Лабораторное занятие №4
«Взаимодействие со средой исполнения»

Аргументы командной строки

Задание №1

Написать функцию, предназначенную для решения квадратного уравнения $a \cdot x^2 + b \cdot x + c = 0$ для произвольных значений коэффициентов a, b и c. Значения коэффициентов передать в функцию main с помощью параметров командной строки.

Замечания: для того, чтобы получить параметры, переданные при вызове программе, следует воспользоваться вторым вариантом описания функции main:

int main(int argc, char * argv[]) или int main(int argc, char ** argv)

Для преобразования строкового представления числа к числовому значения можно воспользоваться функциями (заголовочные файлы string.h, stdlib.h):
int atoi(const char * string); — преобразует строку string в целое значение типа int.
long int atol(const char * string); — преобразовывает строку string в длинное целое.
double atof(const char * string); — преобразует строку в значение типа double.

Задание №2

Написать программу, которой могут быть переданы три «*короткие опции*»: -h (справка по использованию программы и завершение работы), -o out_file_name (задание нестандартного имени выходного файла), -c (особый режим работы). Кроме того, программе могут быть переданы дополнительные аргументы, задающие имена входных файлов. Вывести сообщение, указывающее режим работы программы.

Замечания: для того, чтобы выполнить разбор коротких опций командной строки можно воспользоваться специальной функцией getopt (заголовочные файлы getopt.h, unistd.h) (http://www.gnu.org/software/libc/manual/html_node/Getopt.html#Getopt):

```
int getopt(int argc, char *argv[], const char *optstring);

extern char *optarg;
extern int optind, opterr, optopt;
```

Аргументы argc и argv передаются непосредственно от функции main(), а optstring является строкой символов опций. Если за какой-либо буквой в строке следует двоеточие, значит эта опция принимает аргумент.

Для использования getopt() ее вызывают повторно в цикле. При этом функция возвращает код очередной фактически переданной опции. Когда все указанные опции обработаны, функция возвращает значение -1. Каждый раз обнаружив действительный символ опции, функция возвращает этот символ. Если пользователь ввел опцию, не указанную в строке opts, то getopt() возвращает код символа "?" (*знак вопроса*).

Для чтения аргументов опций и независимых аргументов предназначены следующие внешние переменные, также объявленные в указанных заголовочных файлах:
extern char * optarg; extern int optind;

Строка optarg содержит зависимый аргумент обрабатываемой опции, а в переменной optind хранится индекс первого свободного аргумента в массиве argv. При этом следует

учитывать, что getopt() перемещает все свободные аргументы в конец argv. Таким образом, они оказываются в диапазоне между optind и argc-1. Если optind больше или равно argc, то это означает, что свободные аргументы отсутствуют.

Пример:

```
#include <unistd.h>
#include <stdlib.h>
#include <stdio.h>

int main(int argc, char **argv) {
   if(argc == 1) { // если запускаем без аргументов, выводим справку
      printf("getopt test\n");
      printf("usage:\n");
      printf(" opts -a n -b m -o s\n");
      printf("example:\n");
      printf(" $ opts -a 323 -b 23 -o '-'\n");
      printf(" 323 — 23 = 300\n");
      return 0;
   }
   char *opts = "a:b:o:"; // доступные опции, каждая принимает аргумент
   int a, b; // тут храним числа
   char op; // а тут оператор
   int opt; // каждая следующая опция попадает сюда
   while((opt = getopt(argc, argv, opts)) != -1) {
                                 // вызываем getopt пока она не вернет -1
      switch(opt) {
         case 'a': // если опция -a, преобразуем строку с аргументом в число
            a = atoi(optarg);
            break;
          case 'b': // тоже для -b
             b = atoi(optarg);
             break;
          case 'o': // в op сохраняем оператор
             op = optarg[0];
             break;
      }
   }
   switch(op) {
      case '+': // если опаратор + складываем, и т.д.
         printf("%d + %d = %d\n", a, b, a + b);
         break;
      case '-':
         printf("%d — %d = %d\n", a, b, a — b);
         break;
      case '*':
         printf("%d * %d = %d\n", a, b, a * b);
         break;
      case '/':
         printf("%d / %d = %d\n", a, b, a / b);
         break;
   }
```

```
    return 0;
}
```

Задание №3

Написать программу, которой могут быть переданы указанные в предыдущем задании «*короткие опции*» и соответствующие им «*длинные опции*»: (--help, --output *out_file_name* и --compile).

