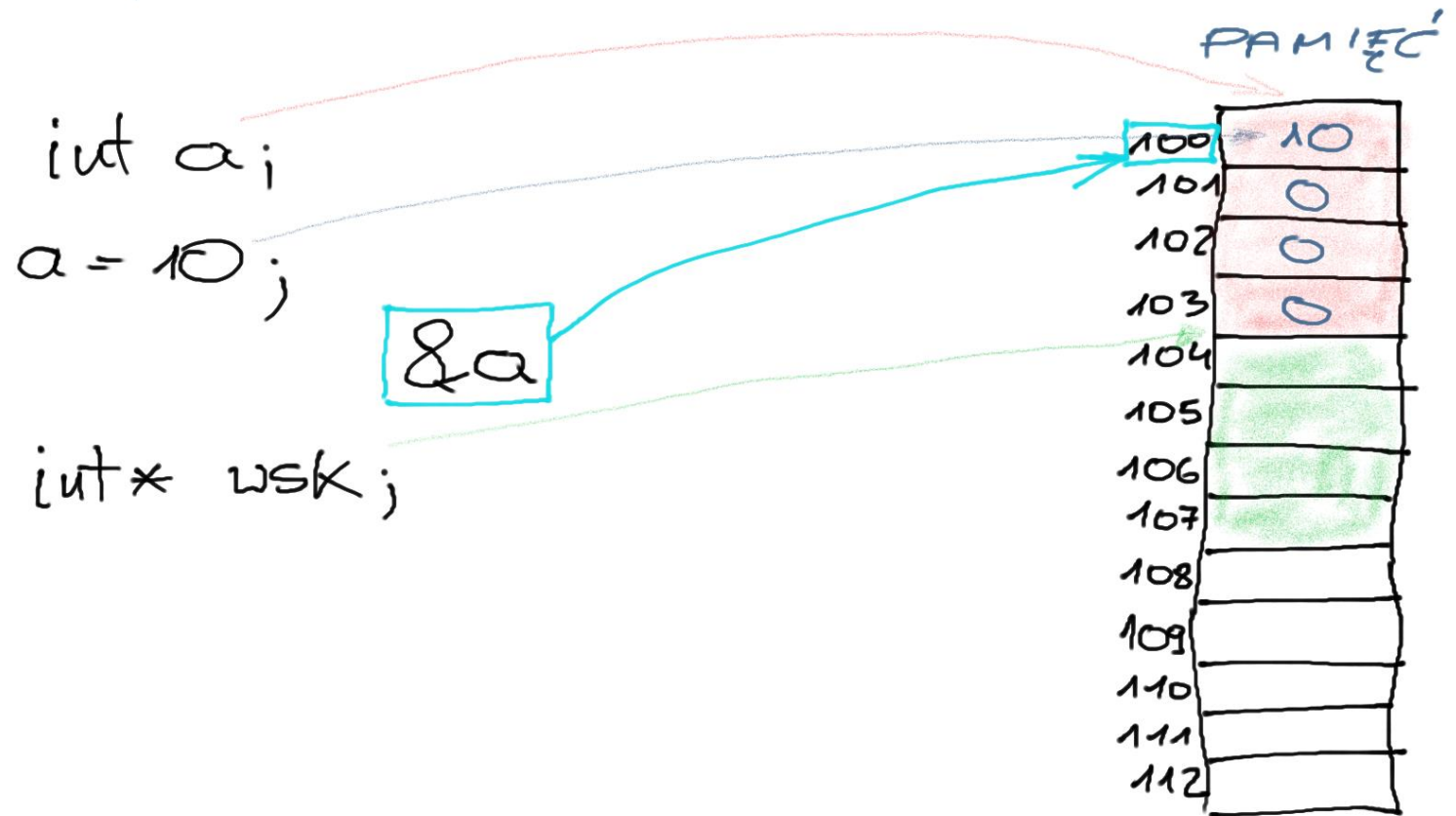
```
int a;  
a = 10;
```



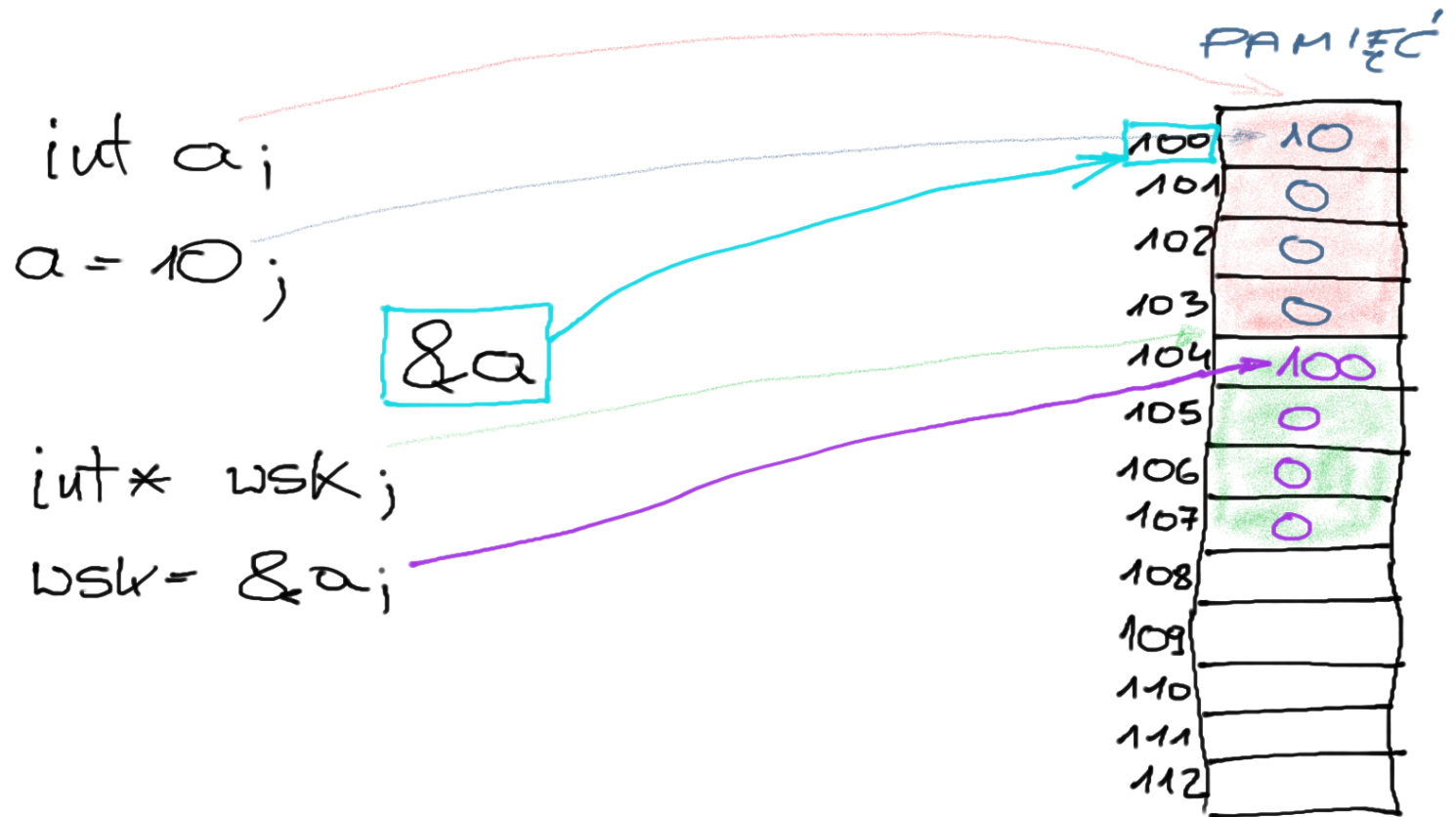
Podsumowanie



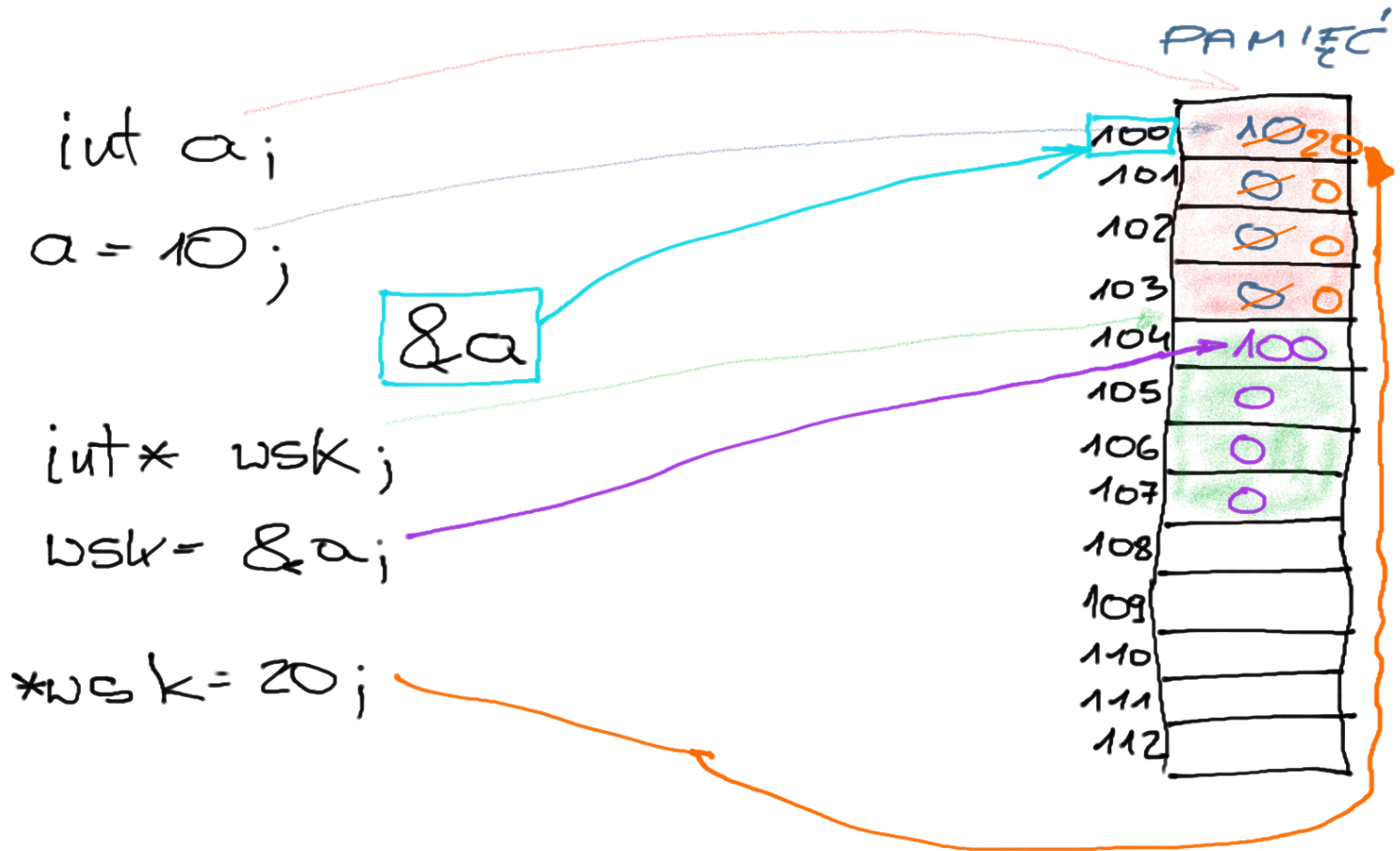
Podsumowanie



Podsumowanie



Podsumowanie

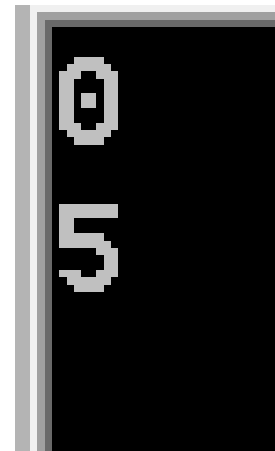


Ćwiczenie I

```
1  #include <stdio.h>
2  #include <conio.h>
3  □ int main(){
4    int i;
5    unsigned char tab[10]={0,1,2,3,4,5,6,7,8,9};
6    unsigned char *wsk=tab;
7    printf("%d\n",*wsk);
8    wsk+=5;
9    printf("%d\n",*wsk);
10   getch();
11   return 0;
12 }
```

Ćwiczenie I

```
1  #include <stdio.h>
2  #include <conio.h>
3  int main(){
4      int i;
5      unsigned char tab[10]={0,1,2,3,4,5,6,7,8,9};
6      unsigned char *wsk=tab;
7      printf("%d\n",*wsk);
8      wsk+=5;
9      printf("%d\n",*wsk);
10     getch();
11     return 0;
12 }
```



Ćwiczenie 2

```
1  #include <stdio.h>
2  #include <conio.h>
3  □ int main(){
4    int i;
5    unsigned char tab[20]={0,1,2,3,4,0,1,2,3,4,0,1,2,3,4,0,1,2,3,4};
6    unsigned short *wsk=tab;
7    printf("%d\n",*wsk);
8    wsk+=2;
9    printf("%d\n",*wsk);
10   getch();
11   return 0;
12 }
```

