Содержание

указатели

Указатели представляют собой объекты, значением которых служат адреса других объектов (переменных, констант, указателей) или функций.

```
// определение указателя
тип_данных* название_указателя;
```

Указатель должен быть того же типа, что и объект, адрес которого он хранит. Указатель типа int не может указывать на переменную float. Указатель на void может хранить адрес объекта любого типа.

Указателю можно присвоить (=) значение другово указателя (хранимый в нем адрес). Операции сравнения применяются только к указателям одного типа и константе NULL.

Если объект (например int) занимает в памяти больше 1 байта - указатель будет ссылаться на ячейку, где храниться первый байт объекта.

Если мы не хотим, чтобы указатель указывал на какой-то конкретный адрес, то можно присвоить ему условное нулевое значение с помощью константы NULL, которая определена в заголовочном файле stdio.h:

```
int *pa = NULL;
```

Иногда требуется присвоить указателю одного типа значение указателя другого типа. В этом случае следует выполнить операцию приведения типов.

Получение адреса

Для получения адреса к переменной применяется операция &

```
#include <stdio.h>

int main(void) {
  int x = 10;
  int *p;
  p = &x;
  printf("%p \n", p); // например: 0060FEA8
  return 0;
}
```

таким же образом (&) можно получить адрес самого указателя.

Разыменование указателя

Для получения значения по адресу применяется операция * или операция разыменования (dereference operator).

```
#include <stdio.h>

int main(void){
  int x = 10;
  int *p;
  p = &x;
  printf("Address = %p \n", (void*) p); // Address = 0060FEA8
  printf("x = %d \n", *p); // x = 10
  return 0;
}
```

Константы

Мы можем манипулировать значениями констант через указатели

```
#include <stdio.h>

int main(void) {
  const int cx = 10;
  // получаем адрес константы, преобразуем в указатель типа  int*  и изменяем по нему значение
  *(int*) &cx = 20;
  printf("cx: %d\n", cx); // cx: 20
}
```

Сам указатель тоже может быть const. Константный указатель может указывать и на обычную автоматическую переменную. Константному указателю нельзя изменить адрес, который в нем храниться, зато можно изменить значение по этому адресу.

Константный указатель на константу не может менять ни свой адрес, ни значение по адресу.

Массивы указателей

Массив указателей определяется одним из трех способов:

```
int array[] = {1, 2, 3, 4};
int *p1[3];
int *p2[] = { &array[1], &array[2], &array[0] };
int *p3[3] = { &array[3], &array[1], &array[2] };
```

Вместо *p[i] для доступа к элементу из array мы могли бы написать **(p+i):

- `p+i` - к адресу в указателе p прибавляем число i и таким образом перемещаемся по указателям в массиве p.
- `(p+i)` - разыменовываем i-тый указатель в массиве и в результате получаем адрес одного из элементов из массива array.
- `**(p+i)` - получаем значение по полученному на предыдущем шаге адресу элемента из массива array.

Указатель на массив

тип_элементов_массива (*имя_переменной_указателя)[количество_элементов];

```
int array1[] = {11, 12, 13};
int (*pa1)[3] = &array;  // указатель на массив с 3 элементами int

int array2[] = {11, 12, 13, 14, 15};
int (*pa2)[] = &array2; // указатель типа int (*)[5]

int (*pa3)[3] = &array2; // некоторые компиляторы такое не пропускают
```

Указатель и массив строк

Соответственно если указатель типа char можно представить в виде строки, то массив указателей типа char представляет собой массив строк

```
#include <stdio.h>

int main(void) {
  char *fruit[] = {"apricot", "apple", "banana", "lemon", "orange"};
  for(int i=0; i < 5; i++) {
    printf("%s \n", fruit[i]);
    // аналогично
    printf("%s \n", *(fruit + i));
  }
  return 0;
}
```

Указатели на указатели

Если указатель хранит адрес переменной, то указатель на указатель хранит адрес указателя, на который он указывает[1)].

