Лабораторная работа №1:

Множество – простейшая информационная конструкция и математическая структура, позволяющая рассматривать какие-то объекты как целое, связывая их. Объекты, связываемые некоторым множеством, называются элементами этого множества. Если объект связан некоторым множеством, то говорят, что существует вхождение объекта в это множество, а объект принадлежит этому множеству. Допускается неограниченное количество вхождений одного объекта в какое-либо множество. Допускаются множества, каждое из которых имеет только одно вхождение какого-либо единственного элемента этого множества, а также допускается множество, не имеющее вхождений, – пустое множество. Среди множеств выделяют ориентированные множества и неориентированные множества. Множество может быть задано с помощью механизма, процедуры. Если некоторая процедура даёт ответ для любого объекта: является он или нет элементом некоторого множества, то такая процедура называется разрешающей процедурой для такого множества. Если же некоторая процедура позволяет получить любой новый элемент некоторого множества, отличный от известных или выданных ранее элементов этого множества, то такая процедура называется порождающей процедурой для такого множества. Неориентированное множество, имеющее малое количество вхождений, может быть представлено в тексте в следующем виде:

$$S = \{a, b, a, a, c\}.$$

Элементы a, b и c принадлежат множеству с именем S, причём множество S имеет три вхождения элемента a ($S|a| = 3$) и по одному вхождению элементов b и c ($S|b| = S|c| = 1$). Неориентированное множество A называют подмножеством неориентированного множества B тогда и только тогда, когда для любого элемента x, который принадлежит множеству A, истинно $A|x| \preceq B|x|$. Если некоторый элемент x не принадлежит множеству S, то истинно $S|x| = 0$. Если множество A является подмножеством множества B, то это записывают так:

$$A \subseteq B.$$

Неориентированные множества A и B равны тогда и только тогда, когда $B \subseteq A$ и $A \subseteq B$.

Все вхождения, которые имеет любое ориентированные множество, упорядочены. Два ориентированных множества равны тогда и только тогда, когда все их элементы входят в одинаковом порядке. Ориентированное множество может быть представлено в тексте в следующем виде:

$$<a, b, a, c>.$$

Множеством *с кратными* вхождениями элементов называют множество S тогда и только тогда, когда существует x такой, что истинно $S|x| > 1$.

Множеством *без кратных* вхождений элементов называют множество S тогда и только тогда, когда для любого x истинно $S|x| < 2$.

Пересечением неориентированных множеств A и B *с учётом* кратных вхождений элементов будем называть неориентированное множество S тогда и только тогда, когда для любого x истинно $S|x| = min\{A|x|, B|x|\}$.

Объединением неориентированных множеств A и B *с учётом* кратных вхождений элементов будем называть неориентированное множество S тогда и только тогда, когда для любого x истинно $S|x| = max\{A|x|, B|x|\}$.

Разностью неориентированных множеств A и B *с учётом* кратных вхождений элементов будем называть неориентированное множество S тогда и только тогда, когда для любого x истинно $S|x| = max\{A|x|-B|x|, 0\}$.

Симметрической разностью неориентированных множеств A и B *с учётом* кратных вхождений элементов будем называть неориентированное множество S тогда и только тогда, когда для любого x истинно $S|x| = max\{A|x|-B|x|, B|x|-A|x|\}$.

Суммой неориентированных множеств ***A*** и ***B*** элементов называют неориентированное множество ***S*** тогда и только тогда, когда для любого ***x*** истинно ***S*|*x*| = *A*|*x*|+*B*|*x*|**.

Пересечением неориентированных множеств ***A*** и ***B*** *без учёта* кратных вхождений элементов будем называть неориентированное множество ***S*** тогда и только тогда, когда для любого ***x*** истинно ***S*|*x*| = *min{A*|*x*|, *B*|*x*|, *1}***.

