Лекция 2. Указатели.

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 (прроцессор, 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

Самостоятельно

- Представление отрицательных целых чисел (дополнительный код)
- Представление чисел с плавающей точкой (floating point)
- Приведение типов (явное, неявное)

NB! Будет тест.

Массивы

Одномерные:

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

Двумерные:

```
int m[10][10];
int m[2][2] = { {1,2} , {3,4} };
```

Массивы

```
int array[10] = {0, 1, 2, 3, 4, 5, 6, 7, 8, 9};
int a = array[index];
array[10] = 1; array[-1] = 1;
```

“Труднонаходимые” ошибки

1. Выход за пределы массива не контролируется компилятором
2. Исходы:
   - программа корректно отработала (не задели данные)
   - программа некорректно (зависла, неверный ответ) отработала (задели данные)
   - ОС аварийно завершила программу (задели чужие данные)
3. Исход зависит от того, что (данные, другие программы) в памяти в данный момент

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

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

Указатели

Сдвиг зависит от типа объекта, на который указывает указатель.

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

array[i] == i[array]:

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

Различие между разными видами указателей

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

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

Применение - 1

```
void swap(double a, double b){
  double tmp = a;
  a = b;
  b = tmp;
}
int main() {
  double c = 3; double d = 4;
  swap(c, d);
  return 0;
}
```

Применение - 1

```
void swap(double a, double b){
  double tmp = a;
  a = b;
  b = tmp;
}
int main() {
  double c = 3; double d = 4;
  swap(c, d);
  return 0;
}
```

1. Ничего не получится.

Применение - 1

```
void swap(double a, double b){
  double tmp = a;
  a = b;
  b = tmp;
}
int main() {
  double c = 3; double d = 4;
  swap(c, d);
  return 0;
}
```

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

Применение - 1

```
void swap(double *pa, double *pb){
  double tmp = *pa;
  *pa = *pb;
  *pb = tmp;
}
int main() {
  double c = 3; double d = 4;
  swap(&c, &d);
  return 0;
}
```

Применение - 2

Передать в функцию большой объект и не копировать его!

```
char str[] = "Hello";
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) .