Замечания: для того, чтобы выполнить разбор длинных опций командной строки можно воспользоваться специальной функцией getopt_long, которая может обрабатывать как длинные, так и короткие опции (заголовочные файлы getopt.h, unistd.h) (http://www.gnu.org/software/libc/manual/html_node/Getopt-Long-Options.html#Getopt-Long-Options):

```
int getopt_long(int argc, char *argv[], const char *optstring,
                const struct option *longopts, int *longindex);
```

Первые три аргумента этой функции такие же, как и у функции getopt(). Аргумент longopts является массивом указателем на массив структур option, описывающий длинные опции и соответствующие им короткие опции, а аргумент longindex указывает на переменную, в которую помещается индекс обнаруженной длинной опции в longopts; если в этом нет необходимости, то в качестве данного параметра может быть NULL.

Структура option содержит четыре поля:

```
struct option { const char *name; int has_arg; int *flag; int val; }
```

name — имя опции без предшествующих черточек;
has_arg — признак того, имеет ли длинная опция аргумент;
может принимать три значения:
    0 — не принимает аргумент;
    1 — обязательный аргумент;
    2 — необязательный аргумент.
flag — если этот указатель равен NULL, то getopt_long() возвращает значение поля val, иначе она возвращает 0, а переменная на которую указывает flag заполняется значением val;
val — обычно содержит некоторую символьную константу, если длинная опция соответствует короткой, то эта константа должна быть такой же как и та что появляется в аргументе optstring.

Следует отметить, что последний элемент массива longopts, должен содержать только нулевые значения:

```
name = NULL;    has_arg = 0;    flag = NULL;    val = 0;
```

Пример:

```
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <getopt.h>

void usage(char *name)
{
printf("usage: %s\n \     \t-h this message\n \     \t-c [config file]\n \     \t--help this
message\n \     \t--config=config_file\n", name);
```

```
return;
}

int main (int argc, char *argv[])
{
int c;
 const struct option long_opt[] = {
                 {"help", 0, 0, 'h'},
                 {"config", 1, 0, 'c'},
                 {NULL,0,NULL,0}
               };
while ((c = getopt_long(argc, argv, "c:h", long_opt, NULL)) != -1){
 switch( c ){
    case 'h':
       usage(argv[0]);
       return(-1);

    case 'c':
       printf("option 'c' selected, filename: %s\n", optarg);
       return(1);
    case '?':
       fprintf(stderr, "%s\n", "Unknown option");
       return 2;

    default:
       usage(argv[0]);
       return(-1);
  }
}

return(0);
}
```

Переменные окружения

Задание №4

Написать программу, которая выводит полный список переменных окружения.

Замечание: для того, чтобы в программе получить доступ ко всему списку переменных окружения вместе с их значениями можно воспользоваться переменной environ, которая в программе объявляется, как:

```
#include <stdlib.h> /* или unistd.h */

extern char **environ;
```

Данная переменная указывает на массив строк, называемый *environment* (окружение). После последней переменной окружения в данном массиве следует NULL.

Задание №5

Написать программу, которая выводит в стандартный поток вывода значение

некоторой конкретной переменной окружения, указанной при вызове программы. Если имя переменной окружения не указано при вызове программы, то вывести информацию об использовании программы.

Замечание: Окружение состоит из строк вида *имя=значение*. Для получения значения конкретной переменной окружения предназначена функция getenv().

```
#include <stdlib.h>

char *getenv(const char *name);
```

Функция getenv ищет в окружении строку с заданным именем и возвращает значение, ассоциированное с этим именем. Она вернет NULL, если требуемая переменная не существует. Если переменная есть, но ее значение не задано, функция getenv завершится успешно и вернет пустую строку, в которой первый байт равен null. Строка, возвращаемая getenv, хранится в статической памяти, принадлежащей функции, поэтому для ее дальнейшего использования вы должны скопировать эту строку в другую, поскольку она может быть перезаписана при последующих вызовах функции getenv.

