Семестр 1. Лекция 5. const в C. Обзор libc: stdio.

Евгений Линский

6 Октября 2017

const у переменной

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

Зачем?

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

```
void  print_hex(const int a) {
  printf("%x",a);
}
int main() {
  int b = 4;
  print_hex(b);
}
```

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

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

1. Что хотел сказать программист?

const у указателя

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

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

```
FILE *f1 = fopen("in.txt",....);// файл на диске
FILE *f2 = stdin; // можно читать с клавиатуры
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

```
FILE * f = fopen("in.txt", mode);
if( f == NULL ) {
  // файл не открылся
}
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

```
FILE * f = fopen("in.txt", mode);
if( f == NULL ) {
  // файл не открылся
}
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

```
FILE * f = fopen("in.txt", mode);
if( f == NULL ) {
  // файл не открылся
}
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).

Текстовые файлы

```
int a = 0; int b = 0;
FILE* fin = fopen("in.txt", "r");
FILE* fout = fopen("out.txt", "w");
fscanf(fin, "%d %f", &a, &b);
fprintf(fout, "%d %f", a, b);
fclose(fout);
fclose(fin);
```

```
char s[256];
fscanf(fin, "%s", s); // считывает до whitespace ' ',
'\n','\t'
```

1. В чем проблема?

http://www-inst.eecs.berkeley.edu/~cs161/fa08/papers/stack_smashing.pdf

Текстовые файлы

```
int a = 0; int b = 0;
FILE* fin = fopen("in.txt", "r");
FILE* fout = fopen("out.txt", "w");
fscanf(fin, "%d %f", &a, &b);
fprintf(fout, "%d %f", a, b);
fclose(fout);
fclose(fin);
```

```
char s[256];
fscanf(fin, "%s", s); // считывает до whitespace ' ',
'\n','\t'
```

1. В чем проблема?
2. Не контролируется максимальное число считанных символов
   http://www-inst.eecs.berkeley.edu/~cs161/fa08/papers/stack_smashing.pdf

Текстовые файлы

```
int a = 0; int b = 0;
FILE* fin = fopen("in.txt", "r");
FILE* fout = fopen("out.txt", "w");
fscanf(fin, "%d %f", &a, &b);
fprintf(fout, "%d %f", a, b);
fclose(fout);
fclose(fin);
```

```
char s[256];
fscanf(fin, "%s", s); // считывает до whitespace ' ',
'\n','\t'
```

1. В чем проблема?
2. Не контролируется максимальное число считанных символов http://www-inst.eecs.berkeley.edu/~cs161/fa08/papers/stack_smashing.pdf
3. *fgets(s, 256, fin)* читает до '\n'

printf/fprintf/sprintf

```
fprintf(stdout, ...); //printf
fscanf(stdin, ....); //scanf
char s1[] = "3  4";
sscanf(s2,"%d %d", &a, &b);
char s2[256];
sprintf(s2, "%d + %d = %d", a, b, c);
```

1. Все это небыстро, т.к. внутри функции нужно разобрать форматную строку
2. Технология: функция с переменным числом параметров (см. *va_arg*)

Бинарные файлы

```
FILE* fin = fopen("in.txt", "r");
FILE* fout = fopen("out.txt", "w");
int array[100];
//размер одного элемента == 4, колво элементов == 100
fread(array, sizeof(int), 100, fin);
fwrite(array, sizeof(int), 50, fout);
fclose(fout);
fclose(fin);
```

А еще что можно?

1. fseek — переместиться на заданную позицию в файле (удобно пропускать ненужные поля в заголовке бинарного файла)
2. ftell — возвращает текущую позицию (как узнать размер файла?)

Ошибки

```
size_t res1 = fread(array, 100 * sizeof(int), 1, fin1);
size_t res2 = fread(array, sizeof(int), 100, fin2);
int res3 = fscanf(fin, "%d %f", &a, &b); //int т.к.
может быть EOF (обычно -1)
```

1. fread — число считанных **элементов**
2. fscanf — число считанных **элементов по формату**

Ошибки

```
while(!feof(fin)) {
  fread(...)
  if(ferror(fin)) {
    ...
  }
}
```

1. feof — возвращает индикатор конца файла
2. ferror — возвращает индикатор ошибки

fflush

```
int main() {
  FILE *fout = fopen("...", "...");
  ...
  int a = 3; int b = 5;
  fprintf(fout,  "%d %d", a, b);
  int *array = malloc(10000000);
  if(array == NULL) return -1
  ...
  fclose(fout);
  return 0;
}
```

1. В чем проблема?

fflush

```
int main() {
  FILE *fout = fopen("...", "...");
  ...
  int a = 3; int b = 5;
  fprintf(fout,  "%d %d", a, b);
  int *array = malloc(10000000);
  if(array == NULL) return -1
  ...
  fclose(fout);
  return 0;
}
```

1. В чем проблема?
2. "3 5" может осесть в буфере внутри FILE

fflush

```
int main() {
  FILE *fout = fopen("...", "...");
  ...
  int a = 3; int b = 5;
  fprintf(fout, "%d %d", a, b);
  int *array = malloc(10000000);
  if(array == NULL) return -1
  ...
  fclose(fout);
  return 0;
}
```

1. В чем проблема?
2. "3 5" может осесть в буфере внутри FILE
3. Можно вызвать *fflush(fout)* для принудительного сбрасывания буфера