1. Даны описания:

```
unsigned u; signed char sc; int i; float f; unsigned short us;
long double ld; double d; short s;
```

Определить тип выражений:

(a) `u - us *i` `unsigned` (полное обозначение: `unsigned int`).

Смотрим раздел 10.2.3 (об арифметических преобразованиях) в задачнике Руденко, стр. 67. В соответствии с пунктом 1 (повышение типа для коротких целых) тип подвыражения `us` повышается до `int`, если значение `us` «умещается» в `int`, или до `unsigned int`, если «не умещается». В соответствии с пунктом 2 (балансировка типов операндов) значение подвыражения `us*i` будет иметь тип `int`, если тип `us` «повысился» до `int`, или `unsigned int`, если тип `us` «повысился» до `unsigned int`. Значение всего выражения будет иметь, в связи с балансировкой, тип `unsigned int`, т.к. левый операнд – `u` – имеет тип `unsigned int`, а правое подвыражение – `us*i` – не выше, чем `unsigned int`.

(b) `(sc+d) *ld` `longdouble`
Операнд `ld` имеет «максимальный» тип в данном выражении – `long double`

(c) `( f + 3 ) / ( 2.5f - s *3.14 )` `double`

Константа `3.14` имеет тип `double`, и это «максимальный» тип в данном выражении. Константа `2.5f` имеет тип `float` из-за суффикса f . При отсутствии суффикса любая вещественная константа имеет тип `double`.

(d) `sc -'0'+ '1'` `int`
Операнд `sc` повышается до `int` (кроме того, литерные константы `'0'` и `'1'` тоже имеют тип `int`)
(e) `(1-2)? 5: d` `double`

В тернарной операции `?:` тип результата балансируется по 2-му и 3-му операндам.

2. Верно ли записаны выражения? Для верно записанных выражений вычислить их значения. Считать, что тип `int` имеет размер два байта:

```
int a, b, c, d, e;
a = 2; b = 13; c = 7; d = 19; e = -4;
```

(a) `(b / a) / c ==0` (b) `(d / a) % c ==2` (c) `(c % d)-e ==11`

(d) `((-e) % a) + (b / a) *(-5)+5 ==-25`

(e) `sizeof (int)(unsigned int)(-1)` – ошибка. Выражение трактуется как `(sizeof (int))(unsigned int)(-1)`, т.к. `sizeof (int)` — первичное выражение. Таким образом, либо не хватает знака операции между двумя подвыражениями

`(sizeof (int))` и `(unsigned int)(-1)` либо не хватает скобок:

`sizeof ( (int) (unsigned int)(-1) )`

3. Что напечатает следующая программа?

```
#include <stdio.h>
char str[ ] = "SSSWILTECH1\1\11W\1WALLMP1";
main()
  { int i, c;
    for ( i = 2; ( c = str [ i ] ) != '\0'; i++) {
       switch (c) {
         case 'a': putchar('i'); continue;
         case '1': break;
         case 1: while((c =str[++ i ] ) != '\1' && c!= '\0');
         case 9: putchar('S');
         case 'E': case 'L': continue;
         default: putchar(c); continue; }
       putchar(' '); }
    putchar('\n');

}
```

ОТВЕТ :

SWITCH_SWAMP_<переход на нов. строку>

Пояснения: \1 – это символ с кодом 1, \11 – это символ с кодом 9.

Когда срабатывает `case '1': break;` выходим из `switch`, но до перехода на следующую итерацию цикла напечатается пробел `putchar(' ');`

4. Опишите функцию `transp` (типа `void`), которая транспонирует матрицу целых чисел (тип `int`) размера `N×N`, где `N` — заданная с помощью директивы `define` константа.

Решение

Матрица `int a[N][N]` в языке Си — это массив из `N` элементов, каждый из которых является массивом из `N` целых чисел. Массивы не могут передаваться как параметр функции и не могут быть результатом функции. Кроме того, для массивов не определена операция присваивания, и поэтому невозможно описать функцию `transp` так, чтобы верным было выражение `a=transp(a)`. Можно описать `transp` как процедуру, то есть функцию с типом результата `void`. Тогда ее можно использовать в роли оператора `transp(a);`, который изменяет матрицу `a`. Переменная `a` имеет тип «массив», и в качестве фактического параметра, указанного в вызове функции, `a` является простым выражением. По правилам языка Си значением выражения типа «массив» будет указатель на начальный элемент этого массива.[1] Начальным элементом матрицы `a` является массив из `N` целых чисел (начальная строка матрицы). То есть функция `transp` получит указатель на начальную строку матрицы. Количество строк матрицы в данной задаче не обязательно передавать в функцию `transp` в виде отдельного параметра[2], так как матрица квадратная, а размер `N` фиксирован в программе директивой `define`.