Задание №6

Написать программу, которая присваивает заданной переменной окружения указанное значение, затем проверяет только что установленную переменную и выводит полученную пару *переменная — значение* в стандартный поток вывода. Если при запуске программы необходимая информация не указана, то вывести рекомендацию по использованию программы.

Замечание: Для установки значения переменной окружения для работающей программы можно воспользоваться функциями:

```
#include <stdlib.h>

int setenv(const char *name, const char *value, int overwrite);
```

Функция setenv() добавляет к окружению переменную name со значением value, если переменной с этим именем не существует. Если же переменная окружения name уже существует, то ее значение изменяется на value в случае, если флаг overwrite не равен нулю; если же overwrite равен нулю, то значение переменной name не изменится. Функция setenv() возвращает 0 при успешном завершении и -1, если произошла ошибка (в памяти среды было недостаточно места).

```
#include<stdlib.h>

int putenv(const char *init_str);
```

Функция putenv() добавляет новую или заменяет существующую переменную окружения, используя строку инициализации init_str в формате *переменная=значение*. Функция putenv() возвращает 0 при успешном завершении и -1, если произошла ошибка. Следует отметить, что функция putenv() реализована не во всех *Unix*-подобных системах, что может сказаться на переносимости программ.

Написать две версии программы, чтобы протестировать работу этих двух функций.

Для формирование строки инициализации можно применить строковые функции strcpy() и strcat():

```
#include <string.h>

char * strcpy( char * destptr, const char * srcptr );
```

Функция копирует *Cи*-строку srcptr, включая завершающий нулевой символ в строку назначения, на которую ссылается указатель destptr. Чтобы избежать переполнения, строка, на которую указывает destptr должна быть достаточно длинной, чтобы в неё поместилась копируемая строка (включая завершающий нулевой символ). Копируемая строка и строка назначения не должны перекрываться в памяти. Функция возвращает указатель на строку назначения.

```
char * strcat( char * destptr, const char * srcptr );
```

Объединение строк. Функция добавляет копию строки srcptr в конец строки destptr. Нулевой символ конца строки destptr заменяется первым символом строки srcptr, и новый нуль-символ добавляется в конец уже новой строки, сформированной объединением символов двух строк в строке destptr. Функция возвращает указатель на строку назначения.

Задание №7

Напишите программу, которая удаляет указанную переменную из окружения. Если имя переменной, которую нужно удалить, не указано при запуске программы, то удалить все окружение. После этого вывести текущее окружение в стандартный поток вывода.

Замечание: для удаления переменной из окружения можно использовать функцию unsetenv():

```
int unsetenv (const char * NAME);
```

Функция unsetenv() удаляет переменную с именем NAME из окружения. Ранее функция unsetenv() ничего не возвращала, а в настоящее время unsetenv() возвращает 0 при успешном завершении или -1 в случае ошибки. Передача в функцию unsetenv() имени несуществующей переменной не считается ошибкой.

Для очистки всего окружения можно воспользоваться функцией clearenv().

```
#include <stdlib.h>

int clearenv(void);
```

Следует отметить, что функция clearenv() реализована не во всех *Unix*-подобных системах.

Получение информации

Информация о пользователе

Задание №8

Написать программу, получающую информацию о пользователе, который ее запустил.

Замечания: Когда пользователь регистрируется в системе *Linux* он указывает имя пользователя и пароль. После того как эти данные проверены, пользователю предоставляется командная оболочка. В системе у пользователя также есть уникальный идентификатор пользователя, называемый *UID* (*user identifier*). Каждая программа, выполняемая *Linux*, запускается от имени пользователя и имеет связанный с ней *UID*.

У *UID* есть свой тип uid_t, определенный в файле sys/types.h. Обычно это короткое целое (small integer). Одни идентификаторы пользователя заранее определены системой, другие создаются системным администратором, когда новые пользователи становятся известны системе. Как правило, идентификаторы пользователей имеют значения, большие 100.

```
#include <sys/types.h>
#include <unistd.h>

uid_t getuid(void);
char *getlogin(void);
```

Функция getuid возвращает *UID*, с которым связана программа. Обычно это *UID* пользователя, запустившего программу.

Функция getlogin возвращает регистрационное имя, ассоциированное с текущим пользователем.

