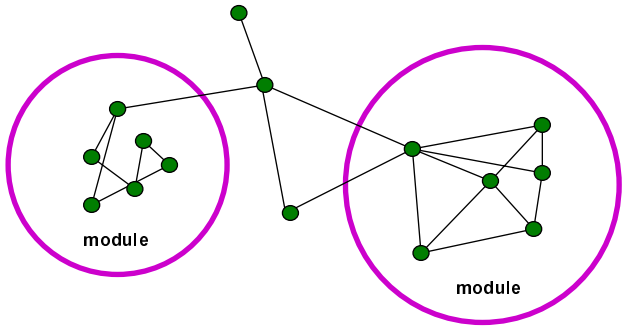
53

Chapter 2

Abstract Data Types

54

Modularity



● - functions.
\ - function calls.

Program elements that are tightly coupled should be organized as *modules*.
Their connection to the outside world should be minimal.

Copyright 1995 - 2003
C. Gotsman & Y.M. Kimchi
Computer Science Dept. Technion

For personal use only

55

Modularity in C

is achieved via Multi-Source Compilation

swap.c

```
void swap(int *x, int *y)
{int t=*x; *x=*y; *y=t; }
```

main.c

```
int main(void)
{ int a=2,b=3;
  swap(&a,&b);
}
```

```
gcc -c swap.c
gcc -c main.c
gcc main.o swap.o -o swap
```

- Code may be divided up between a number of physical source files.
- Each source file `src.c` may be *compiled* separately and independently using

  `gcc -c src.c`

  creating the object file `src.o`.
- Two (or more) object files `src1.o`, `src2.o` may be *linked* to an executable `ex` using

  `gcc src1.o src2.o -o ex`

56

Declarations and Definitions in C

swap.h

```
void swap(int*,int*);
```

swap.c

```
#include "swap.h"
void swap(int *x, int *y)
{int t=*x; *x=*y; *y=t; }
```

file1.c

```
#include "swap.h"
int a,b;
swap(&a,&b);  /* OK */
```

file2.c

```
#include "swap.h"
int x = swap(3,4); /*Bang*/
```

- A function *declaration* informs the compiler of the types of the function arguments, and the type of the returned value (prototype). Declarations may appear many times (as long as they are consistent).
- A function *definition* contains the function body (*implementation*). This may appear only once. A definition serves as a declaration.
- All functions should be declared (or defined) before used. If a function is used in a file, but the definition is on a different file, the declaration should be *included* into the file.
- In C, an undeclared function is assumed to return `int`, and its arguments are not checked.

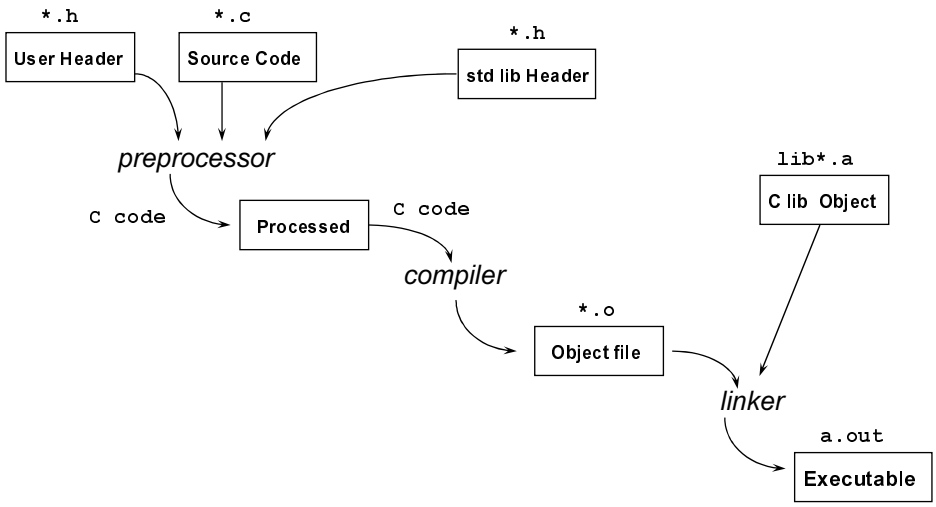
  Never use this "feature" !

