Практическое занятие 6. Вход/выход из программы, мультифайловая сборка, форматированный вывод

Задание 1. Аргументы программы (аргументы функции main()).

Цель - изучить программный доступ к передаваемым программе аргументам.

ВАЖНО! "**argc**" и "**argv**" это традиционные и единственные аргументы функции "**main()**", определяемые стандартом ANSI C. Они позволяют передавать аргументы командной строки в программу. Параметр "**argc**" содержит число аргументов командной строки и является целым числом. Он всегда больше или равен 1, поскольку имя программы является первым аргументом. Параметр "**argv**" - это указатель на массив символьных указателей. Каждый элемент данного массива указывает на аргумент командной строки. Все аргументы командной строки - это строки.

1. В домашнем каталоге создаем и переходим в папку "**TASK_06_01**", в которой будем выполнять все операции задания:

```
$cd ~ <Enter>
$mkdir TASK_06_01 <Enter>
$cd TASK_06_01 <Enter>
$
```

2. Создаем файл "**args.c**" и добавляем в него следующие строки:

```
#include <stdio.h>

int main(int argc, char **argv) {
  int i;
  // Выводим на экран число аргументов, передаваемых программе
  printf("%d\n", argc);

  // В цикле выводим все аргументы, передаваемые программе
  for (i=0; i < argc; i++){
        printf("%s\n", argv[i]);
  }

  return 0;
}
```

3. Самостоятельно собрать программу. Далее запустить ее без аргументов, а потом с произвольным набором аргументов. Проанализировать вывод на экран.

Задание 2. Завершение программы.

Цель - изучить разные способы завершения Си программы.

ВАЖНО! Существует несколько способов завершения Си программы. Основные, это возврат "**return**" из функции "**main()**" и вызов функции "**exit()**". Оба приводят к завершению выполнения задачи и устанавливают код завершения программы.
Системный вызов "**exit()**" объявлен в системном заголовочном файле "**unistd.h**". В качестве аргумента функции "**exit()**" передается желаемый код завершения программы.

1. В домашнем каталоге создаем и переходим в папку "**TASK_06_02**", в которой будем выполнять все операции задания:

```
$cd ~ <Enter>
$mkdir TASK_06_02 <Enter>
$cd TASK_06_02 <Enter>
$
```

1. Разработаем программу, которая иллюстрирует завершение Си программы путем возврата из функции "**main**()". Создаем файл "**ret.c**" и добавляем в него следующие строки:

```
int main(int argc, char **argv) {

  // Выходим из программы с кодом завершения 0
  // (возвращаем 0 из функции main)
  return 0;
}
```

2. Самостоятельно скомпилировать и запустить программу. В командной строке проанализировать ее код завершения (напоминаю, что код завершения содержится в переменной окружения "**$?**").

3. Самостоятельно изменить в программе возвращаемое функцией "**main()**" значение на "**1**". Собрать/запустить измененную программу и проанализировать код ее завершения.

4. Самостоятельно изменить в программе возвращаемое функцией "**main()**" значение на "**123**". Далее собрать/запустить измененную программу и проанализировать код завершения.

5. Разработаем программу, иллюстрирующую завершение Си программы при помощи вызова "**exit**()". Создаем файл "**exit.c**" и добавляем в него следующие строки:

```
// Системный вызов exit() объявлен в системном
// заголовочном файле stdlib.h
#include <stdlib.h>

int main(int argc, char **argv) {
  // Завершаем программу при помощи вызова exit()
  // с кодом возврата 0
  exit(0);

  // В рамках этой программы этого return мы никогда не дойдем.
  return 0;
}
```

6. Самостоятельно скомпилировать и запустить программу. В командной строке проанализировать ее код завершения.

7. Самостоятельно изменить значение передаваемое функции "**exit()**" параметра на "**1**", собрать/запустить измененную программу и проанализировать код завершения.

8. Самостоятельно изменить значение передаваемое функции "**exit()**" параметра на "**123**", собрать/запустить измененную программу и проанализировать код завершения.

Задание 3. Сборка программ при помощи "make".

Цель - познакомиться с сборкой программ при помощи утилиты "**make**".