Ćwiczenie 2

```
1  #include <stdio.h>
2  #include <conio.h>
3  int main(){
4      int i;
5      unsigned char tab[20]={0,1,2,3,4,0,1,2,3,4,0,1,2,3,4,0,1,2,3,4};
6      unsigned short *wsk=tab;
7      printf("%d\n",*wsk);
8      wsk+=2;
9      printf("%d\n",*wsk);
10     getch();
11     return 0;
12 }
```

256

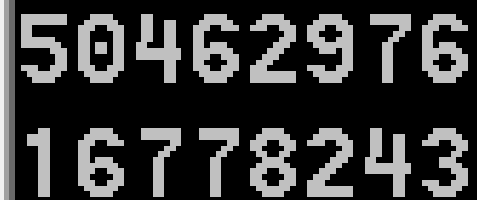
4

Ćwiczenie 3

```
1  #include <stdio.h>
2  #include <conio.h>
3  int main(){
4      int i;
5      unsigned char tab[20]={0,1,2,3,4,0,1,2,3,4,0,1,2,3,4,0,1,2,3,4};
6      unsigned int *wsk=tab;
7      printf("%d\n",*wsk);
8      wsk+=2;
9      printf("%d\n",*wsk);
10     getch();
11     return 0;
12 }
```

Ćwiczenie 3

```
1  #include <stdio.h>
2  #include <conio.h>
3  int main(){
4      int i;
5      unsigned char tab[20]={0,1,2,3,4,0,1,2,3,4,0,1,2,3,4,0,1,2,3,4};
6      unsigned int *wsk=tab;
7      printf("%d\n",*wsk);
8      wsk+=2;
9      printf("%d\n",*wsk);
10     getch();
11     return 0;
12 }
```



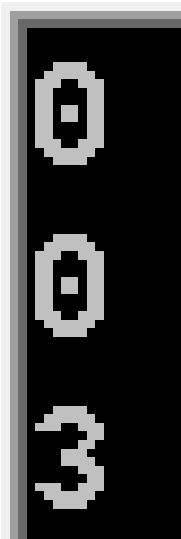
50462976
16778243

Ćwiczenie 4

```
1  #include <stdio.h>
2  #include <conio.h>
3  int main(){
4  int i;
5  unsigned int tab[20]={0,1,2,3,4,0,1,2,3,4,0,1,2,3,4,0,1,2,3,4};
6  unsigned char *wsk=tab;
7  printf("%d\n",*wsk);
8  wsk+=9;
9  printf("%d\n",*wsk);
10 wsk+=3;
11 printf("%d\n",*wsk);
12
13 // ponizej nie analizujemy
14 unsigned char *wsk1=tab;
15 for (i=0;i<20;i++){
16     printf("%02d ",*wsk1++);
17     if ((i+1)%4==0) printf("; ");
18 }
19 getch();
20 return 0;
21 }
```


Ćwiczenie 4

```
1  #include <stdio.h>
2  #include <conio.h>
3  int main(){
4  int i;
5  unsigned int tab[20]={0,1,2,3,4,0,1,2,3,4,0,1,2,3,4,0,1,2,3,4};
6  unsigned char *wsk=tab;
7  printf("%d\n",*wsk);
8  wsk+=9;
9  printf("%d\n",*wsk);
10 wsk+=3;
11 printf("%d\n",*wsk);
12
13 // ponizej nie analizujemy
14 unsigned char *wsk1=tab;
15 for (i=0;i<20;i++){
16     printf("%02d ",*wsk1++);
17     if ((i+1)%4==0) printf("; ");
18 }
19 getch();
20 return 0;
21 }
```



Ćwiczenie 4

```
1  #include <stdio.h>
2  #include <conio.h>
3  int main(){
4  int i;
5  unsigned int tab[20]={0,1,2,3,4,0,1,2,3,4,0,1,2,3,4,0,1,2,3,4};
6  unsigned char *wsk=tab;
7  printf("%d\n",*wsk);
8  wsk+=9;
9  printf("%d\n",*wsk);
10 wsk+=3;
11 printf("%d\n",*wsk);
12
13 // poniżej nie analizujemy
14 unsigned char *wsk1=tab;
```

```
0
0
3
00 00 00 00 ; 01 00 00 00 ; 02 00 00 00 ; 03 00 00 00 ; 04 00 00 00 ;
```

```
19 getch();
20 return 0;
21 }
```



Koniec