Воспоминания об указателях.

Евгений Линский

Типы

Название типа	Кол-во байт для хранения	Диапазон
char	1	$-2^7..2^7-1$
short	2	$-2^{15}..2^{15}-1$
int	4	$-2^{31}..2^{31}-1$
long	4	$-2^{31}..2^{31}-1$
long long	8	$-2^{63}..2^{63}-1$
unsigned char	1	$0..2^8-1$
unsigned short	2	$0..2^{16}-1$
unsigned int	4	$0..2^{32}-1$
unsigned long	4	$0..2^{32}-1$
unsigned long long	8	$0..2^{64}-1$
float	4	$1,4 \cdot 10^{-45}..3.4 \cdot 10^{38}$
double	8	$4,94 \cdot 10^{-324}..1.79 \cdot 10^{308}$

В чем подвох?

Зависимость от платформы

1. На самом деле размеры типов зависят от платформы (процессор, ОС, компилятор)
2. int — “естественный” тип (компьютеру проще работать: ширина регистров, особенности набора инструкций)
3. На самом деле, например: $sizeof(short) \leq sizeof(int) \leq sizeof(long)$
4. *sizeof* — оператор языка (не функция), во время компиляции заменяется на размер типа

Массивы

Одномерные:

```
1  int array[10]; // размер  10*sizeof(int)
2  //Инициализация:
3  int array[10] = {0, 1, 2, 3, 4, 5, 6, 7, 8, 9};
4  int array[10] = {0}; // обнулить
5  int array[] = {0, 1, 2, 3, 4, 5, 6, 7, 8, 9};
6  //для типа char:
7  char array[] = {0, 1, 2, 3, 4, 5, 6, 7, 8, 9};
8  char array[] = {'H', 'e', 'l', 'l', 'o'};
9  char array[] = "Hello"; // размер?
```

Массивы

Одномерные:

```
1  int array[10]; // размер  10*sizeof(int)
2  //Инициализация:
3  int array[10] = {0, 1, 2, 3, 4, 5, 6, 7, 8, 9};
4  int array[10] = {0}; // обнулить
5  int array[] = {0, 1, 2, 3, 4, 5, 6, 7, 8, 9};
6  //для типа char:
7  char array[] = {0, 1, 2, 3, 4, 5, 6, 7, 8, 9};
8  char array[] = {'H', 'e', 'l', 'l', 'o'};
9  char array[] = "Hello"; // размер?
```

1. Рзамер 6 (завершающий 0)!

Указатели

1. Указатель (pointer) — число, адрес (т.е. смещение от начала) соответствующего элемента в памяти
2. int* p; — указатель на ячейку, в которой хранится int (в ‘p’ будет хранится адрес)
3. Количество байт для хранения указателя зависит от архитектуры компьютера (на x86 сейчас — 64 бита)
4. sizeof(int*) == sizeof(char*) == sizeof(double*) etc

Указатели

```
1    int a = 42;
2    int *p = &a; // & -- взять адрес a
3    int b = *p; // взять значение по адресу p (разыменовать)
4    printf("%p", p); // вывести адрес
```

Указатели

Сдвиг зависит от типа объекта, на который указывает указатель.

```
1    int array[5] = {1, 2, 3, 4, 5};
2    char str[] = "hello";
3    int *pi = array; // pi = &array[0]
4    char *pc = str; // pc = &str[0]
5    pi += 1; // сдвиг адреса на sizeof(int)
6    pc += 1; // сдвиг адреса на sizeof(char)
```

array[i] == i[array]:

```
1  array[i] --> *(array + i)
2  i[array] --> *(i + array)
```

Различие между разными видами указателей

```
1  char str[4];
2  char *pc = &c[0]; // &c[0] - адрес нулевого элемента массива
3                    // или адрес массива
4  int *pi = pc; // C -- ok, C++ -- error (разные типы)
5  int *pi = (int*)pc; // C -- ok, C++ -- ok
```

```
1  char array[10];
2  char *p;
3  p = array; // в p передается адрес массива
4             // (адрес нулевого элемента)
5  array = p; // это ошибка
```

Применение - 1

```
1  void swap(double a, double b){
2    double tmp = a;
3    a = b;
4    b = tmp;
5  }
6  int main() {
7    double c = 3; double d = 4;
8    swap(c, d);
9    return 0;
10 }
```

Применение - 1

```
1  void swap(double a, double b){
2    double tmp = a;
3    a = b;
4    b = tmp;
5  }
6  int main() {
7    double c = 3; double d = 4;
8    swap(c, d);
9    return 0;
10 }
```

1. Ничего не получится.

Применение - 1

```
1  void swap(double a, double b){
2    double tmp = a;
3    a = b;
4    b = tmp;
5  }
6  int main() {
7    double c = 3; double d = 4;
8    swap(c, d);
9    return 0;
10 }
```

1. Ничего не получится.
2. Функция работает с копиями параметров (a и b поменяются, c и d нет).

