

Приемы программирования на языке C

Содержание

1 Ресурсы по языку Си	1
2 Вводные замечания	1
3 Установка MSYS2 и MinGW-W64 для ОС Windows	2
4 Приемы работы в редакторе Eclipse	2
4.1 Настройка редактора Eclipse	2
4.2 Сборка и запуск проекта	3
4.3 Компиляция и запуск программы в редакторе Eclipse	5
5 Visual Studio Code как среда разработки для языка Си	5
6 Алфавит, идентификаторы, служебные слова	5
6.1 Константы и строки	5
6.2 Переменные и именованные константы	7
7 Структура программы	7
Список литературы	10
Список листингов	10

1. Ресурсы по языку Си

<https://learn.c.info/c/>

2. Вводные замечания

Язык Си – это компилируемый язык программирования высокого уровня, кроссплатформенный, позволяющий создавать программы которые будут работать во всех операционных системах, но для каждой операционной системы компиляцию нужно выполнять отдельно.

Существует несколько стандартов языка Си: C90 (ANSI C/ISO C), C99 и C11. Для того чтобы использовать правила конкретного стандарта, нужно в составе команды компиляции указать следующие флаги: `-std=c90`, `-std=c99` или `-std=c11`. Современный язык Си включает возможности стандарта C11.

Узнать используемый стандарт языка Си внутри программы можно с помощью *макроса* `__STDC_VERSION__`

```
printf("%ld\n", __STDC_VERSION__); // 201112
```

Получить информацию о версии компилятора позволяет макрос `__VERSION__`

```
printf("%s\n", __VERSION__); // Apple LLVM 12.0.0 (clang-1200.0.32.2)
```

Когда мы в командной строке вводим название программы без предварительного указания пути к ней, то

- вначале поиск программы выполняется в текущем рабочем каталоге (обычно это каталог, из которого запускается программа),
- а затем в путях, указанных в системной переменной `PATH`.

Системные каталоги имеют более высокий приоритет, чем каталоги, указанные в переменной `PATH`.

3. Установка MSYS2 и MinGW-W64 для ОС Windows

Установить компилятор `gcc` на ОС Windows можно следующим образом. Детали процедуры установки можно найти в книге [2, стр. 24]. Предварительно нам нужно установить библиотеку MSYS2. Переходим на сайт <https://www.msys2.org> и скачиваем файл `msys2-x86_64-20230718.exe`, а затем запускаем его.

Библиотека MSYS2 будет установлена в каталог `C:\msys64`. В этом каталоге расположены скрипты для запуска: `msys2.exe`, `mingw32.exe`, `mingw64.exe`.

Файл `msys2.exe` запускает командную строку, в которой мы можем установить различные библиотеки. Сначала обновим программу, выполнив команду

Окно `msys2.exe`

```
$ pacman -Syu
```

Теперь можно установить библиотеку MinGW-W64

```
$ pacman -S mingw-w64-x86_64-toolchain
```

Для установки всех компонентов нажимаем клавишу `<Enter>`, а затем на запрос подтверждения установки вводим букву `Y` и нажимаем клавишу `<Enter>`.

Библиотека MinGW-W64 будет установлена в каталог `C:\msys64\mingw64`. Добавив путь до `C:\msys64\mingw64\bin` в системную переменную `Path`, можно будет вызывать компилятор `gcc` из любой точки

```
$ gcc --version
g++.exe (Rev2, Built by MSYS2 project) 13.2.0
...
```

Все установленные библиотеки скомпилированы под 64-битные операционные системы. Для установки 32-битных версий библиотек нужно в команде заменить фрагмент `x86_64` фрагментом `i686`. Пример

```
$ pacman -S mingw-w64-i686-toolchain
```

4. Приемы работы в редакторе Eclipse


4.1. Настройка редактора Eclipse

Чтобы сделать иконки панели покрупнее, следует добавить в файл `eclipse.ini` следующие строки

```
-Dswt.enable.autoScale=true  
-Dswt.autoScale=150  
-Dswt.autoScale.method=nearest
```

Чтобы редактор поддерживал Vim, следует в меню **Help»Eclipse Markertplace** в строке Find вбить «Vrappre» и затем следовать инструкциям по установке.

