# VARIADIC

Минимум о функциях с переменным числом аргументов

К. Владимиров, Intel, 2018
mail-to: konstantin.vladimirov@gmail.com

# Опять полиномы

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

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

- Например произведение полиномов

```
struct Poly mult(struct Poly lhs, struct Poly rhs);
```

- Предположим, хочется протестировать произведение
- Как проще всего создать два полинома?

$A(x) = x^3 + 3x^2 + 4x + 7$

$B(x) = x^3 + 5x^2 + x + 4$

# Тоскливый способ создать полиномы

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

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

- Сложный способ это явно выделить память и записать туда значения
- $A(x) = x^3 + 3x^2 + 4x + 7$

```
struct Poly A = {4, NULL};
A.p = (int *) calloc(4, sizeof(int));
A.p[0] = 7;
A.p[1] = 4;
A.p[2] = 3;
A.p[3] = 1;
```

# Функция-конструктор из массива

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

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

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

```
struct Poly create_poly (unsigned n, int *data) {
  // TODO: выделить память и копировать данные
}

int polydata[] = {7, 4, 3, 1};
struct Poly A = create_poly(4, polydata);
```

- Напишите эту функцию!

# Вариабельная функция-конструктор

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

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

- И наконец можно написать вариабельную функцию-конструктор
- $A(x) = x^3 + 3x^2 + 4x + 7$

```
struct Poly create_poly (unsigned n, ...) {
  // TODO: какая-то магия
}

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

- Это выглядит волшебством. Давайте разберёмся как это работает

# Вариабельные функции

- Произвольное количество аргументов после троеточия

```
int sum_all (int n, ...) {
```

- 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`

# Пример: суммирование целых

- Собираем всё вместе: функция суммирует целые числа

```
int add_nums(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;
}
```

- Пример вызова

```
int n = add_nums(4, 10, 14, 24, 40); assert(n == 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

# Вариабельная функция-конструктор

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

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

- И наконец можно написать вариабельную функцию-конструктор
- $A(x) = x^3 + 3x^2 + 4x + 7$

```
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!