Copyright 1995 - 2003
C. Gotsman & Y.M. Kimchi
Computer Science Dept. Technion

For personal use only

57

From Source to Executable



58

Static Functions

file1.c

```
void do_it(double x, double y)
{ ... }
int a,b;
do_it(a,b); /* calls first do_it() */
```

- **Regular functions are recognized (and may be used) by all C code linked together.**

file2.c

```
static void do_it(double x, double y)
{ ... }
double a,b;
do_it(a,b); /* calls second do_it()*/
```

- **Static functions are recognized *only* in the file in which they are defined.**

file3.c

```
double i,j;
do_it(i,j); /* calls first do_it(),
                    no ambiguities      */
```

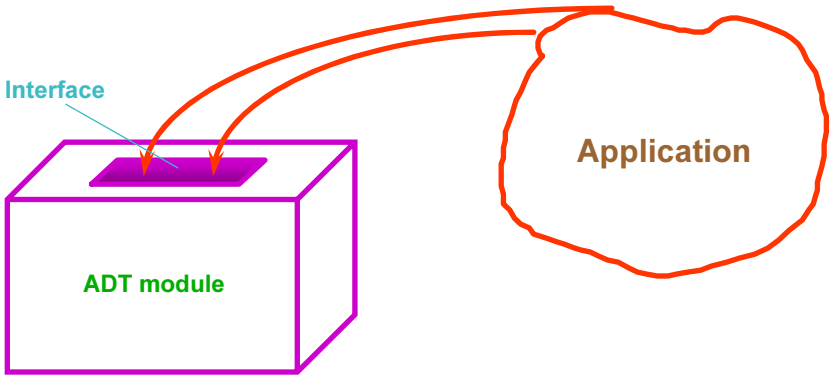
- **The linker will not handle two functions with the same name.**

Copyright 1995 - 2003
C. Gotsman & Y.M. Kimchi
Computer Science Dept. Technion

For personal use only

59

Abstract Data Type



An ADT is a module performing well defined functions with a compact interface.

Example: Complex number.

60

Example: Complex Number

Interface

- Create complex number.
- Destroy complex number.
- Get real part.
- Get imaginary part.
- Get absolute value.
- Add/Subtract/Multiply/Divide complex numbers.
- Print complex number.

Copyright 1995 - 2003
C. Gotsman & Y.M. Kimchi
Computer Science Dept. Technion

For personal use only

61

Information Hiding

- User of ADT sees only the interface.
- Implementation details are hidden.

Advantages

- Implementation may be changed / improved without user knowing, as long as interface is preserved.
- Provides user with abstract "black box".
- No technical details to confuse user.

62

Example: SET of Elements

Interface

- *Create* set of specific elements.
- *Destroy* set.
- *Add* element to set.
- *Remove* element from set.
- *Is* element *in* set ?
- *Is* set *empty* ?
- Enumerate set elements.
- *Union* of two sets.
- *Intersection* of two sets.
- *Print* set.

Other possible interface functions:

Copy set.
Are two sets *equal* ?
Is a set a *subset* of another ?
Size of a set.

Why should (not) these functions be part of the interface ?

Copyright 1995 - 2003
C. Gotsman & Y.M. Kimchi
Computer Science Dept. Technion

For personal use only

63

Implementing ADT's in C

- **Separate** header (.h) and body (.c) files.
  - Interface declarations in header.
  - Implementation in body.
- **Separate** Compilation
  - ADT body may be compiled to object (.o) file *without* application.
  - Application may be compiled with *different* implementations.