1. В домашнем каталоге создаем и переходим в папку "**TASK_06_03**", в которой будем выполнять все операции задания:

```
$cd ~ <Enter>
$mkdir TASK_06_03 <Enter>
$cd TASK_06_03 <Enter>
$
```

2. Создаем файл "**test.c**" и добавляем в него следующие строки:

```
#include <stdio.h>

int main(int argc, char **argv) {
  printf("HELLO\n");

  return 0;
}
```

3. Собираем программу в командной строке:

```
$gcc test.c -o test.x <Enter>
```

```
$
```

4. Запускаем программу и убеждаемся, что она работает.

5. Организуем сборку программы при помощи команды "**make**". Предварительно удаляем файл "**test.x**" из рабочего каталога. Далее создаем файл с именем "**Makefile**" и добавляем в него следующие строки:

```
# Первая цель (target) test.x, зависимая от test.o
test.x: test.o
    gcc test.o -o test.x

# Цель формирования объектника test.o из test.c
test.o: test.c
    gcc -c test.c -o test.o
                        # "-o test.o" - можно опустить, так как по
                        # умолчанию формируется именно файл test.o.
                        # Далее "-o test.o" использовать не будем.

# Дополнительная цель для удаления временных объектных файлов
# и самой скомпилированной программы
clean:
    rm -rf *.o test.x
```

6. Запускаем сборку:

```
$make <Enter>
gcc -c test.c -o test.o
gcc test.o -o test.x
$
```

7. Если все сделано правильно, то в рабочем каталоге появятся файлы **"test.o"** и **"test.x".** Первый - объектный файл, скомпилированный из "**test.c",** а второй - выполняемый файл программы. Запускаем **"test.x"** и убеждаемся, что программа работает.

8. Повторно запускаем сборку и анализируем результат.

9. Выполняем очистку рабочего каталога от временный файлов, используя цель "**clean**":

```
$make clean
```

   Убеждаемся, что в рабочем каталоге нет файлов с расширением "***.o**" и файла **"test.x"**.

10. Самостоятельно организовать сборку при помощи команды make для 1 и 2 заданий.

11. Самостоятельно добавить цель "**all"**, которая будет использоваться по умолчанию и которая сначала инициирует удаление временных файлов (цель "**clean**"), а потом сборку программы (цель "**test.x**").

Задание 4. Мультифайловая сборка.

Цель - организовать сборку программы, состоящей из нескольких файлов с исходными кодами.

1. В домашнем каталоге создаем и переходим в папку "**TASK_06_04**", в которой будем выполнять все операции задания:

```
$cd ~ <Enter>
$mkdir TASK_06_04 <Enter>
$cd TASK_06_04 <Enter>
$
```

2. Создаем заголовочный файл "**hello.h**", в котором декларируем функцию "**hello()**" для выводу на экран строки "**Hello world!**" (не забываем про защиту от повторного использования):

```
#ifndef _HELLO_H_
#define _HELLO_H_

void hello(void); // Декларируем функцию без аргументов и
                  // возвращаемого значения

#endif
```

3. Создаем файл "**hello.c**" с реализацией функции "**hello()**" (не забываем включить необходимые заголовочные файлы):

```
#include <stdio.h>
#include "hello.h"

void hello(void){
  printf("Hello world!\n");
}
```

4. Создаем файл "**main.c**", в котором вызываем функцию "**hello()**", реализованную в другом файле:

```
/* Включаем заголовочный файл с
   описанием функции hello() */
#include "hello.h"

int main(int argc, char **argv) {

   /* Вызываем функцию hello(), определенную в другом файле */
   hello();

   return 0;
}
```

5. Самостоятельно, по аналогии с заданием 3, создать "**Makefile**" для сборки программы. Скомпилировать и запустить программу. Убедиться в том, что программа корректно работает.

Задание 5. Библиотечные функции. Форматированный вывод.

Цель - познакомиться с форматированным выводом информации при помощи функции "**printf()**".

Описание функции **"printf()"**[1]

Функция вывода "**printf**()" предназначена для форматированного вывода информации в стандартный вывод "**stdout**", по умолчанию - на экран.