Прототип[3], или заголовок, функции выглядит так:

```
void  transp (int m[N][N]);
```

или так:

`void transp (int m[][N]);` В обоих случаях формальный параметр `m` рассматриваются компилятором как указатель на массив из `N` элементов типа `int`:

```
void transp(int (*m)[N] );
```

Заметим, что опускать вторую размерность нельзя, т.к. иначе компилятор не сможет правильно вычислить адрес элемента `m[i][j]`. Действительно, по правилам Си это выражение заменяется на `*(*(m+i)+j)`, причем по законам адресной арифметики, `m+i` — это адрес i-го элемента массива `m`, т.е. адрес i-ой строки матрицы. Если размер строки неизвестен, то невозможно вычислить адрес `m+i`.

Итак, реализуем транспонирование, совершая обмен значений пар элементов `m[i][j]` и `m[j][i]`. Заметим, что проходить нужно не по всей матрице, а только по треугольнику над (или под) главной диагональю, т.е. индексы изменяются так: для каждого `i` из диапазона `0≤i<N` `j` пробегает диапазон `i<j<N`.

В противном случае, если обходить все элементы матрицы, обмен значений для каждой пары `m[i][j]` и `m[j][i]` будет совершен дважды, и в итоге матрица не изменится.

```
void transp (int m[][N]){
/* транспонирование матрицы m размера N×N */

int i,j, temp;
for (i=0; i<N; i++ )
   for (j=i+1; j<N; j++){
       temp = m[i][j];
       m[i][j]= m[j][i];
       m[j][i] = temp;
   }
return;
}
```

---

[1] Преобразование к указателю не происходит только в трех случаях: когда выражение типа «массив» является операндом операции `sizeof` или адресной операции `&`, а так же при инициализации символьного массива строковой константой. В последнем случае значением строковой константы будет массив символов с завершающим '\0', а не указатель на начальный символ строки.

[2] Обычно размер массива указывается в виде второго параметра, поскольку по первому параметру — указателю на начальный элемент массива — компилятор не может определить количество элементов. Даже если тип формального параметра задан как тип-массив с указанием количества элементов в нем , он все равно преобразуется к указателю и информация о размере теряется.

[3] В прототипе указываются имя функции, тип возвращаемого значения, параметры и их типы. Если тело функции отсутствует (это значит, что определение этой функции находится в другом месте), то имена параметров в прототипе можно опускать, оставляя только имена типов.

5. Напишите программу *Prog* со следующим поведением. При запуске этой программы из командной строки, она рассматривает переданные ей аргументы как слова предложения и печатает «Да», если предложение является палиндромом; иначе печатает «Нет».

Предложение является палиндромом, если последовательность составляющих его букв без учета регистра, пробельных символов и знаков препинаний читается одинаково слева направо и справа налево. Для упрощения задачи полагаем, что знаков препинания в рассматриваемом программой *Prog* предложении нет.

Например,
$> *Prog А роза упала на лапу Азора*
Да

Решение

Один из вариантов решения — переписать всё предложение, распределенное по элементам `argv[i]`, в одну строку, приводя все буквы к нижнему регистру, и проверить, является ли получившаяся строка палиндромом, т.е. читается ли она одинаково слева направо и справа налево.

Опишем для этого функцию `is_palindrom(s)`, возвращающую `1` или `0` в зависимости от того, является строка `s` палиндромом или нет. Комментарии располагаются перед фрагментами, к которым они относятся.

```
int is_palindrom(char *s){
char *p = s;
/* пустая строка является палиндромом */ if (*s ==’\0’) return
1;

/* для непустой строки устанавливаем указатель p на последнюю
букву в строке: для этого повторяем продвижение p на одну
позицию вправо с помощью p++; если *(p+1) равно ’\0’, то p
указывает на последнюю букву, поскольку p+1 указывает на
завершающий строку символ ’\0’. Явное сравнение с нулем можно
опустить, т.к. условия *(p+1)!=’\0’ и *(p+1) эквивалентны */

while (*(p+1)) p++;
/* продвигая указатели s и p навстречу друг другу до тех пор,
пока они не встретятся, сравниваем первую букву с последней,
вторую с предпоследней и т.д.; если обнаружим несовпадение
сравниваемых (симметрично расположенных) букв, то прекращаем
дальнейшее продвижение указателей */
for ( ; s<p && *s==*p; s++,p--);

/* если s<p, то цикл for завершился из-за несовпадения букв и
тогда ответ 0, иначе -- 1. Выражение !(s<p) или s>=p дает нужный
ответ */
return s>=p;
}
```

