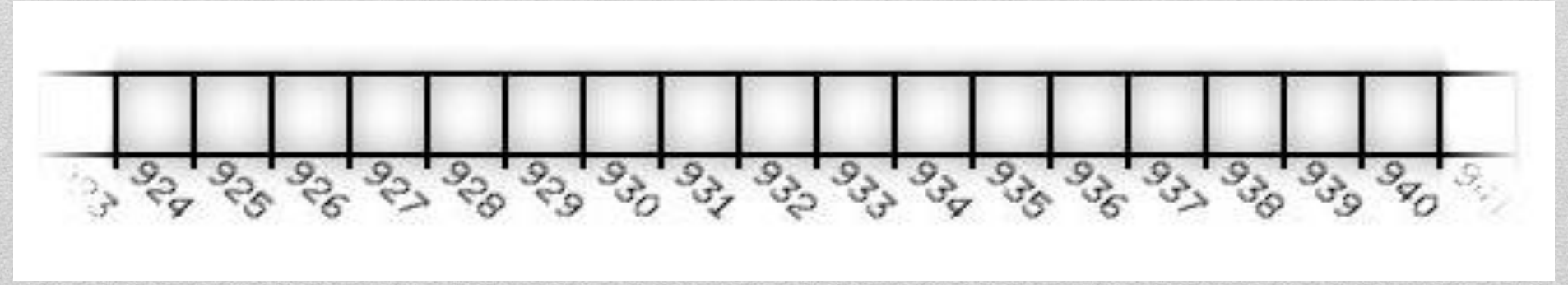
ЖИТОМИРСЬКА ПОЛІТЕХНІКА
ДЕРЖАВНИЙ УНІВЕРСИТЕТ

Лекція 16.
Покажчики у мові Сі

Згадаємо теорію…

Змінна – це область пам’яті, яка має ім’я і в якій зберігається значення певного типу даних

Будь-яке значення змінної зберігається у пам’яті.



Пам’ять під **локальні** змінні виділяється при запуску функції і звільняється при завершенні функції.

Пам’ять під **глобальні** змінні виділяється при запуску функції **main** і звільняється при завершенні функції **main**.

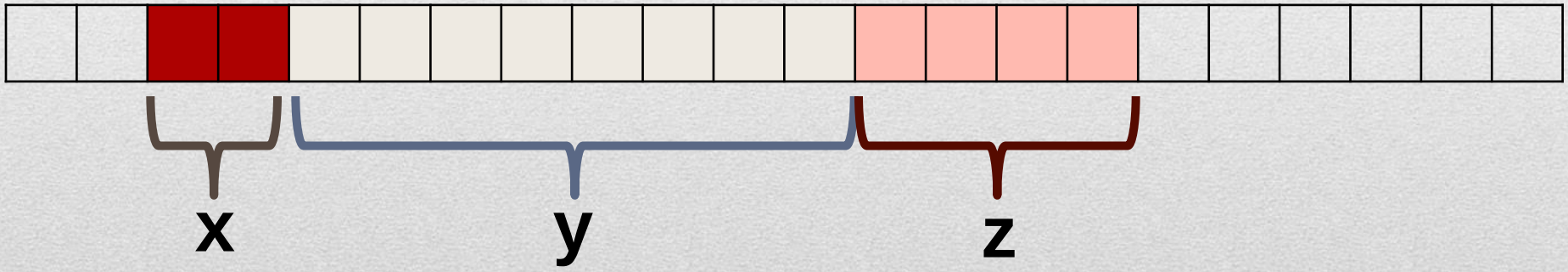
Виділення пам’яті під змінну – це закріплення за змінною конкретних комірок пам’яті.

