**Определение**

*У*казатель – это переменная, которая хранит адрес области памяти. Указатель, как и переменная, имеет тип. Синтаксис объявления указателей

[?](https://learnc.info/c/pointers.html)

|  |
| --- |
| <тип> \*<имя>; |

Например   
float \*a;

long long \*b;

Два основных оператора для работы с указателями – это оператор & взятия адреса, и оператор \* разыменования. Рассмотрим простой пример.

[?](https://learnc.info/c/pointers.html)

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21 | #include <conio.h>  #include <stdio.h>    void main() {      int A = 100;      int \*p;        //Получаем адрес переменной A      p = &A;        //Выводим адрес переменной A      printf("%p\n", p);        //Выводим содержимое переменной A      printf("%d\n", \*p);        //Меняем содержимое переменной A      \*p = 200;        printf("%d\n", A);      printf("%d", \*p);        getch();  } |

Рассмотрим код внимательно, ещё раз

[?](https://learnc.info/c/pointers.html)

|  |  |
| --- | --- |
| 5 | int A = 100; |

Была объявлена переменная с именем *A*. Она располагается по какому-то адресу в памяти. По этому адресу хранится значение 100.

[?](https://learnc.info/c/pointers.html)

|  |  |
| --- | --- |
| 6 | int \*p; |

Создали указатель типа ***int***.

[?](https://learnc.info/c/pointers.html)

|  |  |
| --- | --- |
| 9 | p = &A; |

Теперь переменная *p* хранит адрес переменной *A*. Используя оператор \* мы получаем доступ до содержимого переменной *A*.   
Чтобы изменить содержимое, пишем

[?](https://learnc.info/c/pointers.html)

|  |  |
| --- | --- |
| 18 | \*p = 200; |

После этого значение *A* также изменено, так как она указывает на ту же область памяти. Ничего сложного.   
Теперь другой важный пример

[?](https://learnc.info/c/pointers.html)

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16 | #include <conio.h>  #include <stdio.h>    void main() {      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));        getch();  } |

Несмотря на то, что переменные имеют разный тип и размер, указатели на них имеют один размер. Действительно, если указатели хранят адреса, то они должны быть целочисленного типа. Это тип, который ведёт себя как целочисленный, однако его размер зависит от разрядности системы. В большинстве случаев разницы между ними нет. Зачем тогда указателю нужен тип?

**Арифметика указателей**

*В*о-первых, указателю нужен тип для того, чтобы корректно работала операция разыменования (получения содержимого по адресу). Если указатель хранит адрес переменной, необходимо знать, сколько байт нужно взять, начиная от этого адреса, чтобы получить всю переменную.   
Во-вторых, указатели поддерживают арифметические операции. Для их выполнения необходимо знать размер.   
операция + N сдвигает указатель вперёд на N\*sizeof(тип) байт.  
Например, если указатель int \*p; хранит адрес CC02, то после p += 10; он будет хранить адрес СС02 + sizeof(int)\*10 = CC02 + 28 = CC2A (Все операции выполняются в шестнадцатиричном формате). Пусть мы создали указатель на начало массива. После этого мы можем "двигаться" по этому массиву, получая доступ до отдельных элементов.

[?](https://learnc.info/c/pointers.html)

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15 | #include <conio.h>  #include <stdio.h>    void main() {      int A[10] = {1, 2, 3, 4, 5, 6, 7, 8, 9, 10};      int \*p;        p = A;        printf("%d\n", \*p);      p++;      printf("%d\n", \*p);      p = p + 4;      printf("%d\n", \*p);        getch();  } |

Заметьте, каким образом мы получили адрес первого элемента массива

[?](https://learnc.info/c/pointers.html)

|  |  |
| --- | --- |
| 8 | p = A; |

Массив, по сути, сам является указателем, поэтому не нужно использовать оператор &. Мы можем переписать пример по-другому

[?](https://learnc.info/c/pointers.html)

|  |  |
| --- | --- |
| 8 | p = &A[0]; |

Получить адрес первого элемента и относительно него двигаться по массиву.   
Кроме операторов + и - указатели поддерживают операции сравнения. Если у нас есть два указателя a и b, то a > b, если адрес, который хранит a, больше адреса, который хранит b.

[?](https://learnc.info/c/pointers.html)

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21 | #include <conio.h>  #include <stdio.h>    void main() {      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);        if (a < b) {          printf("a < b");      } else {          printf("b < a");      }        getch();  } |

Если же указатели равны, то они указывают на одну и ту же область памят

**Примеры**

Теперь несколько примеров работы с указателями  
1. Пройдём по массиву и найдём все чётные элементы.

