Лекция 29. Введение в современный язык Си. Часть 2.

Ревун Артем Леонидович
ст. преп. кафедры вычислительных систем

Курс «Программирование», весенний семестр, 2024
Сибирский государственный университет телекоммуникаций и информатики (г. Новосибирск)

Содержание

- Краткая история развития языка C
- Основные стандарты: от K&R до C23
- Изменения при смене стандарта языка
- Новые функции стандарта C23

Повторение (1)

Этапы развития:

K&R C – неформальный стандарт (1978)

ANSI C / ISO C (C89/C90) – первый официальный стандарт (1989)

C99 – существенные улучшения (1999)

C11 – модернизация и безопасность (2011)

C17 (C18) – исправления и уточнения (2017)

C23 (в разработке) – дальнейшие улучшения (2023)

Повторение (2)

Удалено из стандарта:

- обязательная поддержка чисел с дополнительным кодом
- определение функций в стиле K&R
- вызов *realloc()* с 0 теперь неопределённое поведение ~~(ха-ха)~~

Нововведения языка:

- директивы препроцессора #embed, #elifdef, #elifndef, #warning
- типы _Decimal32, _Decimal64 и _Decimal128
- атрибуты в стиле C++11:

Повторение (3)

Нововведения языка:

- [[nodiscard]], [[maybe_unused]], [[deprecated]], [[fallthrough]], [[noreturn]], [[reproducible]], [[unsequenced]]
- метки могут появляться до объявлений и в конце выражений
- неименованные параметры в объявлении функций
- бинарные литералы
- разделители цифр: 0xFF’FF’FF’FF
- оператор typeof()

Повторение (4)

Нововведения языка:

- пустая инициализация с помощью {} (включая VLA)
- ключевое слово auto для объявления переменных
- спецификатор constexpr
- ключевое слово static_assert вместо _Static_assert
- ключевое слово thread_local вместо _Thread_local
- ключевые слова alignas, alignof, bool, true, false

Повторение (5)

Изменения в стандартной библиотеке:

- новые заголовочные файлы <stdbit.h> и <stdckdint.h>
- некоторые POSIX функции становятся стандартными:
- memccpy(), strdup() (наконец-то) и strndup()
- gmtime_r(), localtime_r()
- расширения fscanf() и fprintf():
- спецификатор %b для вывода бинарных чисел
- H, D, DD для _Decimal32, _Decimal64 и _Decimal128

Повторение (5)

Изменения в стандартной библиотеке:

- новые заголовочные файлы <stdbit.h> и <stdckdint.h>
- некоторые POSIX функции становятся стандартными:
- memccpy(), strdup() (наконец-то) и strndup()
- gmtime_r(), localtime_r()
- расширения fscanf() и fprintf():
- спецификатор %b для вывода бинарных чисел
- H, D, DD для _Decimal32, _Decimal64 и _Decimal128

Нововведения стандарта C23: метки (1)

Метки могут появляться не только перед операторами case и default или перед инструкциями (statements), но и перед объявлениями (declarations) и в конце выражений.

Метка — это идентификатор, за которым следует двоеточие :. Она используется вместе с goto, чтобы перейти к определённой точке в коде.

```
label_name:
 printf("Мы здесь\n");
 goto label_name;
```

```
switch (value) {
   case 1:
      printf("Один\n");
     break;
   default:
     printf("Неизвестно\n");}
```

Нововведения стандарта C23: метки (2)

Раньше в Си можно было ставить метки только перед операторами , например:

```
label:
    x = 5;
```

Теперь доступно использование меток в стиле

```
label:                  или          label: ;
    int x;
```

Нововведения стандарта C23: метки (3)

1. Метки могут стоять перед объявлениями

```
void foo() {
    label:
    int x = 10; }
```

2. Метки могут стоять после точки с запятой (в "конце выражения")

```
void bar() {
    label: ;
    int y = 20; }
```