Для того, щоб створити змінну, її потрібно оголосити, вказавши тип даних:

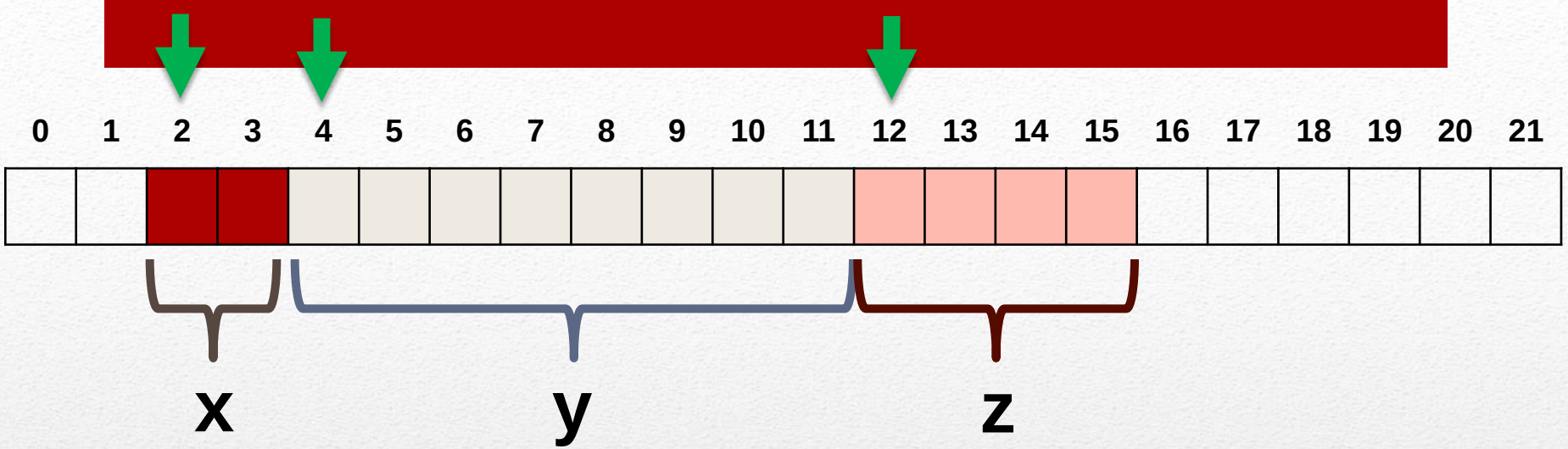
тип даних	ім’я змінної	;

Приклад:

```
short x;
double y;
int z;
```



Тип даних визначає скільки байтів пам’яті буде виділено під змінну.



Кожна комірка у пам’яті має унікальний номер.

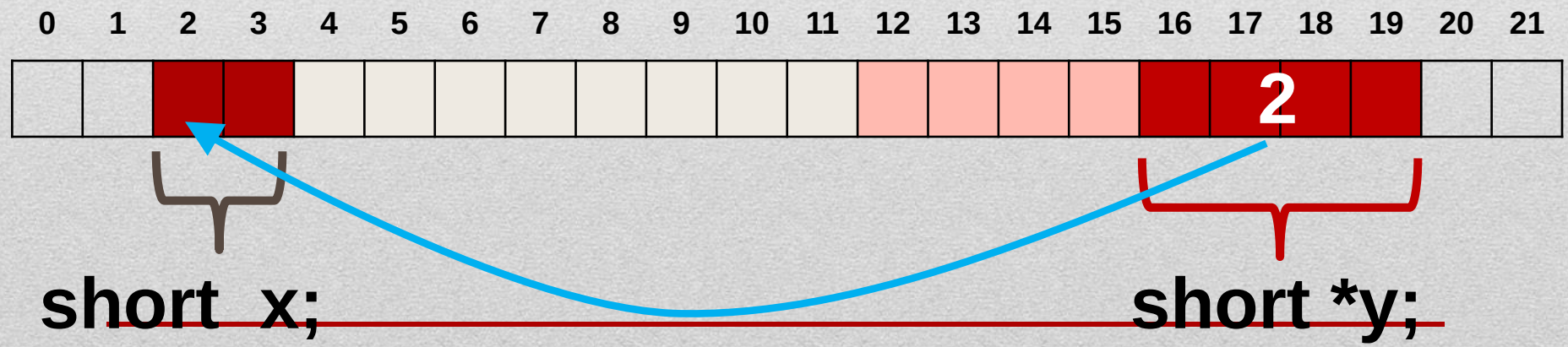
Номер комірки пам’яті, де зберігається значення змінної називають адресою змінної.

Якщо змінна займає кілька байтів, то адреса вказує на першу (початкову) комірку.

У мові Сі є можливість оголосити змінну, яка міститиме номер комірки пам’яті, де зберігатиметься значення певного типу.