При переписи слов предложения в одну строку нам понадобится функция, переводящая большие буквы русского и латинского алфавита в нижний регистр. Опишем такую функцию. Стандарт языка Си не дает никаких гарантий относительно взаимного расположения и упорядоченности букв по алфавиту. Все зависит от используемой кодировки символов. Если используется кодировка ASCII, то перевести символ `c` со значением из диапазона `'A'..'Z'` в нижний регистр можно по формуле `c+('a'- 'A')`. Для русских букв такой формулы может не быть, поэтому мы реализуем версию, которая вообще не зависит от кодировки. Для этого опишем массивы с заглавными `high` и строчными `low` буквами. Будем искать букву в массиве заглавных букв `high`, и выдавать соответствующую строчную букву из `low`. Если буква в массиве `high` не найдена, оставляем ее без изменения.

```
int tolower(int c){
/* переводит  буквы из диапазонов'A'..'Z' и 'A'..'Я' в нижний
регистр, другие символы  не изменяет */
/* определяем  массивы заглавных и строчных букв; рядом стоящие
строки склеиваются в одну*/
static high[]="ABCDEFGHIJKLMNOPQRSTUVWXYZ"
               "АБВГДЕЁЖЗИЙКЛМНОПРСТУФХЦЧШЩЪЫЬЭЮЯ";
static low[]="abcdefghijklmnopqrstuvwxyz"
              "абвгдеёжзийклмнопрстуфхцчшщъыьэюя";

char *p;

/* ищем позицию  буквы c в массиве high с помощью продвижения
указателя p, возвращаем букву в  той же позиции из массива low,
(отстоящей от начала low на p-high элементов). Если не нашли,
возвращаем c
*/
for (p=high; *p!='\0'; p++)
    if (c==*p) return  *(low+(p-high));

return c;

}
```

Прежде чем переписывать слова предложения в строку, нужно отвести под эту строку память — это делается в процессе работы программы с помощью функции `malloc()` из библиотеки `<stdlib.h>`. Отвести память статически, т.е. на этапе компиляции, невозможно, поскольку мы не знаем заранее длину предложения, а массивов с переменной длиной нет в стандарте Си-89.

Чтобы узнать длину предложения, нужно просуммировать длину его слов, т.е. строк, указатели на которые хранятся в массиве `argv`. Длину строки (без учета завершающего нуля) вычисляет функция `strlen()`:

```
int  strlen(char * s) {
char *p = s;
while(*p) p++;
return  p-s;

}
```

Теперь все готово, чтобы написать основную функцию — `main()`, решающую задачу.

```
int main (argc c, char *argv[]){
int i, /* индекс */
  len; /* длина предложения */
char *s, /* указатель на начальный символ динамической строки */
     *p, /* указатель для  прохода по строке s */
     *q; /* указатель для прохода по строкт argv[i]   */
/* вычисляем длину предложения */
for (len=0, i=1; i<argc; len+=strlen(argv[i++]) );
/* выделяем память нужной длины под строку s с учетом
завершающего символа '\0' в конце */
s = p = (char*) malloc(len+1);
/* переписываем слова предложения в s,  переводя заглавные буквы
в строчные, и добавляем э\0' в конец  */
for (i = 1, q = argv[i]; i<argc ; i++ ){
    while(*q)  *p++ = tolower(*q++);
}

*p='\0';

/* вычисляем и печатаем ответ */
printf( "%s\n", is_palindrom(s)? "Да" : "Нет" );

return 0;

}
```

В начало текста программы следует добавить директивы `#include <stdio.h>` для использования `printf()` и `#include <stdlib.h>` для использования `malloc()`.