4.2. Сборка и запуск проекта

Для того чтобы преобразовать текстовый файл `Test64c.c` с программой в исполняемый `exe`-файл, делаем текущей вкладку с содержимым файла `Test64c.c` и в меню **Project** выбираем пункт **Build Project**. В результате компиляции в рабочем каталоге будет создан каталог  **Debug**. Внутри этого каталога находится файл `Test64c.exe`, который можно запустить на выполнение с помощью двойного щелчка мыши на значке файла.

Для запуска делаем текущей вкладку с содержимым файла `Test64c.c` и в меню **Run** выбираем пункт **Run**. В открывшемся окне выбираем пункт **Local C/C++ Application** и нажимаем кнопку **OK**. Результат выполнения программы отобразится в окне **Console**.

Простейший пример программы

```
#include <stdio.h>  
  
int main(void) {  
    printf("Hello, world");  
  
    // в main() ключевое слово return можно не указывать  
    return 0;  
}
```

Здесь `#include` – это *директива препроцессора*, с помощью которой включается файл `stdio.h`, в котором есть функция `printf()`, предназначенная для форматированного вывода данных в окно консоли. Так как название файла указано внутри угловых скобок, его поиск будет выполнен в *путях поиска заголовочных файлов*.

Содержимое файла `stdio.h` на одной из стадий компиляции целиком вставляется вместо инструкции с директивой `#include`.

Функция `printf()` содержится внутри файла `stdio.h`, поэтому в первой строке программы мы включаем этот файл с помощью директивы `#include`. Если заголовочный файл не включить, то функция будет недоступна.

После всех *инструкций* указывается точка с запятой. Исключением являются [2, стр. 46]:

- *составные инструкции* (в нашем примере после закрывающей фигурной скобки блока функции `main()` точка с запятой не указывается)
- и *директивы препроцессоров* (в нашем примере нет точки с запятой после инструкции с директивой `#include`).

Согласно стандарту, внутри функции `main()` ключевое слово `return` можно не указывать. В этом случае компилятор должен самостоятельно вставить инструкцию, возвращающую значение 0 [2, стр. 46].

Программу можно скомпилировать и без редактора кода. Пример компиляции на ОС Windows

```
gcc -Wall -Wconversion -O3 -finput-charset=cp1251 -fexec-charset=cp1251 -o helloworld.exe  
helloworld.c
```

Первое слово (`gcc`) вызывает компилятор `gcc.exe`. Флаг `-Wall` указывает выводить все предупреждающие сообщения, возникающие во время компиляции программы, флаг `-Wconversion` задает вывод предупреждений при возможной потере данных, а флаг `-O3` определяет уровень оптимизации. С помощью флага `-finput-charset` указывается кодировка файла с программой, а с помощью флага `-fexec-charset` – кодировка C-строк. Название создаваемого в результате компиляции файла (`helloworld.exe`) задается после флага `-o`. Далее указывается название исходного текстового файла с программой на языке Си (`helloworld.c`).

Помимо файлов с исходным кодом (имеют расширение `*.c`) в проекте могут быть *заголовочные файлы* (имеют расширение `*.h`).

В заголовочных файлах указываются *прототипы функций* и *различные объявления*. Инструкции, начинающиеся с символа `#`, – это *директивы препроцессора* [2, стр. 47].

Например для заголовочного файла с именем `HelloWorld.h`

```
#ifndef HELLOWORLD_H_
#define HELLOWORLD_H_
#endif /* HELLOWORLD_H_ */
```

Здесь директива препроцессора `#ifndef` проверяет отсутствие константы с именем `HELLOWORLD_H_`, `#define` – создает константу с именем `HELLOWORLD_H_`, а `#endif` – обозначает конец блока проверки отсутствия константы.

Заголовочный файл мы подключаем к файлу с исходным кодом (`*.c`) с помощью директивы `#include`: `#include "HelloWorld.h"` (кавычки!!! а не угловые скобки). Встретив в исходном коде директиву `#include`, *компилятор* вставляет все содержимое заголовочного файла на место директивы. Если мы вставим две одинаковые директивы `#include`, то содержимое заголовочного файла будет вставлено дважды. Чтобы этого избежать прототипы функций и прочие объявления вкладываются в блок, ограниченный директивами `#ifndef` и `#endif`. В директиве `#ifndef` указывается константа, совпадающая с именем заголовочного файла. Все буквы в имени константы заглавные, а точка заменена символом подчеркивания. Если константа не существует (при первом включении так и будет), то с помощью директивы `#define` эта константа создается и содержимое блока вставляется в исходный код. При повторном включении заголовочного файла константа уже существует, поэтому содержимое блока будет проигнорировано. Таким образом, заголовочный файл вставлен не будет, а значит, и ошибки не возникает.