Объединением неориентированных множеств ***A*** и ***B*** *без учёта* кратных вхождений элементов будем называть неориентированное множество ***S*** тогда и только тогда, когда для любого ***x*** истинно ***S*|*x*| =*min{max{A*|*x*|, *B*|*x*|}, *1}***.

Разностью неориентированных множеств ***A*** и ***B*** *без учёта* кратных вхождений элементов будем называть неориентированное множество ***S*** тогда и только тогда, когда для любого ***x*** истинно ***S*|*x*| = *max{min{A*|*x*|, *1}-min{B*|*x*|, *1}, 0}***.

Симметрической разностью неориентированных множеств ***A*** и ***B*** *без учёта* кратных вхождений элементов будем называть неориентированное множество ***S*** тогда и только тогда, когда для любого ***x*** истинно ***S*|*x*| = *max{min{A*|*x*|, *1}-min{B*|*x*|, *1}, min{B*|*x*|, *1}-min{A*|*x*|, *1}}***.

Булеаном неориентированного множества ***A***, которое является множеством без кратных вхождений элементов, называют неориентированное множество ***S*** тогда и только тогда, когда для любого ***x*** истинно ***S*|*x*| < *2*** и ***((S*|*x*|=*1*) ⇔ (*x* ⊆ *A))***.

Декартовым произведением неориентированных множеств ***A*** и ***B*** называют неориентированное множество ***S*** тогда и только тогда, когда для любого ***z*** истинно: если ***S*|*z*| > *0***, то ***z* = <*x*, *y*>; *S*|*z*| = *A*|*x*|**B*|*y*|** и наоборот.

Задания:

1. Реализовать программу, формирующую множество равное объединению произвольного количества исходных множеств (без учёта кратных вхождений элементов).
2. Реализовать программу, формирующую множество равное пересечению произвольного количества исходных множеств (без учёта кратных вхождений элементов).
3. Реализовать программу, формирующую множество равное симметрической разности произвольного количества исходных множеств (без учёта кратных вхождений элементов).
4. Реализовать программу, формирующую множество равное разности двух исходных множеств (без учёта кратных вхождений элементов).
5. Реализовать программу, формирующую множество равное булеану исходного множества.
6. Реализовать программу, формирующую множество равное декартовому произведению произвольного количества исходных множеств.
7. Реализовать программу, формирующую без повторений всевозможные ориентированные множества из элементов исходного неориентированного множества, количество элементов в сформированных множествах должно быть равно исходному натуральному n.
8. Реализовать программу, формирующую без повторений всевозможные неориентированные множества из элементов исходного неориентированного множества, количество элементов в сформированных множествах должно быть равно исходному натуральному n.
9. Реализовать программу, определяющую является ли одно, либо оба из двух исходных множеств подмножеством или элементом другого.
10. Реализовать программу, формирующую множество равное объединению произвольного количества исходных множеств (с учётом кратных вхождений элементов).
11. Реализовать программу, формирующую множество равное пересечению произвольного количества исходных множеств (с учётом кратных вхождений элементов).
12. Реализовать программу, формирующую множество равное симметрической разности произвольного количества исходных множеств (с учётом кратных вхождений элементов).
13. Реализовать программу, формирующую множество равное разности двух исходных множеств (с учётом кратных вхождений элементов).

Формат данных:

[<имя_множества>=]<множество>

<множество>::=

<ориентированное_множество>|<неориентированное_множество>

<неориентированное_множество>::={[<элемент>{,<элемент>}]}

<ориентированное_множество>::=<<элемент>,<элемент>{,<элемент>}>

<элемент>::=<имя>|<множество>

<имя_множества>::=<имя>

<имя>::=<цифра>|<буква> {<буква>|<подчёркивание>|<цифра>}

<подчёркивание>::=_

<цифра>::=0|…|9

<буква >::=A|…|z

Пример:

A={o,{},A}

B={o,<1,2>,A2,c3,B,b3_A,{},{o,{},A}}

C={o,A2,b3,A,<1,2>,{}}

Пересечением множеств A, B, C будет множество:

{o,{},{o,{},A}}

Требования к реализации программы:

Все корректные входные и выходные данные программы должны строго соответствовать чётко заданному формату данных. Программа должна давать верный результат для любых корректных входных данных. Для любых некорректных входных данных программа должна сообщать о некорректности входных данных и не должна терять управления в результате ошибки исполнения. Все входные данные должны читаться программой из файла.

