VARIADIC

Минимум о функциях с переменным числом аргументов

К. Владимиров, Intel, 2019
mail-to: konstantin.vladimirov@gmail.com

Перевод в строку и конкатенация

- Постановка задачи: нужно собрать строчку из строки, двух чисел и ещё одной строки

```
// "ab", 42, 1, "cd" → "ab 42 1 cd"
void strange_concat(char *dest, char const *s1,
                    int d1, int d2, char const *s2) {
  // TODO: ваш код здесь
}
```

- Будем считать, что размер dest заведомо достаточен
- Как бы вы это сделали?

Загадочный sprintf

- Самый простой способ: просто напечатать всё это в строку

```
// "ab", 42, 1, "cd" → "ab 42 1 cd"
void strange_concat(char *dest, char const *s1,
                    int d1, int d2, char const *s2) {
  sprintf(dst, "%s %d %d %s", s1, d1, d2, s2);
}
```

- Функция sprintf удивительно обобщённая: она позволяет скидывать всё что угодно в строчку и часто пользоваться ей удобнее, чем специфичными
- Хорошо. Но как написать саму функцию sprintf?
- Даже проще: что самое удивительное в функции sprintf?

Вариабельные функции

- Самое удивительное в функции sprintf то, что она берёт сколько угодно аргументов
- Давайте сначала попробуем написать функцию, которая брала бы сколько угодно целых чисел и складывала их

```
int x = sum_all(4, 10, 14, 24, 40); // x == 88
```

- Первый параметр это количество аргументов (иначе откуда его узнать?)
- Кажется её логика попроще

Вариабельные функции

- Произвольное количество аргументов после троеточия

```
int sum_all (int n, ...) {
  int res = 0;

  // здесь нужно просуммировать все аргументы

  return res;
}
```

- Здесь три точки это не сокращение на слайде, это легальный синтаксис
- Остаётся вопрос как всё-таки поулчить доступ к аргументам?

Функции из stdarg

- Список аргументов создаётся через `va_list`

```
va_list args;
```

- Аргумент с которого начинаются вариабельные отмечается через `va_start`

```
va_start(args, n);
```

- Каждый аргумент вынимается через `va_arg` с указанием типа

```
va_arg(args, int);
```

- В конце всё завершается через `va_end`

```
va_end(args);
```

Пример: суммирование целых

- Собираем всё вместе: функция суммирует целые числа

```
int sum_all(int n, ...) {
  int res = 0;
  va_list args;
  va_start(args, n);
  for (int i = 0; i < n; ++i)
    res += va_arg(args, int);
  va_end(args);
  return res;
}
```

- Теперь заработает: `x = sum_all(4, 10, 14, 24, 40); // x == 88`

Именно так работают printf и scanf

- Функции printf и scanf объявлены следующим образом

```
int printf(const char *format, ...);
```

```
int scanf(const char *format, ...);
```

- Они тоже принимают произвольное число параметров и используют строку формата чтобы установить типы
- Любая ошибка в типах ведёт к непоправимым последствиям
- И конечно, именно так работает и sprintf
- Но прежде чем мы до него дойдём, ещё одно простое применение

Вариабельная функция-конструктор

- Все помнят (см. Problem MP) что полином мы представляем как:

```
struct Poly { unsigned n; int *p; };
```

- Можно написать вариабельную функцию-конструктор полинома
- $A(x) = x^3 + 3x^2 + 4x + 7$

```
struct Poly create_poly (unsigned n, ...) {
   //  TODO: выделить память и заполнить её
}

struct Poly A = create_poly(4, 1, 3, 4, 7);
```

- Напишите эту функцию!

Многоликий printf и scanf

- Основные формы:

```
int printf(const char *format, ...);
int scanf(const char *format, ...);
int fprintf(FILE *f, const char *format, ...);
int fscanf(FILE *f, const char *format, ...);
int sprintf(char *s, const char *format, ...);
int sscanf(char *s, const char *format, ...);
```

- По сути обычный printf это fprintf где вместо первого аргумента stdout

Обсуждение

- Можем ли мы имея fprintf написать printf?

```
int printf(const char *format, ...) {
  // как-то вызвать fprintf
}
```

Обсуждение

- Можем ли мы имея fprintf написать printf?

```
int printf(const char *format, ...) {
  // как-то вызвать fprintf
}
```

- Увы, в языке нет способа из функции "пробросить троеточие"
- Можно написать макрос, но мы хотим избежать макросов
- А что если передать va_list?

Волшебство vfprintf

- Теперь и printf и fprintf можно реализовать в терминах новой функции

```
int vfprintf(FILE *f, const char *format, va_list arg);

int fprintf(FILE *f, const char *format, ...) {
  va_list l; int retval;
  va_start(l, format);
  retval = vfprintf(f, format, l);
  va_end(l);
  return retval;
}

int printf(const char *format, ...) // как-то вызвать vfprintf
```

Обсуждение

- Функции, такие как vfprintf и vsprintf очень полезны при написании собственных printf-подобных функций

```
pFile = fopen (szFileName,"r");
if (pFile == NULL)
  PrintFError("Error opening '%s'", szFileName);
```

- Понятно, что здесь PrintFError должна как-то вызвать внутри perror, но как её можно реализовать?

Обсуждение

- Функции, такие как vfprintf и vsprintf очень полезны при написании собственных printf-подобных функций

```
void PrintFError(const char * format, ...) {
  char buffer[256];
  va_list args;
  va_start(args, format);
  vsprintf(buffer,format, args);
  perror(buffer);
  va_end(args);
}
```

- Эта реализация не слишком совершенна (а что если буфер переполнится?), но вполне обычна для языка C