Если программа определила *UID* пользователя, запустившего ее, то можно заглянуть в файл **passwd** для выяснения, например, полного имени пользователя. Для этого определены функции, позволяющие получить дополнительную пользовательскую информацию.

```
#include <sys/types.h>
#include <pwd.h>

struct passwd *getpwuid(uid_t uid);
struct passwd *getpwnam (const char *name);
```

Структура базы данных учетных записей пользователей passwd определена в файле pwd.h и включает элементы, перечисленные в табл:

Элемент passwd	***Описание***
char *pw_name	Регистрационное имя пользователя
uid_t pw_uid	Номер *UID*
gid_t pw_gid	Номер *GID*
char *pw_dir	Исходный каталог пользователя
char *pw_gecos	Полное имя пользователя
char *pw_shell	Командная оболочка пользователя, запускаемая по умолчанию

В некоторых системах *UNIX* может использоваться другое имя для поля с полным именем пользователя: в одних системах это pw_gecos, как в *ОС Linux*, в других — pw_comment. Обе функции (и getpwuid, и getpwnam) возвращают указатель на структуру passwd, соответствующую пользователю. Пользователь идентифицируется по *UID* в функции getpwuid и по регистрационному имени в функции getpwnam. В случае ошибки обе функции вернут пустой указатель и установят переменную errno.

Попробовать получить информацию о пользователе с помощью его *UID* и узнать доступную информацию о пользователе root по его регистрационному имени.

Информация о компьютере

Задание №9

Написать программу, получающую информацию о компьютере, на котором выполняется.

Замечания: Если в системе установлены сетевые компоненты, то сетевое имя компьютера можно получить с помощью функции gethostname:

```
#include <unistd.h>

int gethostname(char *name, size_t namelen);
```

Эта функция записывает сетевое имя машины в строку name. Предполагается, что длина строки, как минимум, namelen символов. Функция gethostname возвращает 0 в случае успешного завершения и -1 в противном случае.

Более подробную информацию о рабочем компьютере можно получить с помощью функции uname:

```
#include <sys/utsname.h>

int uname(struct utsname *name);
```

Функция uname записывает информацию о компьютере в структуру, на которую указывает параметр name. Структура типа utsname, определенная в файле sys/utsname.h, обязательно должна включать элементы, перечисленные в табл.

Элемент структуры utsname	***Описание***
char sysname[]	Имя операционной системы
char nodename[]	Имя компьютера
char release[]	Номер выпуска (релиза) системы
char version[]	Номер версии системы
char machine[]	Аппаратный тип

В случае успешного завершения функция uname возвращает неотрицательное целое и -1 в противном случае с установленной переменной errno для обозначения любой возникшей ошибки.

Уникальный идентификатор каждого рабочего компьютера можно получить с помощью функции gethostid.

```
#include<unistd.h>

long gethostid(void);
```

Данная функция предназначена для возврата уникального значения, характеризующего рабочий компьютер.

Домашнее задание №2
«Взаимодействие со средой исполнения»

Написать программу environ, которая принимает короткие и длинные опции и выполняет следующие задачи:

- если программа вызвана без опций, то в стандартный поток вывода выводится информация о текущем окружении;
- если указана короткая опция -h (или длинная --help), то в стандартный поток вывода будет выведена информация по работе с программой;
- если указана короткая опция -i <переменная> (или длинная --info <переменная>), то в стандартный поток вывода будет выведено значение указанной переменной или сообщение о том, что данной переменной в окружении нет;
- если указана короткая опция -s <переменная=значение> (или длинная --set <переменная=значение>), то указанная переменная окружения получит новое значение и в стандартный поток вывода будет выведено установленное значение указанной переменной;
- если указана короткая опция -a <переменная> и -v <значение> (или длинная --assign <переменная> и --value <значение>), то указанная переменная окружения получит новое значение и в стандартный поток вывода будет выведено установленное значение указанной переменной; если значение не указано, то присваивается пустая строка; если переменная не указана, то присваивание не производится и в стандартный поток ошибок выводится соответствующее сообщение;
- если указана короткая опция -d <переменная> (или длинная --del <переменная>), то указанная переменная удаляется из окружения;
- если указана короткая опция -c (или длинная --clear), то программа полностью очищает окружение.

Каждое действие должно быть оформлено в виде функции; функции, выполняющие действия, должны быть собраны в отдельном исходном файле. Подготовить Makefile для сборки программы.