Исходный текст программы должен содержать имя, фамилию и отчество разработчика. Там же должны быть указаны: дата разработки и описание функциональности и назначения программы. Все функции в исходном тексте программы должны быть прокомментированы. Для каждой функции должны быть указаны и пояснены: назначение, входные данные, выходные данные – в том числе и используемые глобальные переменные. Все глобальные переменные и некоторые локальные переменные должны быть прокомментированы.

Требования к защите:

Требуется присутствие всех разработчиков и наличие: исходного текста программы, исполняемого файла и отчёта по лабораторной работе. Отчёт по лабораторной работе должен содержать: постановку задачи, теоретические сведения, использованные при разработке алгоритма, точное словесное описание алгоритма по шагам либо схему программы, наиболее характерные примеры результатов работы программы. Отчёт может быть предоставлен в электронном виде. При защите лабораторной работы требуется: продемонстрировать работоспособность программы на наиболее характерных тестовых примерах, а также – на тестовых входных данных, предложенных преподавателем; уметь пояснить исходный текст и алгоритм работы программы; показать и предоставить исходный текст и отчёт. Недопустима защита работы в момент отсутствия одного из разработчиков. Работа принимается в случае соблюдения всех требований.

Образец выполнения:

Задание:

1. Реализовать программу, формирующую множество без кратных вхождений элементов на основе исходного неориентированного множества с кратными вхождениями элементами.