Применение - 1

```
1  void swap(double *pa, double *pb){
2    double tmp = *pa;
3    *pa = *pb;
4    *pb = tmp;
5  }
6  int main() {
7    double c = 3; double d = 4;
8    swap(&c, &d);
9    return 0;
10 }
```

Применение - 2

Передать в функцию большой объект и не копировать его!

```
1  char str[] = "Hello";
2  int l = strlen(str);
```

Применение - 3

```
int strlen(char* ptr){
  int len = 0;
  while (ptr[len] != '\0'){
    ++len;
  }
  return len;
}
```

```
int strlen(char* ptr){
  char* p = ptr;
  while (*p != '\0'){
    ++p;
  }
  return p - ptr;
}
```

Применение - 3

```
int strlen(char* ptr){
  int len = 0;
  while (ptr[len] != '\0'){
    ++len;
  }
  return len;
}
```

```
int strlen(char* ptr){
  char* p = ptr;
  while (*p != '\0'){
    ++p;
  }
  return p - ptr;
}
```

1. ptr[len] -> *(ptr+len) одно сложение!

Применение - 3

```
int strlen(char* ptr){
  int len = 0;
  while (ptr[len] != '\0'){
    ++len;
  }
  return len;
}
```

```
int strlen(char* ptr){
  char* p = ptr;
  while (*p != '\0'){
    ++p;
  }
  return p - ptr;
}
```

1. ptr[len] -> *(ptr+len) одно сложение!
2. '\0' — символ с кодом 0.

Применение - 3

```
int strlen(char* ptr){
  int len = 0;
  while (ptr[len] != '\0'){
    ++len;
  }
  return len;
}
```

```
int strlen(char* ptr){
  char* p = ptr;
  while (*p != '\0'){
    ++p;
  }
  return p - ptr;
}
```

1. ptr[len] –> *(ptr+len) одно сложение!
2. '\0' — символ с кодом 0.
3. (p - ptr) — длина строки (складывать указатели нельзя) .

const у указателя

const защищает то, что *перед* ним.

```
char s1[] = "hello";
char s2[] = "bye";
char const * p1 = s1;
p1[0] = 'a'; // compilation error
p1 = s2; // ok
char * const p2 = s1;
p2[0] = 'a'; // ok
p2 = s2; // compilation error
char const * const p3 = s1;
```

Но можно и так:

```
const char * p1; // equal to char const * p1;
```

const у указателя

```
1  size_t strlen(const char * s);
2  int main() {
3    char str[] = "Hello";
4    site_t s = strlen(str);
5  }
```

1. Что хотел сказать программист?

const у указателя

```
1  size_t strlen(const char * s);
2  int main() {
3    char str[] = "Hello";
4    site_t s = strlen(str);
5  }
```

1. Что хотел сказать программист?
2. Функция *strlen* не изменяет свой аргумент. Например, программист в main может не делать копию *str* перед вызовом *strlen*.

const у указателя

```
1  void strange(const char * s) {
2    char* str = (char*)s;
3    str[0] = 'A'
4  }
```

```
1  char s[] = "Hello";
2  strange(s);
```

Все ли хорошо?

```
1  const char* const s = "Hello";
2  strange(s);
```

Все ли хорошо?

strchr

Задача: написать функцию, которая возвращает адрес первого вхождения символа в строку.

```
char* strchr(const char* s, char ch) {
  while (*s != 0) {
    if (*s == ch)
      return s;
    s++;
  }
  return NULL;
}
```

- Если функция ничего не нашла, то она возвращает нулевой адрес.
- В стандартной библиотеке языка C есть макрос # define NULL 0 для более явного обозначения нулевого адреса.

strchr

Задача: найти первое и второе вхождение

```
char s[] = "Hello, world!";
char* p1;
char* p2;

p1 = strchr(s, 'o');
p2 = strchr(p1, 'o');
```

Что не так?

strchr

Задача: найти первое и второе вхождение

```
char s[] = "Hello, world!";
char* p1;
char* p2;

p1 = strchr(s, 'o');
p2 = strchr(p1, 'o');
```

Что не так?

1. Есть ли нет первого вхождения, то будет ошибка (p1 — NULL)

strchr

Задача: найти первое и второе вхождение

```
char s[] = "Hello, world!";
char* p1;
char* p2;

p1 = strchr(s, 'o');
p2 = strchr(p1, 'o');
```

Что не так?

1. Есть ли нет первого вхождения, то будет ошибка (p1 — NULL)
2. Если есть первое вхождение, найдет его повторно

strchr

Задача: найти первое и второе вхождение

```
char s[] = "Hello, world!";
char* p1;
char* p2;

p1 = strchr(s, 'o');
if (p1 != NULL)
  p2 = strchr(p1 + 1, 'o');
```

strcpy

```
char* strcpy(char* to, const char* from) {
  char *save = to;
  for (; (*to = *from) != '\0'; ++from, ++to);
  return(save);
}
```

Скопируем ли мы последний символ 0?

strcat

```
char* strcat(char *dest, const char *src) {
  strcpy (dest + strlen(dest), src);
  return dest;
}
```

inplace reverse

```
void strrev(char* head) {
  char *tail = head;
  while(*tail) ++tail;
  --tail;
  for( ; head < tail; ++head, --tail) {
      char h = *head, t = *tail;
      *head = t;
      *tail = h;
  }
}
```

strtok

```
char* strtok( char* str, const char* delim );
```

```
    char input[] = "one + two * (three - four)!";
    const char* delimiters = "! +- (*)";
    char* token = strtok(input, delimiters);
    while (token) {
        printf("%s\n", token);
        token = strtok(NULL, delimiters);
    }
```

Как это работает? Есть ли подсказка в объявлении функции?