Така змінна називається **покажчиком**.

Покажчик – це змінна, значенням якої є адреса пам’яті, де зберігається значення певного типу.



Оскільки покажчики в С зберігають адреси пам'яті, їх розмір не залежить від типу даних, на які вони вказують. Цей розмір покажчиків у С залежить лише від архітектури системи.

Змінна, оголошена як **покажчик**, **займає** 4 **байта** в оперативній пам'яті (у разі 32-бітної версії компілятора), 2 **байта** (16-бітної) , 8 байт у (64-бітної)

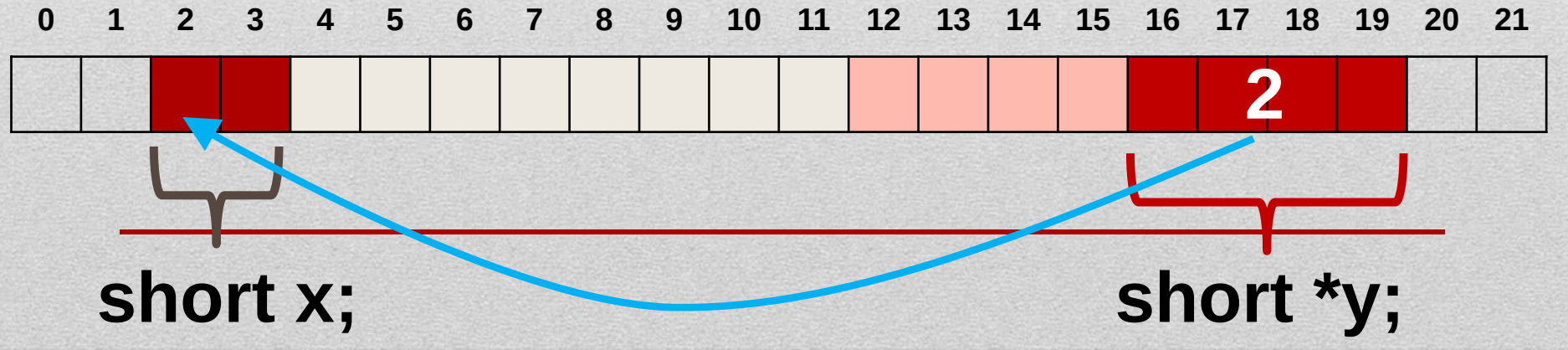
Щоб оголосити змінну типу покажчика на деякий тип даних, потрібно перед іменем змінної поставити зірочку:

тип даних * ім'я змінної ;

Приклад:

```
short *x;
double *y;
int *z;
```

Змінна-покажчик у пам'яті займає **4 байти** *(у разі 32-бітної версії компілятора)*, незалежно від того, на значення якого типу вона вказує.



В одному оголошенні можна вказувати звичайні змінні та змінні-покажчики:

```
short x, *px, y, *py;
double d, a, b = 0, *pd, *bp;
```

При запуску функції операційна система визначає вільні ділянки пам’яті і розміщує в них оголошені у функції змінні.

Тому при кожному виклику функції одна й та сама змінна може розміщуватися у різних комірках пам’яті.

Причина: при виході з функції усі локальні змінні видаляються, пам’ять очищується. При наступному виклику функції під локальні змінні знову виділяється пам’ять.

Для визначення адреси змінної у пам’яті застосовується операція «взяття адреси» **&**:

& | ім’я змінної

```
int x = 10;
int *px = &x;
```

```
double y = 10;
double *py = &x;
```

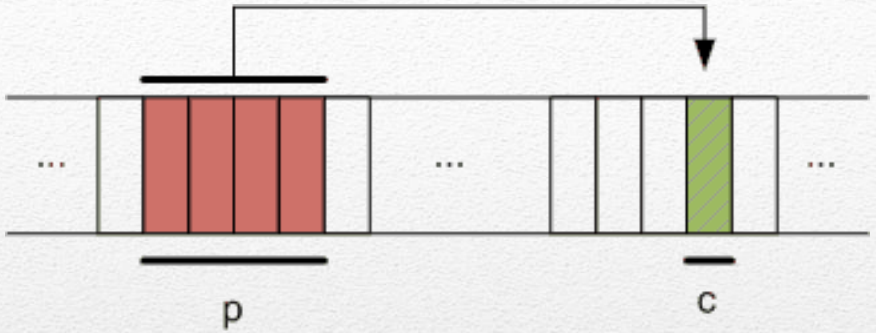
Помилка невідповідності типів. Неможливо записати у **py** адресу змінної типу **int**

Операція взяття адреси «&» застосовується разом зі змінною і повертає адресу цієї змінної.