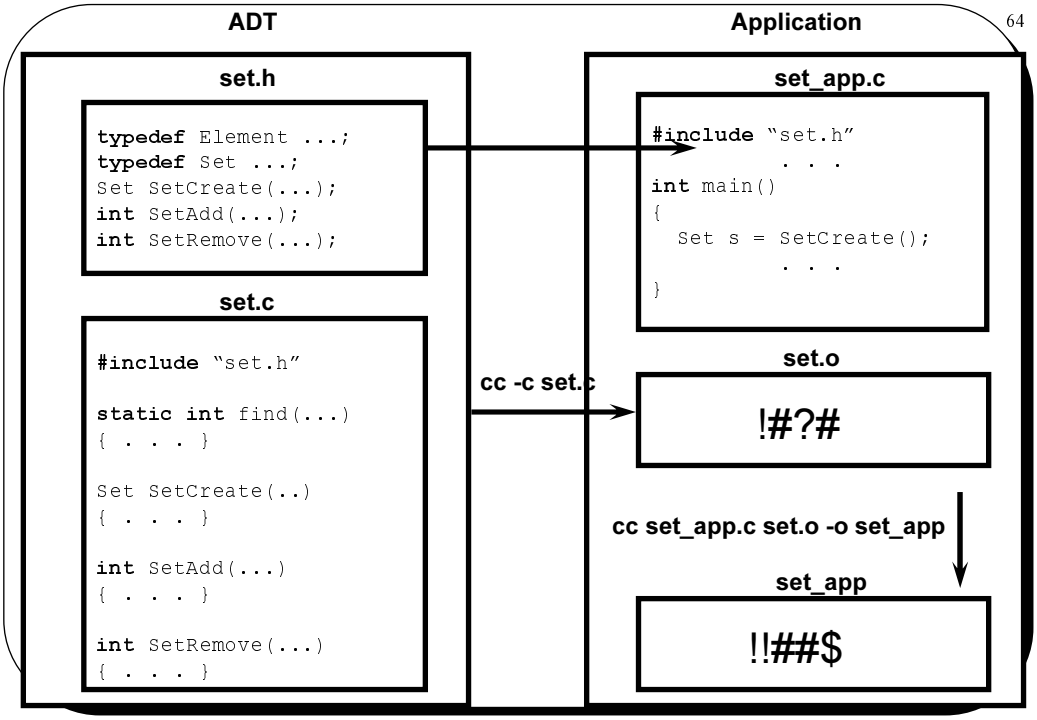
Example:

ADT files: *set.h set.c*
Application file: set*_app.c*

ADT compilation:
cc -c set.c → *set.o*

Application compilation:
cc set.o set_app.c -o set_app → *set_app*

64



Copyright 1995 - 2003
C. Gotsman & Y.M. Kimchi
Computer Science Dept. Technion

For personal use only

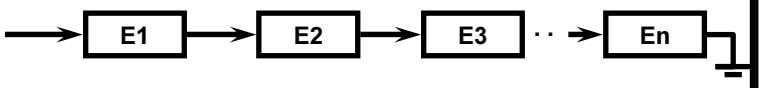
65

SET - Possible Implementations

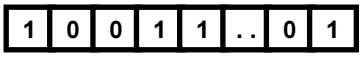
Following are (only) a few of available **container** data-structures, that are capable of storing objects in such ways that allow the behaviour (and an interface) of a set. Even these examples are quite abstract, and each may be implemented in various ways.

- **Array of elements**

- **Linked list**



- **Bit array**



66

Array Implementation of SET

- Finite length array of pointers to "elements".
- Empty cells are marked by NULL.
- New element is placed in first empty cell.
- Element is removed by marking cell as empty.
- Element is searched for exhaustively.