Нововведения стандарта C23: метки (4)

```c
void example() {
  int *arr1 = malloc(100); if (!arr1) goto error;
  int *arr2 = malloc(100); if (!arr2) goto error;
  free(arr2);free(arr1);
  return;
error:
  if (arr2) free(arr2); if (arr1) free(arr1);
  fprintf(stderr, "Ошибка выделения памяти\n");
}
```

Нововведения стандарта C23: метки (4)

```c
void example() {
  int *arr1 = malloc(100); if (!arr1) goto error;
  int *arr2 = malloc(100); if (!arr2) goto error;
  free(arr2);free(arr1);
  return;
error:
  if (arr2) free(arr2); if (arr1) free(arr1);
  fprintf(stderr, "Ошибка выделения памяти\n");
}
```

Нововведения стандарта C23: метки (1)

Метки могут появляться не только перед операторами case и default или перед инструкциями (statements), но и перед объявлениями (declarations) и в конце выражений.

Метка — это идентификатор, за которым следует двоеточие :. Она используется вместе с goto, чтобы перейти к определённой точке в коде.

```
label_name:
 printf("Мы здесь\n");
 goto label_name;
```

```
switch (value) {
   case 1:
      printf("Один\n");
     break;
   default:
     printf("Неизвестно\n");}
```

Нововведения стандарта C23: неименованные параметры (1)

Это параметры функции, которые указаны в её объявлении или определении, но не имеют имени. Они нужны, когда важно указать тип параметра, но его значение игнорируется внутри тела функции .

```
void log_message(int, const char*) {
…
}
```

Нововведения стандарта C23: неименованные параметры (2)

- Если вы объявили параметр, но не используете его, компилятор может выдать предупреждение (unused parameter). Неименованный параметр явно показывает, что это сделано намеренно.
- При реализации callback-функций, где сигнатура требует определённые параметры, но вы их не используете.
- Позволяет писать общие версии функций, где некоторые параметры могут использоваться только в некоторых конфигурациях.

Нововведения стандарта C23: неименованные параметры (3)

```c
void log_message_1(int unused, const char* message) {
    (void)unused;
    printf("Log: %s\n", message);
}
void log_message_2(const char* message) {
    printf("Log: %s\n", message);
}
```

Нововведения стандарта C23: неименованные параметры (4)

```c
void log_message(int level
[[maybe_unused]], const char* message){
  printf("Log: %s\n", message);
}
int main() {
  log_message(1, "Программа запущена");
  return 0;
}
```

Нововведения стандарта C23: бинарные литералы (1)

Это способ записи целых чисел в двоичной системе счисления прямо в коде.

```
int mask = 0b101010;
```

Бинарный литерал начинается с 0b или 0B.

Далее следуют цифры 0 и 1.

```
long value = 0B10101010101010101010;
int nibble = 0b0101_1100;
```

Нововведения стандарта C23: разделители цифр (1)

Это символ подчёркивания "_" , который можно использовать внутри числовых литералов , чтобы улучшить читаемость больших чисел
Разделитель игнорируется компилятором , то есть он не влияет на значение числа — это только для удобства разработчика.

Нововведения стандарта C23: разделители цифр (2)

```
int      a = 1_234_567;
long     b = 0xFFFF_CCCC;
int      c = 0b1010_0000_1111;
float    d = 3.141_592_653_589_793f;
```

Нововведения стандарта C23: разделители цифр (3)

```
int x = _123;          // ✗ Перед 1 цифрой
int y = 123_;          // ✗ После последней
int z = 0_x123;        // ✗ После префикса
float w = 123._456;// ✗ После точки
float v = 123_.456;// ✗ Перед точкой
```

Нововведения стандарта C23: typeof (1)

`typeof()` — это оператор времени компиляции , который возвращает тип выражения или имени типа.

```
typeof(константное_выражение)
typeof(имя_типа)
```

Это мощное и удобное расширение, которое позволяет получать тип переменной или выражения на этапе компиляции.

