Лекция 6. const в C. Обзор libc: stdio - I.

Евгений Линский

const у переменной

```
1  const float pi = 3.14159;
```

Зачем?

- Компилятор проверяет, что мы не изменим *pi* по ошибке.
- Дать больше информации программисту, читающему или использующему наш код.

```
1  void  print_hex(const int a) {
2    printf("%x",a);
3  }
4  int main() {
5    int b = 4;
6    print_hex(b);
7  }
```

Программист хотел подчеркнуть, что *print_hex* не меняет параметр. Разумно?

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*.

Стандартная библиотека (libc)

1. `cplusplus.com`
2. `cppreference.com`
3. MSDN

Обзор:

- *stdio.h* — ввод/вывод (файл, клавиатура, экран)
- stdlib.h — работа с памятью, алгоритмы
- string.h — работа со строками и массивами
- math.h — математические функции
- time.h — время

Лучше вызвать функции из стандартной библиотеки, а не писать самому!

libc: работа с устройствами

Работа с устройствами (файл) или ресурсами (память):

- Разделение полномочий в ОС: препятствует обращению программ к данным других программ и оборудованию.
- Ядро ОС исполняется в привилегированном режиме работы процессора.
- Для выполнения межпроцессной операции или операции, требующей доступа к оборудованию, программа обращается (системный вызов) к ядру.
- program –1–> libc –2–> OS
  - 1 — call
  - 2 — syscall

stdio

```
1 FILE* f1 = fopen("in.txt",....);// файл на диске
2 FILE* f2 = stdin; // можно читать с клавиатуры
3 FILE* f3 = stdout; // можно писать на экран
```

FILE — структура, описывающая абстракцию для ввода-вывода (файл на диске, клавиатура, экран). Что внутри:

1. Дескриптор — идентификатор (целое число) файла внутри ОС
2. Промежуточный буфер — быстрее накопить буфер, а потом за один системный вызов записать его на диск, чем для каждого байта делать отдельный системный вызов
3. Текущее положение в файле
4. Индикатор ошибки — была ли ошибка при последней операции
5. Индикатор конца файла — достигнут ли конец файла при последней операции

Напрямую с этими полями не работают, а используют функции stdio.

Текстовые и бинарные файла

1. На диске всегда байты, меняется только способ их интерпретации
2. Текстовый формат файла
   1. Интерпретируется как последовательность символов. Пример: число 100 записывается не как один байт, а как три символа '1' '0' '0' (3 байта).
   2. Есть спецсимволы: перевод строки, табуляция.
   3. Проблемы: разные кодировки, в том числе для спецсимволов (перевод строки '\n': Linux — 10, Windows — 10 13)
   4. Просто интерпретировать, но большой размер файла.
3. Бинарный формат файла
   1. Сложные форматы (bmp, wav, elf), для работы нужно описание. Пример: число 100 — как один байт.
   2. Еще пример. Заголовок: первые 4 байта ширина, вторые четыре байта высота. Данные: три байта RGB с выравниванием.
   3. Сложно интерпретировать, но компактный размер файла.

fopen

```
1  FILE*  f = fopen("in.txt", mode);
2  if( f == NULL ) {
3    // файл не открылся
4  }
5  fclose(f);
```

mode: r/w/a == читать/перезаписать/добавить в конец.
rt — в Windows при записи '\n' писать 10 13

1. Зачем делать fclose, если при закрытии программы все ресурсы ОС и так освободит?

fopen

```
1  FILE*  f = fopen("in.txt", mode);
2  if( f == NULL ) {
3    // файл не открылся
4  }
5  fclose(f);
```

mode: r/w/a == читать/перезаписать/добавить в конец.
rt — в Windows при записи '\n' писать 10 13

1. Зачем делать fclose, если при закрытии программы все ресурсы ОС и так освободит?
2. Число дескрипторов ограничено. На FILE тратится память.

fopen

```
1  FILE*  f = fopen("in.txt", mode);
2  if( f == NULL ) {
3    // файл не открылся
4  }
5  fclose(f);
```

mode: r/w/a == читать/перезаписать/добавить в конец.
rt — в Windows при записи '\n' писать 10 13

1. Зачем делать fclose, если при закрытии программы все ресурсы ОС и так освободит?
2. Число дескрипторов ограничено. На FILE тратится память.
3. Ограничения на работу с отрытым файлом (в Windows файл открытый на чтение нельзя удалить).