Замечание. Существуют библиотечные функции `tolower()` из библиотеки `<ctype.h>` и `strlen()` из библиотеки `<string.h>`, которые можно использовать вместо описанных выше, включив нужные библиотеки с помощью директивы `#include`.

```c
/* Решение задачи № 5 из контрольной без использования вспомогательного
массива-строки   (автор идеи - Куракин Александр, 2010 год)
                                                                  */

#include<stdio.h>
#include<string.h>
#include<ctype.h>

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

int palindrom=1,  /* Логическая переменная для ответа "Да" или "Нет".
                     Вначале оптимистично полагаем,
                     что "Да" -- введенное предложение является палиндромом.
                     Доверяй, но проверяй:
                      опишем для проверки вспомогательные указатели-индексы,
                      учитывая что слова предложения расположены в строках
                               argv[1], ... argv[argc-1]                  */

    fw=1,         /* Индекс очередного слова при просмотре
                       предложения слева направо, т.е. "вперед" -- forward */

    bw=argc-1,    /* Индекс очередного слова при просмотре
                       предложения справа налево, т.е. "назад"  -- backward */

    p=0,          /* Индекс для прохода слева направо по слову argv[fw]    */

    q             /* Индекс для прохода справа налево по слову argv[bw]    */

       =strlen(argv[argc-1])-1;


/* С помощью пар индексов (fw,p) и (bw,q) просматриваем предложение
   одновременно слева направо и  справа налево и сравниваем соответствующие
   буквы, пока не найдем различие, или пока не достигнем середины
предложения.
   Середина достигнута, если fw>bw  или    если fw==bw и p>=q
*/



while(palindrom && !(fw>bw || fw==bw && p>=q)){

      if(argv[fw][p]=='\0')  /* продвигаем пару (fw,p) на следующее слово  */
            fw++, p=0;

      if(q < 0)              /* продвигаем пару (bw,q) на предыдущее слово */
            bw--, q=strlen(argv[bw])-1;

      palindrom =  /* сравниваем симметрично расположенные
                           символы предложения (без учета разницы в
                       регистре)      */
                  tolower(argv[fw][p])==tolower(argv[bw][q]);
      p++; q--;   /* продвигаем индексы текущих символов
                                          в словах argv[fw] и argv[bw]  */
}

puts(palindrom ? "Да" : "Нет");

return 0;
}

/* Замечание.  Для некоторых предложений-палиндромов (четной длины) делается
одно «лишнее» сравнение пары букв, граничащих с «воображаемой» серединой
  Пример. $>  Prog  aa bbb bbb aa  */
```

6. Напишите программу, которая получает имя текстового файла как параметр и сообщает, сколько раз встречается первая строка в этом файле. Строки могут быть сколь угодно длинными. В конце каждой строки есть символ ’\n’. Пустой файл строк не содержит.

Решение

```
#include <stdio.h>

/* Программа подсчитывает и печатает, сколько раз первая строка
содержится в файле-аргументе.  Проверка на ошибки работы с файлами
не делается. Считаем, что пустой файл не содержит строк. В непустом
файле каждая строка гарантированно завершается символом  '\n'. */

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

    int count=0; /* счетчик числа вхождений первой строки */
     FILE * fp_cur=fopen(argv[1],"r"); /*открываем поток для
                                        чтения текущей строки */
     FILE * fp_fst=fopen(argv[1],"r"); /*открываем поток,
        связанный с тем  же файлом,  для чтения первой строки */
     int  c_cur=fgetc(fp_cur);/*текущий символ текущей строки */
     int  c_fst=fgetc(fp_fst);/* текущий символ первой строки */

     while(c_cur!=EOF){
          while (c_cur==c_fst && c_cur!='\n'){
               c_cur=fgetc(fp_cur);
               c_fst=fgetc(fp_fst);

          }

          count+=(c_cur=='\n' && c_fst=='\n');

          while(c_cur!='\n') c_cur=fgetc(fp_cur);  /*переход к
                                        новой текущей строке */

          c_cur=fgetc(fp_cur);


          rewind(fp_fst); /* возврат к началу первой строки */

          c_fst=fgetc(fp_fst);
     }

     printf("count=%d\n",count);

     return 0;
}
```

6. Напишите программу, которая получает имя текстового файла как параметр и сообщает, сколько раз встречается первая строка в этом файле. Строки могут быть сколь угодно длинными. Соседние строки разделяются символом ’\n’. Пустой файл содержит одну пустую строку.

Решение

```
#include <stdio.h>

/* Программа подсчитывает и печатает, сколько раз первая строка
содержится в файле-аргументе. Проверка на ошибки работы с файлами
не делается. Считаем, что пустой файл содержит одну пустую строку,
а файл из одного символа '\n' содержит две пустых строки
(разделенных этим символом) */

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

    int count=0; /* счетчик числа вхождений первой строки */
     FILE * fp_cur=fopen(argv[1],"r"); /*открываем поток для
чтения
                                          текущей строки */
     FILE * fp_fst=fopen(argv[1],"r"); /*открываем поток,
       связанный с тем же файлом, для чтения первой строки */
     int  c_cur;  /* текущий символ текущей строки */
     int  c_fst;  /* текущий символ первой строки */

    while(1){
        do{
           c_cur=fgetc(fp_cur);
           c_fst=fgetc(fp_fst);
        } while (c_cur==c_fst && c_cur!='\n' && c_cur!= EOF);


count+=(c_cur=='\n'||c_cur==EOF)&&(c_fst=='\n'||c_fst==EOF);

        if (c_cur==EOF) break;

        while(c_cur!='\n'){ /* переход к концу текущей строки */
           c_cur=fgetc(fp_cur);
           if (c_cur==EOF) break;
        }

        rewind(fp_fst);  /* возврат на начало файла для первой
                            строки */
    }

    printf("count=%d\n",count);

    return 0;
}
```