Вместо этих директив в самом начале заголовочного файла можно указать директиву препроцессора `#pragma` со значением `once`, которая также препятствует повторному включению файла (в старых компиляторах директива может не поддерживаться)

```
#pragma once
// Объявление функций и пр.
```

Название заголовочного файла в директиве `#include` может быть указано [2, стр. 51]:

- внутри угловых скобок `#include <stdio.h>`,
- внутри кавычек `#include "HelloWorld.h"`.

В первом случае заголовочный файл ищется в путях поиска заголовочных файлов. При этом *текущий рабочий каталог* не просматривается. Добавить каталог в пути поиска заголовочных файлов позволяет флаг `-I` в команде компиляции. Обычно с помощью *угловых скобок* включаются заголовочные файлы *стандартной библиотеки* или библиотеки *стороннего разработчика*.

Во втором случае мы имеем дело с заголовочным файлом, который *вначале* ищется в *текущем рабочем каталоге* (или относительно него), а затем в путях поиска заголовочных файлов,

как будто название указано внутри угловых скобок. Таким способом (`#include "HelloWorld.h"`) обычно включаются заголовочные файлы проекта.

Можно указать:

- о просто название заголовочного файла

```
#include "HelloWorld.h"
```

- о абсолютный путь к нему

```
#include "C:\\cpp\\projects\\HelloWorld\\src\\HelloWorld.h"
```

- о или относительный путь к нему

```
#include "../HelloWorld.h"
```

4.3. Компиляция и запуск программы в редакторе Eclipse

Компиляция в редакторе Eclipse выполняется в два прохода. При первом проходе создается объектный файл `HelloWorld.o`, а на втором проходе на его основе создается исполняемый файл.

По умолчанию для проекта задается режим компиляции *Debug*. В этом режиме дополнительно сохраняется информация для отладчика, и EXE-файл будет создан *без оптимизаций*. Когда программа уже написана и отлажена, нужно выбрать режим *Release*. Для этого в меню **Project** выбираем пункт `Build Configuration >> Set Active >> Release`.

В результате компиляции в разных режимах были созданы два EXE-файла – в подкаталоге *Debug* и в подкаталоге *Release*. Первый файл содержит отладочную информацию, а второй – нет. При компиляции второго была дополнительно выполнена оптимизация, поэтому именно этот файл нужно отдавать заказчику.

5. Visual Studio Code как среда разработки для языка Си

Скачать Visual Studio Code можно здесь <https://code.visualstudio.com/>. Для ОС Windows нужно еще установить GCC. На ОС Linux компилятор gcc доступен «из коробки». На ОС MacOS компилятор gcc можно установить с помощью утилиты **brew**.

После установки IDE останется только создать директорию с проектом под язык Си. Когда Visual Studio Code увидит файл с расширением `*.c`, она предложит установить специальное расширение «C/C++ Extension Pack v1.X.X».

6. Алфавит, идентификаторы, служебные слова

Идентификаторы, начинающиеся с одного символа подчеркивания «`_`» или с двух символов подчеркивания «`__`», зарезервированы для использования в библиотеках и компиляторах. Поэтому такие идентификаторы не рекомендуется выбирать в качестве имен в прикладной программе на языке Си. Рекомендуется при программировании имена констант записывать целиком заглавными буквами [1, стр. 15].

6.1. Константы и строки

По определению, константа представляет значение, которое не может быть изменено. Синтаксис языка определяет 5 типов *констант*:

1. символы,
2. константы перечисляемого типа,
3. вещественные числа,
4. целые числа,
5. нулевой указатель («null»-указатель).

Управляющие последовательности ('**\n**', '**\r**', etc.) являются частным случаем экскейп-последовательностей (ESC-последовательностей), к которым также относятся лексемы вида '**\ddd**', либо '**\xhh**'.

Символьная константа (символ) имеет *целый тип*, то есть символы можно использовать в качестве целочисленных операндов в выражениях.

