Основы программирования

Лекция № 3, 16 марта 2017 г.

http://xkcd.ru/371

Указатели (продолжение)

Указатель, никуда не указывающий

NULL — специальное константное значение, символизирующее, что указатель не указывает ни на какую память. Объявлено в заголовочном файле `stdlib.h`.

```
int* ptr = NULL;
int value = *ptr; // run time error
*ptr = 37; // run time error
```

Разные названия, но суть одна (segmentation fault, segfault, access violation, «Программа выполнила недопустимую операцию...», ...).

Указатель, указывающий хоть куда

`void*` — специальный тип указателя, который может указывать на любые данные в памяти. Может быть приведен к любому другому типу указателей и обратно.

```
double x = 37;
double* px = &x;
void* p = px;
int* py = p;
```

Минутка философии: «Какова природа void?» — спросил учитель, ...

`http://thecodelesscode.com/case/5?lang=ru`

Динамическая память

Преимущества динамической памяти

- Выделяется и освобождается динамически по запросу программы.
- Размер задается динамически.

Выделение блока памяти

```
void* malloc(size_t size);
```

Функция выделяет блок памяти размером `size` байт и возвращает указатель на начало блока. В случае, если память выделить не получилось, возвращает `NULL`. Объявлена в заголовочном файле `stdlib.h`.

Освобождение блока памяти

```
void free(void* ptr);
```

Функция освобождает блок памяти. Если `ptr` равен `NULL`, ничего не делает. Объявлена в заголовочном файле `stdlib.h`.

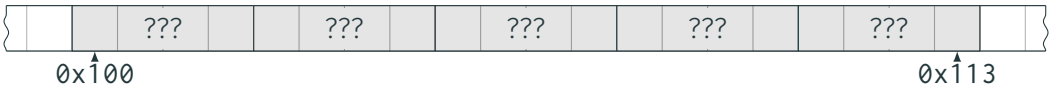
После вызова значение указателя `ptr` остается прежним, но разыменовывать его нельзя.

Неиспользуемую память нужно обязательно освобождать, иначе рано или поздно она может кончиться (утечка памяти).

Пример: массив динамического размера

```
int n = read_number(); // 5
int* p; // ???
```

Пример: массив динамического размера




```
int n = read_number(); // 5
int* p; // 0x100
p = malloc(n * sizeof(int));
if (p == NULL) { /* error */ }
```

Пример: массив динамического размера


```
int n = read_number(); // 5
int* p; // 0x100
p = malloc(n * sizeof(int));
if (p == NULL) { /* error */ }
p[0] = p[n/2] = p[n-1] = 37;
```

Пример: массив динамического размера


```
int n = read_number(); // 5
int* p; // 0x100
p = malloc(n * sizeof(int));
if (p == NULL) { /* error */ }
p[0] = p[n/2] = p[n-1] = 37;
free(p);
```

Указатели на функции

Функции как данные

В языке C с функциями можно работать как с данными:

```
double (*my_func)(double, double) = pow;
double x = my_func(3, 2); // x = 9.0
```

Функция — это набор байтов в памяти, кодирующих тело этой функции с помощью машинных команд. Значит, можно взять адрес этого «набора байтов».

Более формально:

```
double (*my_func)(double, double) = &pow;
double x = (*my_func)(3, 2); // x = 9.0
```

Пример численного вычисления производной в точке

```
double diff(double x, double (*f)(double)) {
    double dx = 0.01;
    return (f(x + dx) - f(x)) / dx;
}

double square(double x) {
    return x * x;
}

printf("%g\n", diff(M_PI/3, sin)); // 0.495662
printf("%g\n", diff(M_PI/6, cos)); // -0.504322
printf("%g\n", diff(3, square));   // 6.01
```

Символы и кодировки

Символы в C

Символы задаются в одинарных кавычках, хранятся в переменных типа char.

```
char x = 'H';
char y = 'i';
printf("%c%c\n", x, y); // Hi
```

Кодировки

Кодировка — соответствие между символом и целочисленным кодом.

ASCII — фундаментальная 7-битная кодировка:

- 32 управляющих символа,
  - `'\0'`, код 0 — пустой символ,
  - `'\t'`, код 9 — табуляция,
  - `'\n'`, код 10 — перевод строки,
  - …
- 96 информационных символов.
  - `'0'..'9'` — цифры,
  - `'a'..'z'`, `'A'..'Z'` — буквы,
  - `','`, `'?'`, `...` — пунктуация.
  - …

Символы как числа

```
char a = 'X';
printf("%c\n", a); // X
printf("%d\n", a); // 88

char b = a + 1;
printf("%c\n", b); // Y
```

Национальные символы

А что делать, если хочется использовать кириллицу, умляуты и иероглифы?

8-битные кириллические кодировки, базирующиеся на ASCII: КОИ-8, Windows-1251, ...

На практике как-то так: ишЊюърџ §ыхъЋЊшєшърішџ

Юникод

Юникод — многобайтовая кодировка, покрывающая почти все письменные языки.

Позволяет кодировать 1 112 064 символов. В версии 9.0, июнь 2016 г., используется лишь 128 237.

Содержит все национальные символы, математические символы, символы древних письменностей и многое другое, включая ☺ 😐 ☹ 😁 😂 😄 😇 😉 😈 😋 😍 🤯 и даже 😸!

Строки

Строки

Строка — это набор символов. В языке C строки представляются в виде массивов символов типа `char`. Константные строки задаются в двойных кавычках.

Длина строки явно не хранится: после последнего символа строки хранится специальный символ `'\0'`.

```
char* str = "Hi";
printf("%s\n", str);                    // Hi
printf("%d - %c\n", str[0], str[0]); // 72 - H
printf("%d - %c\n", str[1], str[1]); // 105 - i
printf("%d - %c\n", str[2], str[2]); // 0 -
```

Тонкость с инициализацией строк

- `test` — указатель на константную строку.

```
char* test = "cat";
test[2] = 'r'; // run time error
```

- `test` — массив, проинициализированный константной строкой.

```
char test[] = "cat";
test[2] = 'r';
printf("%s\n", test); // car
```

Конец третьей лекции