[?](https://learnc.info/c/pointers.html)

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24 | #include <conio.h>  #include <stdio.h>    void main() {      int A[10] = {1, 2, 3, 4, 5, 6, 7, 8, 9, 10};      int even[10];      int evenCounter = 0;      int \*iter, \*end;        //iter хранит адрес первого элемента массива      //end хранит адрес следующего за последним "элемента" массива      for (iter = A, end = &A[10]; iter < end; iter++) {          if (\*iter % 2 == 0) {              even[evenCounter++] = \*iter;          }      }        //Выводим задом наперёд чётные числа      for (--evenCounter; evenCounter >= 0; evenCounter--) {          printf("%d ", even[evenCounter]);      }        getch();  } |

2. Когда мы сортируем элементы часто приходится их перемещать. Если объект занимает много места, то операция обмена местами двух элементов будет дорогостоящей. Вместо этого можно создать массив указателей на исходные элементы и отсортировать его. Так как размер указателей меньше, чем размер элементов целевого массива, то и сортировка будет происходить быстрее. Кроме того, массив не будет изменён, часто это важно.

[?](https://learnc.info/c/pointers.html)

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36  37  38  39  40  41  42 | #include <conio.h>  #include <stdio.h>    #define SIZE 10    void main() {      double unsorted[SIZE] = {1.0, 3.0, 2.0, 4.0,  5.0, 6.0, 8.0, 7.0, 9.0, 0.0};      double \*p[SIZE];      double \*tmp;      char flag = 1;      unsigned i;        printf("unsorted array\n");      for (i = 0; i < SIZE; i++) {          printf("%.2f ", unsorted[i]);      }      printf("\n");        //Сохраняем в массив p адреса элементов      for (i = 0; i < SIZE; i++) {          p[i] = &unsorted[i];      }        do {          flag = 0;          for (i = 1; i<SIZE; i++) {              //Сравниваем СОДЕРЖИМОЕ              if (\*p[i] < \*p[i-1]) {                  //обмениваем местами АДРЕСА                  tmp = p[i];                  p[i] = p[i-1];                  p[i-1] = tmp;                  flag = 1;              }          }      } while(flag);        printf("sorted array of pointers\n");      for (i = 0; i < SIZE; i++) {          printf("%.2f ", \*p[i]);      }      printf("\n");        printf("make sure that unsorted array wasn't modified\n");      for (i = 0; i < SIZE; i++) {          printf("%.2f ", unsorted[i]);      }        getch();  } |

3. Более интересный пример. Так как размер типа char всегда равен 1 байт, то с его помощью можно реализовать операцию swap – обмена местами содержимого двух переменных.

[?](https://learnc.info/c/pointers.html)

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28 | #include <conio.h>  #include <conio.h>  #include <stdio.h>    void main() {      int length;      char \*p1, \*p2;      char tmp;      float a = 5.0f;      float b = 3.0f;        printf("a = %.3f\n", a);      printf("b = %.3f\n", b);        p1 = (char\*) &a;      p2 = (char\*) &b;      //Узнаём сколько байт перемещать      length = sizeof(float);      while (length--) {          //Обмениваем местами содержимое переменных побайтно          tmp = \*p1;          \*p1 = \*p2;          \*p2 = tmp;          //не забываем перемещаться вперёд          p1++;          p2++;      }        printf("a = %.3f\n", a);      printf("b = %.3f\n", b);        getch();  } |

В этом примере можно поменять тип переменных *a* и *b* на ***double*** или любой другой (с соответствующим изменением вывода и вызова ***sizeof***), всё равно мы будет обменивать местами байты двух переменных.

4. Найдём длину строки, введённой пользователем, используя указатель

[?](https://learnc.info/c/pointers.html)

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17 | #include <conio.h>  #include <stdio.h>    void main() {      char buffer[128];      char \*p;      unsigned length = 0;        scanf("%127s", buffer);      p = buffer;      while (\*p != '\0') {          p++;          length++;      }        printf("length = %d", length);      getch();  } |

Обратите внимание на участок кода

[?](https://learnc.info/c/pointers.html)

|  |  |
| --- | --- |
| 11  12  13  14 | while (\*p != '\0') {      p++;      length++;  } |

его можно переписать

[?](https://learnc.info/c/pointers.html)

|  |  |
| --- | --- |
| 11  12  13  14 | while (\*p != 0) {      p++;      length++;  } |

или

[?](https://learnc.info/c/pointers.html)

|  |  |
| --- | --- |
| 11  12  13  14 | while (\*p) {      p++;      length++;  } |

или, убрав инкремент в условие

[?](https://learnc.info/c/pointers.html)

|  |  |
| --- | --- |
| 11  12  13 | while (\*p++) {      length++;  } |