Note: The diagram does not reflect the usage of pointers, as stated in the text

Copyright 1995 - 2003
C. Gotsman & Y.M. Kimchi
Computer Science Dept. Technion

For personal use only

67

Other SET Implementations Using an Array

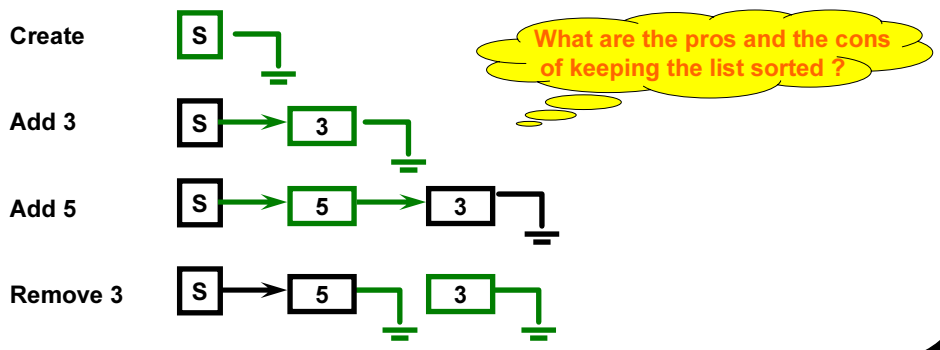
- Existing set elements are *moved* to fill in holes.
- Elements are *sorted* in some order (if possible).
- If the array is full, increase its size.

What are the pros and cons of these methods ?

68

Linked-List Implementation

- Each record represents an element of the set.
- New element is placed at head of list.
- Element is removed by bypassing it.
- Element is searched for exhaustively.



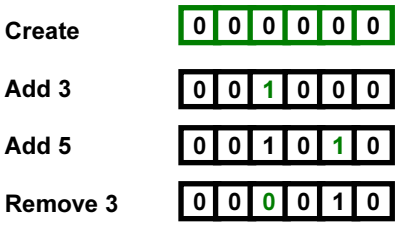
Copyright 1995 - 2003
C. Gotsman & Y.M. Kimchi
Computer Science Dept. Technion

For personal use only

69

Bit Array Implementation

- Each *possible element* is represented by one bit (present / not present).
- Add / Remove by setting a single bit to 1 or 0.
- Search by checking a single bit.
- Intersection / Union by bitwise operators.
- Set domain must be fixed.



Domain = {1,2,3,4,5,6}

70

Implementation Pros and Cons

	Array	Linked List	Bit Array
Set domain size	unlimited	unlimited	limited
Maximal set size	limited	unlimited	limited
Implement	easy	difficult	easy
Search complexity	O(n)	O(n)	O(1)

Copyright 1995 - 2003
C. Gotsman & Y.M. Kimchi
Computer Science Dept. Technion

For personal use only

71

SET Contents

Q: What is an element ?

A: An element can be taken from a fixed *domain*.

Examples of domains:

- Set of integers.
- Set of vectors.
- Set of structures.
- Set of sets !!

72

SET Implementation in C

- Array of *pointers* to `void`.
- These pointers are cast to the specific element type being used.
- User must supply four functions (the semantics) specific to the element domain on set creation.
  - compare function.
  - copy function.
  - free function.
  - label function (i.e, get a printable value).
- These four functions uniquely characterize the set domain and its intended usage.

Copyright 1995 - 2003
C. Gotsman & Y.M. Kimchi
Computer Science Dept. Technion

For personal use only

73

Set Implementation - Coding Style

- In contrast to good practice, in many cases we omit braces (`{, }`) around a single statement at an `if` statement or at a loop statement, e.g,

```
if (..)          instead of      if (..) {
  Do();                              Do();
                                   }
```

(think what happens if `Do()` is a statement MACRO and there is an `else`)

- This is done in order to save lines of code, since space is limited on each slide.
- Note, however, that in the code, that is fully implemented in this chapter, there is not a single function longer than 16 lines (a single slide only).
- Something to think about . . .

74

set.h - type definitions

```
#ifndef SET_HDR_  /* '{' */
#define SET_HDR_

typedef enum {FALSE, TRUE} bool; /* C99 has a "faked" type bool
*/

typedef enum {
   SUCCESS = 1,
   FAILURE = -1
} Result;

typedef void* Element;

typedef struct set_rec* Set;
```

Note that the contents of the `set_rec` structure are still unknown.