Операція розіменування «*» використовується разом з покажчиками і вилучає значення, на яке вказує змінна-покажчик, розташована безпосередньо після символа «*».

ПРИКЛАД №1

```
char c, *p;
p = &c;
```



	Змінна	Покажчик
Адреса	&c	p
Значення	c	*p

```c
#include <stdio.h>
#include <windows.h>
int main(){
    char c='v', *pc;
    pc = &c;
    printf("\n &c = %x    = %c ", &c, c);
    printf("\n &pc= %x    = %x ", &pc, pc);
    printf("\n c = %c, *pc= %c ", c, *pc);
    printf("\n ------------------\n ");
    int a, *p;
    a = 100;
    p = &a;
    printf("\n a  = %d = %x ", a, a);// Значення змінної a
    printf("\n &a = %x", &a);// Адреса змінної
    printf("\n *p = %d = %x ", *p, *p);//  Дані за адресою
    printf("\n p  = %x ", p);// Значення покажчика
    printf("\n &p = %x\n", &p);//Адреса розташування покажчика
system("pause");
return 0;
}
```

```
&c  = 6ffce3   = v
&pc = 6ffcd4   = 6ffce3
c = v, *pc= v
-----------------

a  = 100 = 64
&a = 6ffcc8
*p = 100 = 64
p  = 6ffcc8
&p = 6ffcbc
```

ПРИКЛАД №2

```c
#include <stdio.h>
#include <windows.h>
int main(){

    int A = 100;
    int *p;
    p = &A;

    printf("%p\n", p);
    printf("%d\n", p);

    printf("%d\n", sizeof(p));
    printf("%d\n", *p);

    *p = 200;
    printf(" %d\n", A);
    printf(" %d\n", *p);

system("pause");
return 0;
}
```