Нововведения стандарта C23: typeof (2)

```
int a = 10;
typeof(a) b = 20; // EQ int b = 20;
typeof(int *) ptr; // EQ int *ptr;
typeof(int*) p1; // EQ: int *p1;
typeof(1 + 2.0f) result = 5.0f;
                  // EQ float result = 5.0f;
```

Нововведения стандарта C23: пустая инициализация (1)

Добавлена возможность инициализировать переменные, массивы или структуры пустым списком инициализаторов:

```
int x = {};
int arr[10] = {};
```

Такой синтаксис означает, что все поля или элементы будут проинициализированы нулевыми значениями для соответствующих типов.

Нововведения стандарта C23: пустая инициализация (2)

```
int a = {};    // a == 0
double b = {}; // b == 0.0
char c = {};   // c == '\0’
int arr[5] = {}; // Все элементы равны 0
```

Нововведения стандарта C23: пустая инициализация (3)

```c
typedef struct {
    int x; float y; char str[20];
} Data;
Data d = {};

int n; scanf("%d", &n);
int arr[n] = {};
```

Нововведения стандарта C23: квалификатор auto (1)

Ключевое слово `auto` позволяет компилятору автоматически определять тип переменной на основе значения, которым она инициализируется.

```
auto variable_name = initializer;
```

Тип variable_name будет выведен как тип выражения initializer.

Нововведения стандарта C23: квалификатор auto (2)

```
auto a = 42;          // int
auto b = 3.14;        // double
auto c = 123.45f;     // float
auto d = 'A';         // char
int x = 10;
auto ptr = &x;      // int*
auto* ptr2 = &x;    // тоже int*
```

Нововведения стандарта C23: квалификатор auto (3)

```c
auto arr[] = {1, 2, 3}; // int[3]


typedef struct {
    int id; char name[32];
} User;
User user = {.id = 1, .name = "Alice" };
auto u = user;
```

Нововведения стандарта C23: спецификатор constexpr (1)

Это спецификатор типа хранения (storage-class specifier), который гарантирует, что значение переменной или результат вызова функции будет вычислен на этапе компиляции , если это возможно.

- Позволить использовать константные выражения в местах, где требуется постоянство и предсказуемость .
- Улучшить оптимизацию кода.
- Обеспечить типобезопасность и стабильность значений.

Нововведения стандарта C23: спецификатор constexpr (2)

```
constexpr type name =
                    константное_выражение;
constexpr type имя_функции(параметры) {
    // тело функции
}
constexpr int max_value = 100;
int arr[max_value];
```

Нововведения стандарта C23: спецификатор constexpr (3)

```
constexpr int square(int x) {
    return x * x; }

int main(void) {
    constexpr int s = square(5);
    int arr[s];
    return 0;}
```

Нововведения стандарта C23: спецификатор constexpr (4)

```
typedef struct {
    int width;
    int height;
} Rect;

constexpr Rect screen =
{ .width = 800, .height = 600 };
```

Нововведения стандарта C23: макросы версии (1)

```c
#if defined(__STDC_VERSION__) && __STDC_VERSION__ >= 202311L
/* C23 compatible source code. */
#elif defined(__STDC_VERSION__) && __STDC_VERSION__ >= 201710L
/* C17 compatible source code. */
#elif defined(__STDC_VERSION__) && __STDC_VERSION__ >= 201112L
/* C11 compatible source code. */
#elif defined(__STDC_VERSION__) && __STDC_VERSION__ >= 199901L
/* C99 compatible source code. */
#elif defined(__STDC_VERSION__) && __STDC_VERSION__ >= 199409L
/* C95 compatible source code. */
#elif defined(__STDC__) && __STDC__
/* C89 compatible source code. */
#else
/* K&R C compatible source code. */
#endif
```

Спасибо за внимание. Лекционный курс завершён.

Ревун Артем Леонидович
ст. преп. кафедры вычислительных систем

Курс «Программирование», весенний семестр, 2024
Сибирский государственный университет телекоммуникаций и информатики (г. Новосибирск)