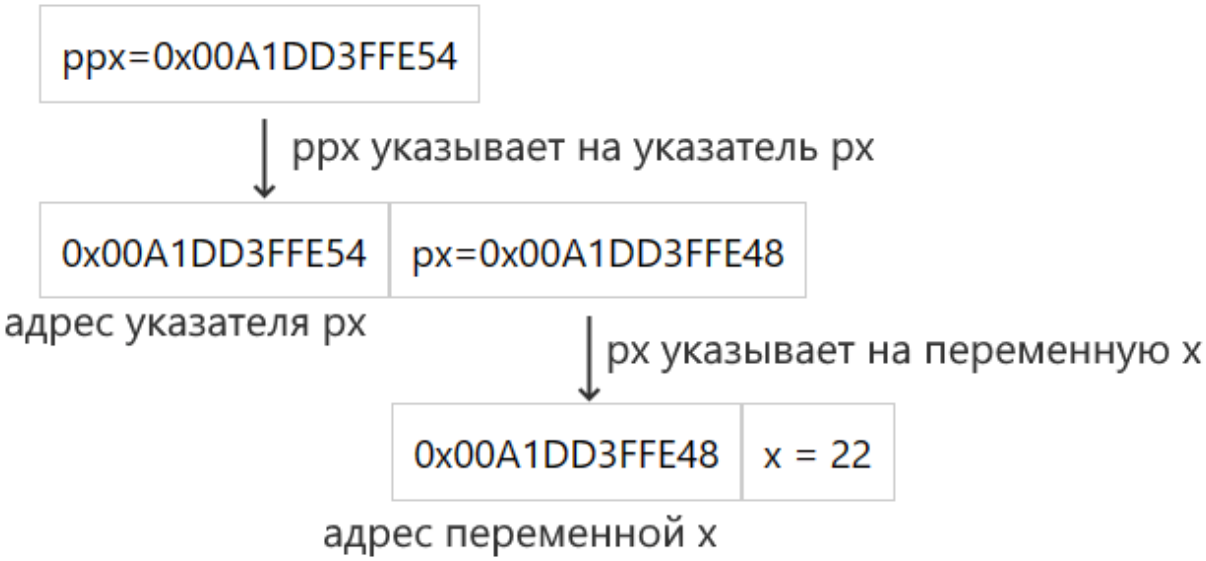
```
#include <stdio.h>

int main(void) {
  int  x = 22;
  int *px = &x; // указатель px хранит адрес переменной x
```

```
  int **ppx = &px; // указатель ppx хранит адрес указателя px

  printf("Address of px: %p \n", (void *)ppx);
  printf("Address of x: %p \n", (void *)*ppx);
  printf("Value of x: %d \n", **ppx);
  return 0;
}
```

Здесь указатель ppx хранит адрес указателя px. Поэтому через выражение *ppx можно получить значение, которое хранится в указателе px - адрес переменной x. А через выражение **ppx можно получить значение по адресу из px, то есть значение переменной x.



Указатели на функцию

Указатель на функцию представляет собой выражение или переменную, которые используются для представления адреса функции. Указатель на функцию содержит адрес первого байта в памяти, по которому располагается выполняемый код функции. Самым распространенным указателем на функцию является ее имя.

общий синтаксис:
тип (*имя_указателя) (типы_параметров);

```
#include <stdio.h>

void hello() {
  printf("Hello, World \n");
}

void goodbye() {
  printf("Good Bye, World \n");
}
```

```c
int sum(int x, int y) {
  return x + y;
}

int subtract(int x, int y) {
  return x - y;
}

int main(void) {
  // определяем указатель на функцию
  void (*message) (void);

  message=hello; // указатель указывает на функцию  hello
  message(); // вызываем функцию, на которую указыывет указатель
  message = goodbye; // указатель указывает на функцию  goodbye
  message(); // вызываем функцию, на которую указыывет указатель

  int a = 10;
  int b = 5;
  int result;
  int (*operation)(int, int);
  operation=sum;
  result = operation(a, b);
  printf("result = %d \n", result); // result=15
  operation = subtract;
  result = operation(a, b);
  printf("result = %d \n", result); // result=5

  return 0;
}
```

Массивы указателей на функции

тип (*имя_массива[размер]) (параметры)

```c
#include <stdio.h>

void sum(int x, int y) {
  printf("x + y = %d \n", x + y);
}

void subtract(int x, int y) {
  printf("x - y = %d \n", x - y);
}

void multiply(int x, int y) {
  printf("x * y = %d \n", x * y);
}
```

```c
int main(void) {
  int a = 10;
  int b = 5;
  void (*operations[3])(int, int) = {sum, subtract, multiply};

  // получаем длину массива
  int length = sizeof(operations)/sizeof(operations[0]);

  for(int i=0; i < length; i++) {
    operations[i](a, b); // вызов функции по указателю
  }

  return 0;
}
```

1)

Такие ситуации еще называются многоуровневой адресацией.