Программа:

```
/************************************************
      Название: Cantorizator
   Разработчик: Иванов Иван Иванович
         Дата: 31.08.2002
      Описание: программа получает через консольные
                параметры имя файла, читает из него
                все  множества, оставляя  в  каждом
                только  по одному  вхождению любого
                элемента этого множества;
                результат выводится  на стандартный
                вывод
************************************************/

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

#define OPENSET_SYMBOL  '{'
#define CLOSESET_SYMBOL '}'
#define TERMSET_SYMBOL  ';'
```

```
#define TERMNAME_SYMBOL ','
#define DEFNAME_SYMBOL  '='

#ifndef STR_MAX_LENGTH
#define STR_MAX_LENGTH 80
#else
#error STR_MAX_LENGTH already defined
#endif

#ifndef TRUE
#ifndef FALSE
#define FALSE 0
#define TRUE  1
#else
#define TRUE (!FALSE)
#endif
#else
#ifndef FALSE
#define FALSE (!TRUE)
#endif
#endif
```

```
/***********************************************
                                                *
  объявление структуры  для представления множес- *
  тва                                           *
                                                */
typedef struct {
  int                       u;
  char *                    name;
  struct _element_of_set *  pFirst;
} set;

typedef set element;         /* элемент множества */

/***********************************************
                                                *
  объявление структуры  для представления и орга- *
  зации вхождения элемента во множество         *
                                                */
typedef struct _element_of_set {
  struct _element_of_set *  pNext;
```

```
  element *                 pElement;
} element_of_set;

/***********************************************
                                               *
  функции вывода множества или элемента        *
                                               */
void fprint_els(FILE *, void *, int, int);
void fprint_set(FILE *, set *);
void fprint_element(FILE *, element *);

/***********************************************
                                               *
  функции чтения множества или элемента из задан- *
  ного файла                                   *
                                               */
int fscan_els(FILE *, void **, int);
int fscan_set(FILE *, set **);
int fscan_element(FILE *, element **);

/***********************************************
```

```
                                                    *
  функция инициализации множества                   *
                                                    */
void initialize_set(set *);

/************************************************
                                                    *
  функция сравнения двух элементов                  *
                                                    */
int compare_elements(element *, element *);

/************************************************
                                                    *
  функция добавления элемента во множество          *
                                                    */
element_of_set *  add_element(set *, element *);

/************************************************
                                                    *
  функция поиска элемента во множестве              *
                                                    */
```

```
element *  find_element(set *, element *);

/************************************************
                                                *
  функция получения  следующего  элемента из мно- *
  жества                                        *
                                                */
element *  get_next_element(set *, element_of_set *);

/************************************************
                                                *
  функция установки  следующего  элемента из мно- *
  жества                                        *
                                                */
void *  set_next_element(set *,
                         element_of_set *,
                         element *);

/************************************************
                                                *
  функция перехода к следующему вхождению       *
```

```
                                                        */
void go_next(set *, element_of_set **);

/************************************************
                                                        *
  функция исключения последующего элемента из           *
  множества                                             *
                                                        */
void remove_next_element(set *, element_of_set *);