Целочисленные именованные константы можно вводить с помощью перечисления `enum`. Пример

```
enum DAY {SUNDAY, MONDAY, ...};  
enum BOOLEAN {NO, YES};
```

В первой строке `DAY`, а во второй строке `BOOLEAN` это необязательный произвольный идентификатор – название перечисления.

Если в списке нет ни одного элемента со знаком '=', то значения констант начинаются с 0 и увеличиваются на 1 слева направо. Таким образом, `NO` равно 0, а `YES` – 1. Именованная константа со знаком '=' получает соответствующее значение, а следующая за ней именованные константы без явных значений увеличиваются на 1 каждая.

То есть если

```
enum BOOLEAN {NO=10, YES};  
printf("NO=%d, YES=%d", NO, YES); // NO=10, YES=11
```

В Python можно сделать так

```
from enum import Enum, auto  
  
class Boolean(Enum):  
    NO = 0  
    YES = auto()  
  
Boolean.NO.value # 0  
Boolean.YES.value # 1
```

Формально строки не относятся к константам языка Си, а представляют собой отдельный тип его лексем. Строковая константа определяется как последовательность символов, заключенных в двойные кавычки (не в апострофы).

Представление *строковых констант* в памяти ЭВМ подчиняются следующим правилам. Все символы строки размещаются подряд, и каждый символ (в том числе представленный экскейп-последовательностью) занимает ровно 1 байт. В конце записи строковой константы компилятор помещает символ '**\0**'.

Таким образом, количество байтов, выделяемое в памяти ЭВМ для представления значения строки, ровно на 1 больше, чем число символов в записи этой строковой константы.

При работе с символьной информацией нужно помнить, что длина символьной константы '**'F'**' равна 1 байту, а длина строки "**F**" равна 2 байтам.

6.2. Переменные и именованные константы

Одним из основных понятий языка Си является *объект* – именованная область памяти. Частный случай объекта – переменная.

Каждый из целочисленных типов (`char`, `short`, `int`, `long`) может быть определен либо как *знаковый* `signed` либо как *беззнаковый* `unsigned` (по умолчанию `signed`).

Различие между этими двумя типами – в правилах интерпретации *старшего бита внутреннего представления*. Спецификатор `signed` означает, что старший бит внутреннего представления воспринимался как знаковый; `unsigned` означает, что старший бит внутреннего представления входит в код представляемого числового значения, которое считается в этом случае беззнаковым. Выбор знакового или беззнакового представления определяет предельные значения, которые можно представить с помощью описанной переменной. Например на IBM PC переменная типа `unsigned int` позволяет представить числа от 0 до 65 535, а переменная типа `signed int` (или просто `int`) соответствуют значения в диапазоне от -32768 до +32767.

Именованные константы можно вводить с помощью *директивы препроцессора* `#define`, например

```
// препроцессорная константа
#define EULER 2.718282 // точка с запятой не нужна!!!
```

Что эквивалентно

```
const double EULER = 2.718282;
```

До начала компиляции текст программы на языке Си обрабатывается специальным компонентом транслятора – *препроцессором*. Далее текст от препроцессора поступает к компилятору. Итак, основное отличие констант, определяемых *препроцессорными директивами* `#define`, состоит в том, что эти *константы вводятся* в текст программы *до этапа компиляции* – препроцессор обрабатывает исходный код программы и делает в этом тексте замены и подстановки [1, стр. 29].

7. Структура программы

Программ состоит из инструкций, расположенных в текстовом файле

```
<Подключение заголовочных файлов>
<Объявление глобальных переменных>
<Объявление функций и пр.>
int main(void) {
    <Инструкции>
    return 0;
}
<Определения функций и пр.>
```

В самом начале программы подключаются *заголовочные файлы*, в которых содержатся *объявления идентификаторов без их реализации*.

После подключения файлов производится *объявление глобальных переменных*. Глобальные переменные видны во всей программе, включая функции. Если объявить переменную внутри функции, то *область видимости переменной* будет ограничена рамками функции и в других частях программы использовать переменную нельзя. Такие переменные называются *локальными*.

При объявлении переменной можно сразу присвоить начальное значение. Присваивание значения переменной при объявлении называется *инициализацией переменной*.


```
int x = 10;
int x = 21; int y = 85; int z = 56;
```

Если глобальной переменной не присвоено значение при объявлении, то она будет иметь значение 0. Если *локальной* переменной не присвоено значение, то переменная будет содержать **произвольное значение**. Как говорят в таком случае: переменная содержит «мусор» [2, стр. 59].