Параметрами "**printf**()" являются строка форматов и аргументы, которые необходимо вывести в соответствии с форматом:

```
int printf("СТРОКА_ФОРМАТОВ", АРГУМЕНТ_1, АРГУМЕНТ_2, ..., АРГУМЕНТ_N);
```

"**СТРОКА_ФОРМАТОВ**" состоит из:

управляющих символов;

текста, представленного для непосредственного вывода;

форматов, предназначенных для вывода значений переменных различных типов.

Управляющие символы не выводятся на экран, а только отвечают за расположение выводимых символов. Отличительной чертой управляющего символа является наличие перед ним обратного слэша '\'. Основные управляющие символы см. в приложении 1.

Форматы определяют вид, в котором информация будет выведена на экран. Формат начинается с символа процент '%'. Определяющим в форматах являются спецификаторы, список которых приведен в приложении 2. При необходимости, перед спецификатором формата можно указать общее количество знакомест и количество знакомест, занимаемых дробной частью.

Функция проходит по строке **"СТРОКА_ФОРМАТОВ"** и соответствующим образом заменяет первое вхождение "%**ФОРМАТ**" на первый аргумент "**АРГУМЕНТ_1**", второе вхождение "%**ФОРМАТ**" на второй аргумент " **АРГУМЕНТ_2**" и т.д.

1. В домашнем каталоге создаем и переходим в папку "**TASK_06_05**", в которой будем выполнять все операции задания:

```
$cd ~ <Enter>
$mkdir TASK_06_05 <Enter>
$cd TASK_06_05 <Enter>
$
```

2. Создаем файл "**prnt.c**" и добавляем в него следующие строки:

```
#include <stdio.h> // В файле stdio.h задекларирована функция printf()

int main(int argc, char **argv) {
  int a = 5; // Заводим локальную переменную типа int, значение
             // которой далее выведем на экран
```

[1] Более подробно см. встроенную справку (**$man 3 printf**)

```
  float f = 1.2345; // Заводим локальную переменную типа float,
                    // значение которой далее выведем на экран

  printf("a = %d\n", a); // Формируем СТРОКУ_ФОРМАТОВ так, чтобы
                         // сначала на экран было выведено "a = ",
                         // а потом значение переменной a

  printf("f = %f\n", f); // Формируем СТРОКУ_ФОРМАТОВ так, чтобы
                         // сначала на экран было выведено "f = ",
                         // а потом значение переменной f. В строке
                         // СТРОКУ_ФОРМАТОВ присутствует управляющий
                         // символ \n (перевод строки)

  printf("f = %10.5\n", f); // Формируем СТРОКУ_ФОРМАТОВ так, чтобы
                            // сначала на экран было выведено "f = ",
                            // а потом значение переменной f, причем
                            // общее число знакомест, выделенное для
                            // переменной, равнялось 10, а количество
                            // позиций после десятичной точки
                            // равнялось 5

  printf("a = %d,\t f = %10.5f\n", a, f); // Одновременно выведем на
                                          // экран значения переменных a и f

  return 0;
}
```

3. Самостоятельно вывести на экран число "**-1**" в разных форматах (двоичном, восьмеричном, шестнадцатеричном).

Приложение 1. Основные управляющие символы

'\n'	перевод строки
'\t'	горизонтальная табуляция
'\v'	вертикальная табуляция
'\b'	возврат на символ
'\r'	возврат на начало строки
'\a'	звуковой сигнал

Приложение 2. Основные Форматы

%d	целое число типа "**int**" со знаком в десятичной системе счисления
%u	целое число типа "**unsigned int**"
%x	целое число типа "**int**" со знаком в шестнадцатеричной системе счисления
%o	целое число типа "**int**" со знаком в восьмеричной системе счисления
%hd	целое число типа "**short**" со знаком в десятичной системе счисления
%hu	целое число типа "**unsigned short**"
%hx	целое число типа "**short**" со знаком в шестнадцатеричной системе счисления
%ld	целое число типа"**long int**" со знаком в десятичной системе счисления
%lu	целое число типа "**unsigned long int**"
%lx	целое число типа "**long int**" со знаком в шестнадцатеричной системе счисления
%f	вещественный формат (числа с плавающей точкой типа "**float**")

%lf	вещественный формат двойной точности (числа с плавающей точкой типа "**double**")
%e	вещественный формат в экспоненциальной форме (числа с плавающей точкой типа "**float**" в экспоненциальной форме)
%c	символьный формат
%s	строковый формат