/************************************************
                                                        *
  функция уничтожения элементов множества               *
                                                        */
void destroy_elements(set *);

/************************************************
                                                        *
  функция очистки множества                             *
                                                        */
void clear_set(set *);
```

```
#include <string.h>

set names;     /* множество прочитанных элементов */

/***********************************************
                                               *
  основная функция; входной параметр - имя файла; *
                                               */
int main (int argc, char *  argv[]) {
  int return_value = 0;
  FILE *  input;
  set general_set, *  pSet = NULL, copy;
  element_of_set *  pEOSG = NULL, *  pEOS = NULL;
  element *  pElement;
  if (argc == 2) if ((input = fopen(argv[1],"r")) != NULL) {

        /* инициализация и чтение данных из файла */

    initialize_set(&names);
    initialize_set(&general_set);
```

```
if((return_value = fscan_set(input, &pSet)) == 0) return_value--;
else {
    add_element(&general_set,pSet);
    while((return_value = fscan_set(input, &pSet)) > 0) add_element(&general_set,pSet);
}
fclose(input);

             /* уничтожение кратных вхождений
                          и вывод результата */

if (return_value == 0) do {
  if ((pSet = get_next_element(&general_set, pEOSG)) != NULL) {
    initialize_set(&copy);
    do {
      if ((pElement = get_next_element(pSet, pEOS)) != NULL) {
        if (find_element(&copy,pElement) == NULL) add_element(&copy,pElement);
      }
      go_next(pSet, &pEOS);
    }
    while (pEOS != NULL);
```

```
        fprint_set(stdout, &copy);
        clear_set(&copy);
      }
      go_next(&general_set, &pEOSG);
    }
    while(pEOSG != NULL);
    else printf ("Syntax error.\n");

    destroy_elements(&names);
    return_value = 1;
  }
  return return_value;
}

/************************************************
                                                *
  реализация функций вывода множества или элемен- *
  та                                            *
  ВХОДНЫЕ ДАННЫЕ:                               *
   f - файл-приёмник                            *
   p - указатель на элемент или множество       *
```

```
   l - позиция                                          *
   i - отступ                                           *
                                                        */
void fprint_elements (FILE *  f, set *  p, int l, int i) {
                                      /* рекурсия */
  element_of_set *  pEOS = NULL;
  element *  pElement;
  do {
    if ((pElement = get_next_element(p, pEOS)) != NULL) {
      if (pEOS != NULL) {
        fprintf(f,"%c ",TERMNAME_SYMBOL);
        l += 2*sizeof(char);
      }
      if (pElement->name != NULL) {
        l += strlen(pElement->name);
        if (l > STR_MAX_LENGTH) {
          for (l = 0, fprintf(f,"\n"); l < i; l++)
            fprintf(f," ");
          l += strlen(pElement->name);
        }
        fprintf(f,"%s",pElement->name);
```

```
      }
      else fprint_els(f,pElement,l,i);
    }
    go_next(p, &pEOS);
  }
  while(pEOS != NULL);
  return;
}

void fprint_set(FILE *  f, set *  p) {
  char str[STR_MAX_LENGTH+sizeof(char)],
       dn[2*sizeof(char)] = {DEFNAME_SYMBOL,'\0'};
  int l = 0, i;
  if (p->name != NULL) {
                                    /* выводим имя */

    strncpy(str,p->name,STR_MAX_LENGTH);
    if (strlen(str) < STR_MAX_LENGTH) strcat(str," ");
    else {
      fprintf(f,"%s\n",str);
      strcpy(str," ");
```

```
  }
  if (strlen(str) < STR_MAX_LENGTH) strcat(str,dn);
  else {
    fprintf(f,"%s\n",str);
    strcpy(str,dn);
  }
  if (strlen(str) < STR_MAX_LENGTH) strcat(str," ");
  else {
    fprintf(f,"%s\n",str);
    strcpy(str," ");
  }
  fprintf(f,"%s",str);
  l += strlen(str);
}
else l += sizeof(char);
fprintf(f,"%c",OPENSET_SYMBOL);
i = l;

                  /* выводим элементы множества */

fprint_elements(f,p,l,i);
```

```
  fprintf(f,"%c%c\n",CLOSESET_SYMBOL,TERMSET_SYMBOL);
  return;
}

void fprint_element(FILE *  f, element *  p) {fprint_els(f,p,0,0);return;};

void fprint_els(FILE *  f, void *  _p, int l, int i) {
  char str[STR_MAX_LENGTH+sizeof(char)];
  element *  p = (element *)_p;
  if (p->name != NULL) {
                                    /* выводим имя */

    strncpy(str,p->name,STR_MAX_LENGTH);
    fprintf(f,"%s",str);
  }
  else {
    fprintf(f,"%c",OPENSET_SYMBOL);
    i = l += sizeof(char);

                    /* выводим элементы множества */
```

```
