Указатели

Pointers

Деклариране

- Указател е променлива, в която се съхранява адрес в паметта
- Използва се за индиректен (косвен) достъп до стойността, записана на съответния адрес
- тип_данни * име_променлива;
  - int *p1;
  - double *p3;
  - char *p4;
- Променлива указател винаги заема 4B памет, независимо от типа – типа указва какъв тип данни се очаква да са записани на адреса, към който сочи указателя
- Частен случай – void* - не задава изрично типа на стойността, може да е всеки
  - Когато се използва указателя може да се наложи преобразуване (cast) към конкретен тип

Инициализация

- Ако бъде използван неинициализиран указател или указател, инициализиран с NULL, програмата ще даде грешка по време на изпълнението
- Инициализацията означава в указателя да се запише определен адрес
  - Инициализация с NULL – означава, че указателят не сочи никъде

    int* pa = NULL;
  - Да се насочи към адреса на друга променлива

    int a, *pa;

    pa = &a;
  - Динамично заделяне на памет (#include <stdlib.h>)
    - void *malloc(size_t size);

    int* p = (int*)malloc(sizeof(int)); //заделя памет за една int стойност

    int* p = (int*)malloc(10*sizeof(int)); //заделя памет за 10 int стойности – масив
    - Внимание: динамично заделена памет трябва да се освободи изрично, иначе се получава memory leak (изтичане на памет, т.е. неосвободена памет)
      - void free(void *ptr);

      free(p);

Работа с указатели

- За извличане на стойност от адрес се използва операция *
- *p – връща стойността, записана на адреса, към който сочи указателят p
- След като указател се насочи към адреса на дадена променлива, стойността може да се достъпва през променливата (директно) или през указателя (индиректно)

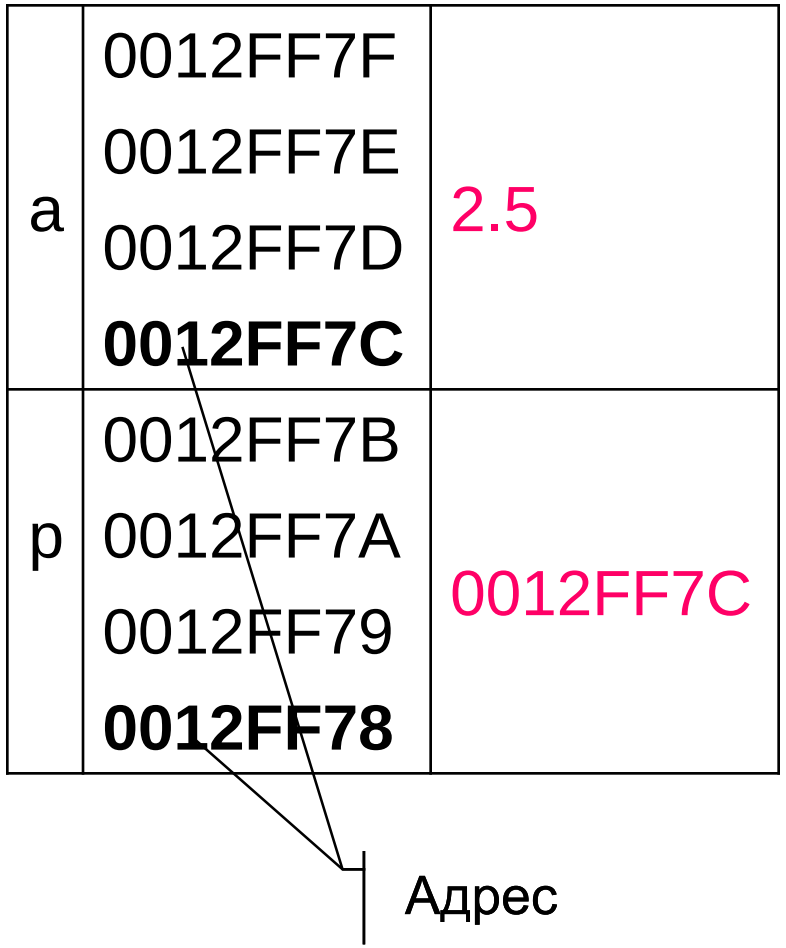
int a		**int *pa**	
a	стойност	*pa	стойност
&a	адрес	pa	адрес

Пример

```
#include <stdio.h>
int main ()
{
   float a, *p;
   printf("Adresi: %p %p\n", &a, &p);
   a = 2.5;
   p = &a;
   printf("a=%.1f p=%p *p=%.1f\n", a, p, *p);
   *p += 4.7f;
 printf("a=%.1f\n", a);
   return 0;
}
```

Резултати:

Adresi: 0012FF7C 0012FF78
a=2.5 p=0012FF7C *p=2.5
a=7.2



Указатели & Масиви

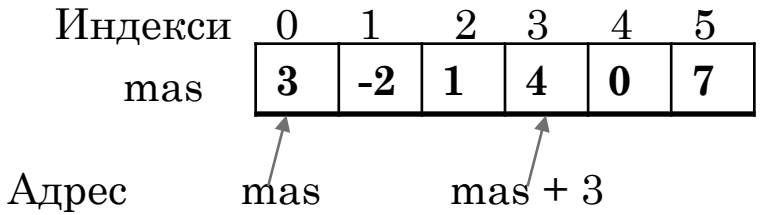
Pointers & Arrays

Масивите са указатели

- При декларирането на масив
  - float mas[10];
  - Името на масива всъщност е указател
  - Указателят сочи към първия елемент от масива (към неговия адрес)
    - mas ≡ &mas[0] und *mas ≡ mas[0]
  - mas е базов адрес и елементите от масива могат да бъдат достъпени индиректно през този базов адрес и отместване (offset) – аритметика с указатели
    - mas – адресът на елемент с индекс 0, mas + 3 ≡ &mas[3], mas + i ≡ &mas[i]
    - Стойността се чете с операция * – *(mas+3) – индиректно адресиране
      - Внимание: *mas + 3 ≠ *(mas+3)

```
i = 2;
*(mas+i) //връща стойност 1
*mas + i //връща стойност 5
i+=2;
*(mas+i) = -1;
```