После директив препроцессора точка с запятой не указывается. В этом случае концом инструкции является конец строки. Директиву препроцессора можно узнать по символу # перед названием директивы.

После объявления глобальных переменных могут располагаться *объявления функций*. Такие объявления называются *прототипами*. Схема прототипа функции выглядит следующим образом

```
<Тип возвращаемого значения> <Название функции>(  
    [<Тип> [<Параметр 1>]  
    [, ..., <Тип> [<Параметр N>]]]);
```

Например, прототип функции, которая складывает два целых числа и возвращает их сумму, выглядит так

```
int sum(int x, int y);
```

После объявления функции необходимо описать ее реализацию, которая называется *определением функции*. Определение функции обычно располагается после определения функции `main()`. Обратите внимание на то, что объявлять прототип функции `main()` не нужно.

Пример определения функции `sum()`

```
int sum(int x, int y) {  
    return x + y;  
}
```

Первая строка в определении функции `sum()` совпадает с объявлением функции. Следует заметить, что в объявлении функции можно не указывать названия параметров. Достаточно будет указать информацию о типе данных. Таким образом, объявление функции можно записать так

```
int sum(int, int);
```

После объявления функции ставится точка с запятой. Если функция не возвращает никакого значения, то перед названием функции вместо типа данных указывается ключевое слово `void`. Пример объявления функции, которая не возвращает значения

```
void print(int); // объявление функции; прототип  
  
// определение функции, которая не возвращает значение  
void print(int x) {  
    printf("%d", x);  
}
```

Самой главной функцией в программе является функция `main()`. Именно функция с названием `main()` будет автоматически вызываться при запуске программы. Функция имеет три прототипа

```
int main(void);  
int main(int argc, char *argv[]);  
int main(int argc, char *argv[], char **penv);
```


Значение `void` внутри круглых скобок означает, что функция не принимает параметры. Второй прототип применяется для получения значений, указанных при запуске программы из командной строки. Количество значений доступно через параметр `argc`, а сами значения через параметр `argv`. Параметр `penv` в третьем прототипе позволяет получить значения переменных окружения.

Ключевое слово `int` означает, что функция возвращает целое число. Число 0 означает нормальное завершение программы. Если указано другое число, то это свидетельствует о некорректном завершении программы. Согласно стандарту, внутри функции `main()` ключевое слово `return` можно не указывать. В этом случае компилятор должен самостоятельно вставить инструкцию, возвращающую значение 0. Возвращаемое значение передается операционной системе и может использоваться для определения корректности завершения программы.

Вместо безликого значения 0 можно воспользоваться макроопределением `EXIT_SUCCESS`, а для индикации некорректного завершения программы – макроопределением `EXIT_FAILURE`. Предварительно необходимо включить заголовочный файл `stdlib.h`.

Пример программы

```
// Включение заголовочных файлов
#include <stdio.h>
#include <stdlib.h>

// Объявление глобальных переменных
int x = 21;
int y = 85;

// Объявление функций и пр.
int sum(int, int);
void print(int);

// Главная функция (точка входа в программу)
int main(void) {
    int z;
    z = sum(x, y);
    print(z);
    return EXIT_SUCCESS;
}

// Определение функций
int sum(int x, int y) {
    return x + y;
}

void print(int x) {
    printf("%d", x);
}
```

Объявление функций можно вынести в отдельный заголовочный файл и включить его с помощью директивы `#include`

MyPrototypes.h

```
#ifndef MYPROTOTYPES_H_
#define MYPROTOTYPES_H_

// Объявление функций и пр.
int sum(int x, int y);
```

```
#endif /* MYPROTOTYPES_H_ */
```

Директивы препроцессора `#ifndef`, `#define` и `#endif` препятствуют повторному включению заголовочного файла. Вместо этих директив можно указать в самом начале файла директиву препроцессора `#pragma` со значением `once`, то есть

MyPrototypes.h

```
#pragma once
```

```
// Объявление функций и пр.  
int sum(int x, int y);
```

Список литературы

1. Подбельский В.В., Фомин С.С. Программирование на языке Си, 2005. – 600 с.
2. Прохоренок Н.А. Язык С. Самое необходимое. – СПб.: БХВ-Петербург, 2020. – 480 с.

Листинги