    fprint_elements(f,p,l,i);
    fprintf(f,"%c",CLOSESET_SYMBOL);
  }
  return;
};

/************************************************
                                                *
  реализация функций ввода множества или элемента *
  ВХОДНЫЕ ДАННЫЕ:                               *
   f - файл-источник                            *
   p - указатель на указатель на элемент или мно- *
       жество                                   *
   l - ожидается только множество либо не только  *
  ВЫХОДНЫЕ ДАННЫЕ:                              *
   количество прочитанных элементов-множеств,     *
   в случае синтаксической ошибки возвращает -1;  *
   возвращает указатель на элемент или множество  *
  ИСПОЛЬЗУЕТСЯ ГЛОБАЛЬНАЯ СТРУКТУРА names         *
                                                */
#define MAX_STR   255
```

```
#define SYNTAX_ERR -1

typedef enum {LOOKING,NAME_READING,NAME_READ,SET_LOOKING,SET_READ} state_type;

int fscan_set(FILE *  f, set **  p)         {return fscan_els(f,(void**)p,TRUE);}
int fscan_element(FILE *  f, element **  p) {return fscan_els(f,(void**)p,FALSE);}
int fscan_els(FILE *  f, void **  p, int l) {
                                    /* рекурсия */
  const char *  nsymbols = "ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz0123456789_",
             *  delimiters = " \t\n\r\f\v";
  int return_value = 0, len;
  char pc[2*sizeof(char)] = {OPENSET_SYMBOL,'\0'};
  char str[MAX_STR+sizeof(char)] = "";
  set *  s = NULL;
  element *  e = NULL;
  state_type state = LOOKING;
  while (!feof(f) && *pc != TERMNAME_SYMBOL) {
    *pc = fgetc(f);
    switch (*pc) {
                                                /*
          к моменту чтения DEFNAME_SYMBOL должно *
```

```
       быть уже прочитано имя                 *
                                              */
case(DEFNAME_SYMBOL):
  if (state != NAME_READ &&
      state != NAME_READING) return_value = SYNTAX_ERR;
  else state = SET_LOOKING;
  break;
                                             /*
            после обнаружения OPENSET_SYMBOL *
            создаём множество  и  читаем его *
            поэлементно                      *
                                              */
case(OPENSET_SYMBOL):
  if (state != LOOKING && state != SET_LOOKING) return_value = SYNTAX_ERR;
  else {
    s = (set *)malloc(sizeof(set));
    initialize_set(s);
    while((return_value = fscan_element(f, &e)) > 0) add_element(s,e);
    if (return_value < 0) free(s);
    else state = SET_READ;
  }
```

```
  break;
                                                /*
          TERMSET_SYMBOL может быть обнаружен   *
          только  если мы ожидали и прочитали   *
          множество                             *
                                                */
case(TERMSET_SYMBOL):
  if (!l || state != SET_READ) {
    return_value = SYNTAX_ERR;
    break;
  }
  l = FALSE;
                                                /*
        к моменту чтения CLOSESET_SYMBOL имя    *
        или  множество  должно  читаться или    *
        быть уже прочитанным,  либо в случае    *
        поиска  элементов пустого  множества    *
        мы  можем находиться  в  режиме про-    *
        смотра                                  *
                                                */
case(CLOSESET_SYMBOL):
```

```
  if (state == LOOKING) {
    if (l) return_value = SYNTAX_ERR;
    state = SET_READ;
    *pc = TERMNAME_SYMBOL;
    break;
  }
  if (*pc != TERMSET_SYMBOL) ungetc(*pc,f);
  *pc = TERMNAME_SYMBOL;
                                              /*
         TERMNAME_SYMBOL может быть обнаружен *
         только на момент прочтения имени или *
         множества;                           *
         если такой элемент уже есть,  то но- *
         вый удаляем  и  используем  наличес- *
         твующий, иначе запоминаем новый      *
                                              */
case(TERMNAME_SYMBOL):
  if (state != NAME_READING &&
      state != NAME_READ && (state != SET_READ || l)) {
    return_value = SYNTAX_ERR;
    break;
```

```
}
if (state == SET_READ) e = s;
else {
  e = (element *)malloc(sizeof(element));
  initialize_set((set *)e);
  state = NAME_READ;
}
if ((len = strlen(str)) != 0) {
  e->name = (char*)malloc(++len*sizeof(char));
  strcpy(e->name,str);
}
return_value++;
if (state == SET_READ) e->u = FALSE;
if ((*p = find_element(&names, e)) != NULL) {
  if (len > 1) ((element *)*p)->name = e->name;
  else free(e->name);
  if (state == SET_READ) if (((element *)*p)->u == TRUE) ((element *)*p)->pFirst = e->pFirst;
  else clear_set(e);
  if (((element *)*p)->u == TRUE) ((element *)*p)->u = e->u;
  free(e);
}
```

```
  else {
    add_element(&names,e);
    *p = e;
  }
  break;
                                          /*