3	-2	1	4	-1	7



Предаване на параметри на функции по адрес

Предаване на параметър по стойност vs Предаване на параметър по адрес

```c
#include <stdio.h>
void swap(int, int);

int main()
{
   int a, b;

   printf("Stoinosti:");
   scanf("%d%d", &a, &b);
   printf("a = %d, b = %d\n", a, b);
   swap(a, b);
   printf("a = %d, b = %d\n", a, b);
}
void swap(int v1, int v2)
{
   printf("swap, predi: v1 = %d, v2 = %d\n", *v1, *v2);
   int temp = v1;
   v1 = v2;
   v2 = temp;
   printf("swap, sled: v1 = %d, v2 = %d\n", *v1, *v2);
}
```

Този вариант не работи, защото се използва предаване на параметри по стойност – стойностите ще се разменят само в рамките на функцията swap. В main функцията стойностите на a и b остават непроменени

```c
#include <stdio.h>
void swap(int*, int*);

int main()
{
   int a, b;

   printf("Stoinosti:");
   scanf("%d%d", &a, &b);
   printf("a = %d, b = %d\n", a, b);
   swap(&a, &b);
   printf("a = %d, b = %d\n", a, b);
}
void swap(int* v1, int* v2)
{
   printf("swap, predi: v1 = %d, v2 = %d\n", *v1, *v2);
   int temp = *v1;
   *v1 = *v2;
   *v2 = temp;
   printf("swap, sled: v1 = %d, v2 = %d\n", *v1, *v2);
}
```

Този вариант работи, защото се използва предаване на параметри по адрес – записвайки стойности в адресите, към които сочат указателите v1 и v2, те реално се записват в променливите a и b

Размяна на две стойности в масив с индиректно адресиране

```
#include <stdio.h>

void swap(int*, int*);

int main()
{
	int a[2];

	printf("Stoinosti za a i b:");
	scanf("%d%d", a, a + 1);
	printf("a = %d, b = %d\n", *a, *(a+1));
	swap(a, a + 1);
	printf("a = %d, b = %d\n", a[0], a[1]);
}

void swap(int* a, int* b)
{
	int temp = *a;
	*a = *b;
	*b = temp;
}
```

Размяна на последователни елементи по начален адрес

```
#include <stdio.h>

void swap(int*);

int main()
{
	int a[2];

	printf("Stoinosti za a i b:");
	scanf("%d%d", a, a + 1);
	printf("a[0] = %d, a[1] = %d\n", *a, *(a+1));
	swap(a);
	printf("a[0] = %d, a[1] = %d\n", a[0], a[1]);
}

void swap(int* nachalo)
{
	int temp = *nachalo;
	*nachalo = *(nachalo+1);
	*(nachalo+1) = temp;
}
```

- Когато функцията swap е предназначена за размяна на два **последователни** елемента от масив, може да ѝ се подаде само адреса на първия елемент, защото е известно, че следващият елемент се намира на адреса на първия елемент + отместване 1 елемент – nachalo + 1

- Указател може да се третира като масив и паметта след неговия адрес да се достъпва чрез индексиране.

- Еквивалентно функцията swap може да бъде записана и като

```
void swap(int* nachalo)
{
	int temp = nachalo[0];
	nachalo[0] = nachalo[1];
	nachalo[1] = temp;
}
```

Намиране на максимална и минимална стойност в масив

```
#include <stdio.h>
void min_max(float mas[], int n, float*min,
float*max);

void read(float mas[], int n);
void print(float mas[], int n);
void read_int(const char* prompt, int* val);
void read_float(const char* prompt, float* val);
```

```
int main()
{
    float mas[100], min, max;
    int br;

    read_int("Vavedete broi elementi:", &br);
    read(mas, br);

    min_max(mas, br, &min, &max);

    printf("min = %.2f, max = %.2f\n", min, max);
    print(mas, br);
}
```

Имплементации на функциите

```
void read_int(const char* prompt, int* val)
{
    printf("%s", prompt);
    scanf("%d", val);
}

void read_float(const char* prompt, float* val)
{
    printf("%s", prompt);
    scanf("%f", val);
}

void read(float mas[], int br)
{
    for (int i = 0; i < br; i++)
        read_float("Vavedete element:", mas + i);
}
```

```
void print(float mas[], int br)
{
    for (int i = 0; i < br; i++)
        printf("Element[%d] = %.2f\n", i + 1, mas[i]);
}

void min_max(float mas[], int n, float* min, float* max)
{
    *min = *max = mas[0];
    for (int i = 1; i < n; i++)
    {
        if (mas[i] < *min)
            *min = mas[i];
        if (mas[i] > *max)
            *max = mas[i];
    }
}
```

Намиране на съответни елементи в два масива и записване в трети

```
/*само една нова функция, всички
останали се преизползват*/

void find_equal(float m1[], float m2[],
                int n, float res[])
{
   for (int i = 0; i < n; i++)
      if (m1[i] == m2[i])
         res[i] = m1[i];
      else
         res[i] = 0;
}
```

```
int main()
{
   float mas[100], mas2[100], mas3[100];
   float min, max;
   int br;

   read_int("Vavedete broi elementi:", &br);

   read(mas, br);
   read(mas2, br);

   find_equal(mas, mas2, br, mas3);

   print(mas3, br);
}
```