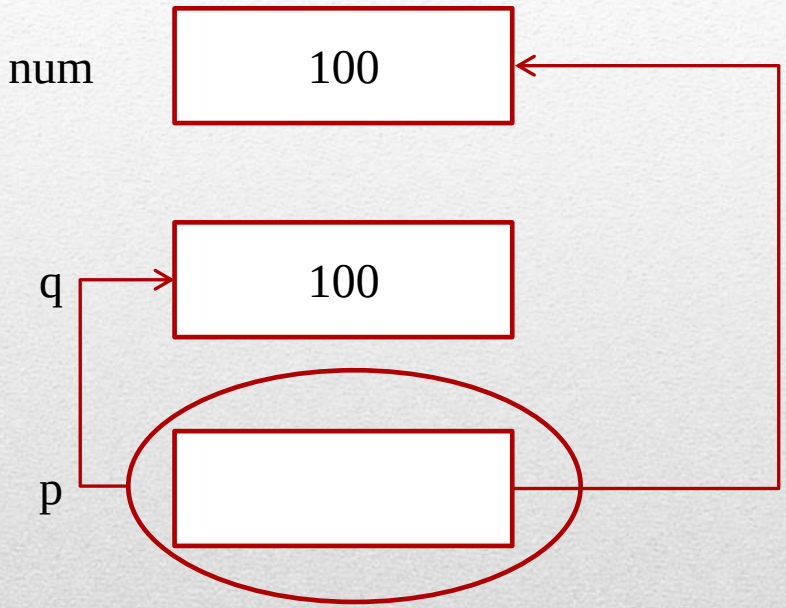
```
 0093FD1C
9698588
4
100
200
200
```

ПРИКЛАД №2

```
int main() {

    int num, q;
    int *p;
    num = 100;
    p = &num;
    q = *p;
    printf("%d\n", q);

return 0;
}
```




```
q=100
```

```c
int A = 100;
int *a = &A;
double B = 2.3;
double *b = &B;

printf(" %d\n", sizeof(A));
printf(" %d\n", sizeof(a));
printf(" %d\n", sizeof(B));
printf(" %d\n", sizeof(b));
```

```
4
4
8
4
```

```c
int a, *b;
a = 134;
b = &a;

printf("\n a = %d  a=%x ", a, a);
printf("\n a   %x ", &a);
printf("\n b = %d  b= %x ", *b,
*b);
printf("\n b = %x", b);
printf("\n b   %x \n", &b);
```

```
a = 134  a=86
a   9ef9e4
b = 134  b= 86
b = 9ef9e4
b   9ef9d8
```

```c
int A[10] = {1,2,3,4,5,6,7,8,9,10};
int *a, *b;

a = &A[0];
b = &A[9];

printf("&A[0] == %p\n", a);
printf("&A[9] == %p\n", b);
```

```
&A[0] == 010FFB34
&A[9] == 010FFB58
```

```c
int *p;
int arr[8] = {5,7,-3,0,15,21,4,2};

printf("\n");
p = &arr[0];
printf(" %d\n", *p);
p++;
printf(" %d\n", *p);
p = p + 3;
printf(" %d\n", *p);
```

```
5
7
15
```

Описати процедуру ShiftLeft3 (A, B, C), що виконує лівий циклічний зсув: значення A переходить в C, значення C - в B, значення B - в A (A, B, C – дійсного типу і є одночасно вхідними та вихідними).

За допомогою цієї процедури виконати лівий циклічний зсув для двох даних наборів з трьох чисел: (A1, B1, C1) і (A2, B2, C2).

```c
void shiftleft3(float *x, float *y, float *z) {
    float tmp;
    tmp = *x;
    *x = *y;
    *y = *z;
    *z = tmp;
}

int main(void){
    float a, b, c;
    for (int i = 1; i <= 2; ++i) {
    printf("A:"); scanf_s("%f", &a);
    printf("B:"); scanf_s("%f", &b);
    printf("C:"); scanf_s("%f", &c);

    shiftleft3(&a, &b, &c);
    printf("A:%f; B:%f; C:%f\n", a, b, c);
    }
return 0;
}
```

```
 A:4.5
 B:8
 C:6.25
A:8; B:6.25; C:4.5
 A:4
 B:5
 C:8.66
A:5; B:8.66; C:4
```

Масив - безпосередньо вказує на перший елемент, покажчик - змінна, яка зберігає адресу першого елемента.

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

int main() {

    int A[5] = { 1, 2, 3, 4, 5 };
    int *p = A;
    for(int i=0; i<5; i++)
        printf("%d\t", *(A + i));
    printf("\n");
    for (int i = 0; i < 5; i++)
        printf("%d\t", *(p + i));
    printf("\n");
    for (int i = 0; i < 5; i++)
        printf("%d\t", A[i]);
system("pause");
return 0;
}
```

Особливості функції scanf

1) після рядку формату вказуються адреси змінних, в які записуються прочитані з клавіатури значення:

```
scanf("%d %d %lf %ld", &a, &b, &c, &d);
```

2) читання кількох значень:

```
int a, b;
double c;
long d;
scanf("%d %d %lf %ld", &a, &b, &c, &d);
```

```
1 2 3 4
```

```
1
2
3
4
```

3) у рядку формату між специфікаторами форматування можна вказувати роздільники:

```
int a, b;
double c;
long d;
scanf("%d, %d, %lf, %ld",
                        &a, &b, &c, &d);
```

```
1, 2, 3, 4
```

4) якщо значення, що вводилося не відповідало формату, то у змінну нічого не записується:

```
int x = 10;
scanf("%d", &x);
printf("x = %d\n", x);
```

```
asdasdasd
x = 10
```

Якщо значення змінної некоректно введене, то усі інші змінні прочитані не будуть:

```
int a, b, c, d;
int count = scanf("%d %d %d %d", &a,
                                 &b, &c, &d);
printf("Coorect values: %d\n", count);
```

```
1 asd 3 4
Coorect values: 1
```

5) функція scanf повертає кількість коректно прочитаних змінних:

```
int a, b, c, d;
int count = scanf("%d %d %d %d", &a,
                                 &b, &c, &d);
printf("Coorect values: %d\n", count);
```

```
1 2 3 asdasd
Coorect values: 3
```