         если в режиме  просмотра обнаружива- *
         ется  символ  имени, то приступаем к *
         чтению имени;                        *
         в режиме чтения читаем, пока хватает *
         места;                               *
         если в процессе чтения обнаруживает- *
         ся разделитель, то завершаем чтение; *
         во  всех остальных  случаях  -  син- *
         таксическая ошибка                   *
                                              */
default:
  if (strchr(nsymbols,*pc) != NULL) {
    if (state == LOOKING) state = NAME_READING;
    if (state != NAME_READING) return_value = SYNTAX_ERR;
    else if (strlen(str) < MAX_STR) strcat(str,pc);
```

```
        else return_value = SYNTAX_ERR;
      }
      else if (strchr(delimiters,*pc) != NULL) {
        if (state == NAME_READING) state = NAME_READ;
      }
      else if (*pc != EOF) return_value = SYNTAX_ERR;
    }
    if (return_value < 0) break;
  }

  if (state != NAME_READ &&
      state != SET_READ &&
      state != LOOKING) return_value = SYNTAX_ERR;
  return return_value;
}

/************************************************
                                                *
  реализация функции сравнения двух элементов   *
  ВХОДНЫЕ ДАННЫЕ:                               *
   o - оригинал                                 *
```

```
   p - образец                                          *
  ВЫХОДНЫЕ ДАННЫЕ:                                      *
   возвращает TRUE/FALSE - равно/неравно                *
                                                        */
int compare_elements(element *  o, element *  p) {
                                    /* рекурсия */
  int pu, ou, return_value = TRUE;
  element_of_set *  pEOSO = NULL, *  pEOSP = NULL;
  element *  pOElement, *  pPElement;
  set s;
  if (o->name != NULL && p->name != NULL &&
      (o->u == TRUE || p->u == TRUE))
    return_value = (strcmp(o->name,p->name) == 0)? TRUE: FALSE;
  else if ((o->name == p->name) ||
      (o->name != p->name && o->u == FALSE && p->u == FALSE)) {
    if (o->pFirst != p->pFirst) {
                              /* копируем оригинал */

      initialize_set(&s);
      do {
        if ((pOElement = get_next_element(o, pEOSO)) != NULL) {
```

```
    add_element(&s,pOElement);
  }
  go_next(o, &pEOSO);
}
while (pEOSO != NULL);

                              /* сравниваем */

do if ((pPElement = get_next_element(p, pEOSP)) != NULL) {
  do if ((pOElement = get_next_element(&s, pEOSO)) != NULL) {
    pu = pPElement->u;
    ou = pOElement->u;
    pPElement->u = pOElement->u = TRUE;
    set_next_element(p,pEOSP,pOElement);
    if ((return_value = compare_elements(pOElement,pPElement)) == TRUE) {
      remove_next_element(&s,pEOSO);
      pEOSO = NULL;
    }
    else go_next(&s, &pEOSO);
    set_next_element(p,pEOSP,pPElement);
    pPElement->u = pu;
```

```
          pOElement->u = ou;
        }
        else {
          return_value = FALSE;
          break;
        }
        while (pEOSO != NULL);
        if (!return_value) pEOSP = NULL; /* return_value == FALSE */
        else go_next(p, &pEOSP);
      }
      else break;
      while (pEOSP != NULL);

/* проверяем, чтобы не оставалось ничего лишнего */

      do {
        if ((pOElement = get_next_element(&s, NULL)) != NULL) {
          remove_next_element(&s,NULL);
          return_value = FALSE;
        }
      }
```

```
      while (pOElement != NULL);
    }
  }
  else return_value = FALSE;
  return return_value;
}

/************************************************
                                                *
  реализация функции инициализации множества    *
  ВХОДНЫЕ ДАННЫЕ:                               *
   pSet - указатель на множество                *
                                                */
void initialize_set (set *  pSet) {
  pSet->u      = TRUE;
  pSet->name   = NULL;
  pSet->pFirst = NULL;
  return;
}

/************************************************
```

```
                                                *
  реализация функции добавления элемента  во мно- *
  жество                                        *
  ВХОДНЫЕ ДАННЫЕ:                               *
   s - указатель на множество                   *
   e - указатель на элемент                     *
  ВЫХОДНЫЕ ДАННЫЕ:                              *
   возвращает NULL в случае неудачи             *
                                                */
element_of_set *  add_element(set *  s, element *  e) {
  element_of_set *  pEOS = (element_of_set *)malloc(sizeof(element_of_set));
  if (pEOS != NULL) {
    pEOS->pNext = s->pFirst;
    pEOS->pElement = e;
    s->pFirst = pEOS;
  }
  return pEOS;
}

/***********************************************
                                                *
```

```
  реализация функции поиска элемента во множестве *
  ВХОДНЫЕ ДАННЫЕ:                                  *
   s - указатель на множество                      *
   e - указатель на элемент                        *
  ВЫХОДНЫЕ ДАННЫЕ:                                 *
   возвращает указатель на искомый элемент         *
                                                   */
element *  find_element(set *  s, element *  e) {
  element_of_set *  pEOS = NULL;
  element *  pElement;
  do if ((pElement = get_next_element(s, pEOS)) != NULL) {
    if (compare_elements(e,pElement)) pEOS = NULL;
    else go_next(s, &pEOS), pElement = NULL;
  }
  else break;
  while(pEOS != NULL);
  return pElement;
}

/************************************************
                                                 *
```

```
  реализация функции получения следующего элемен- *
  та из множества                                  *
  ВХОДНЫЕ ДАННЫЕ:                                  *
   s - указатель на множество                      *
   e - указатель на вхождение                      *
  ВЫХОДНЫЕ ДАННЫЕ:                                 *
   возвращает указатель на следующий элемент       *
                                                   */
element *  get_next_element(set *  s, element_of_set *  e) {
  return (e == NULL)? (s->pFirst != NULL)? s->pFirst->pElement: NULL:
                      (e->pNext != NULL)? e->pNext->pElement: NULL;
}

/************************************************
                                                 *
  реализация функции установки следующего элемен- *
  та из множества                                *
  ВХОДНЫЕ ДАННЫЕ:                                *
   s - указатель на множество                    *
   e - указатель на вхождение                    *
   p - указатель на элемент                      *
```

```
  ВЫХОДНЫЕ ДАННЫЕ:                                  *
   возвращает NULL в случае неудачи                 *
                                                    */
void *  set_next_element(set *  s, element_of_set *  e, element *  p) {
  element_of_set * pEOS  = (e == NULL)? s->pFirst: e->pNext;
  if (pEOS != NULL) pEOS->pElement = p;
  else pEOS = add_element (s,p);
  return pEOS;
}

/************************************************
                                                    *
  реализация функции перехода  к следующему вхож- *
  дению                                             *
  ВХОДНЫЕ ДАННЫЕ:                                   *
   s - указатель на множество                       *
   p - указатель на указатель на вхождение          *
  ВЫХОДНЫЕ ДАННЫЕ:                                  *
   *p - указатель на следующее вхождение            *
                                                    */
void go_next(set *  s, element_of_set **  p) {
```

```
  *p = (*p == NULL)? s->pFirst: (*p)->pNext;
  return;
}

/************************************************
                                                *
  реализация функции исключения последующего эле- *
  мента из множества                            *
  ВХОДНЫЕ ДАННЫЕ:                               *
   s - указатель на множество                   *
   e - указатель на вхождение                   *
                                                */
void remove_next_element(set *  s, element_of_set *  e) {
  element_of_set *  p;
  if (e == NULL) {
    if (s->pFirst != NULL) {
      p = s->pFirst->pNext;
      free(s->pFirst);
      s->pFirst = p;
    }
  }
  else {
    p = e->pNext->pNext;
    free(e->pNext);
    e->pNext = p;
  }
  return;
}

/************************************************
                                                *
  реализация  функции  уничтожения элементов мно- *
  жества                                        *
  ВХОДНЫЕ ДАННЫЕ:                               *
   s - множество                                *
                                                */
void destroy_elements(set *  s) {
  element *  pElement;
  do {
    if ((pElement = get_next_element(s, NULL)) != NULL) {
      remove_next_element(s,NULL);
      clear_set(pElement);
      if (pElement->name != NULL) free(pElement->name);
      free(pElement);
    }
  }
  while(pElement != NULL);
  return;
}

/************************************************
                                                *
  реализация функции очистки множества          *
  ВХОДНЫЕ ДАННЫЕ:                               *
   s - множество                                *
                                                */
void clear_set(set *  s) {
  element *  pElement;
  do {
    if ((pElement = get_next_element(s, NULL)) != NULL) {
      remove_next_element(s,NULL);
    }
  }
  while(pElement != NULL);
  return;
}
```