Copyright 1995 - 2003
C. Gotsman & Y.M. Kimchi
Computer Science Dept. Technion

For personal use only

75

set.h - ADT Interface Declarations

```
Set       SetCreate        (bool    cmp(Element,Element),
  /* These parameters */   Element cpy(Element),
  /* are       pointers */  char*   lbl(Element),
  /* to      functions */   void    fre(Element));
void      SetDestroy       (Set);
Result    SetAdd           (Set,Element);
Result    SetRemove        (Set,Element);
bool      SetIsIn          (Set,Element);
int       SetSize          (Set);
int       SetMaxSize       (Set);  /* 0 (or -1) for unlimited */
Set       SetIntersection  (Set,Set);
Set       SetUnion         (Set,Set);
void      SetPrint         (Set);
#define   SetIsEmpty(s)    ((bool)(0 == SetSize(s)))

/* The most primitive iterator - nested loops are forbidden! */
Element   SetNext          (Set);
Element   SetFirst         (Set);
#define SET_FOREACH(e,s) \
            for (e=SetFirst(s);e!=NULL;e=SetNext(s))
#endif /* SET_HDR_ '}' */
```

76

set_app.c

```
#include <stdio.h>
#include <stdlib.h>          /* A simple application   */
#include "set.h"             /* using sets of integers. */
#define LABEL_SIZE 20
#define TO_INT(e)   (*(int*)(e))
#define ALLOCATE(var,type)   \
     { if((var =(type *) malloc(sizeof(type))) == NULL)\
       { fprintf(stderr,"Cannot allocate\n"); exit(1);}}


char* int_lbl(Element e)
{   char *s;
    if ((s = malloc(LABEL_SIZE)) == NULL)  return NULL;
    sprintf(s,"%d",TO_INT(e));
    return s;
}
```

Copyright 1995 - 2003
C. Gotsman & Y.M. Kimchi
Computer Science Dept. Technion

For personal use only

77

set_app.c

```
bool int_cmp(Element e1,Element e2)
{   return (bool)(TO_INT(e1) == TO_INT(e2));
}


Element int_cpy(Element e)
{   int *j;
    ALLOCATE(j,int);
    *j = TO_INT(e);
    return j;
}
```

78

set_app.c

```
int main(void)
{   Set s1 = SetCreate(int_cmp,int_cpy,int_lbl,free),
        s2 = SetCreate(int_cmp,int_cpy,int_lbl,free);
    for (int j = 0; j < 20; j += 2) {  // C99
      SetAdd(s1,&j);
    }
    for (int j = 0; j < 12; j += 3) {
      SetAdd(s2,&j);
    }
    Set s3 = SetUnion(s1,s2),           /* C99 */
        s4 = SetIntersection(s1,s2);
    SetPrint(s1);    SetPrint(s2);
    SetPrint(s3);    SetPrint(s4);
    SetDestroy(s1);  SetDestroy(s2);
    SetDestroy(s3);  SetDestroy(s4);
    return 0;
}
```

Copyright 1995 - 2003
C. Gotsman & Y.M. Kimchi
Computer Science Dept. Technion

For personal use only

79

Makefile

ADT

```
set.o: set.h set.c
    gcc -c set.c
```

Client is not interested in (and does not have) set.c

Application

```
set_app: set_app.c set.h set.o
    gcc set_app.c set.o -o set_app
```

output of set_app

```
0 2 4 6 8 10 12 14 16 18
0 3 6 9
0 2 4 6 8 10 12 14 16 18 3 9
0 6
```

80

set.c - Implementation as array

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include "set.h"
#define MAX_SET_SIZE 1000
struct set_rec {
   Element  elem[MAX_SET_SIZE];
   int      n_elem; /* set size */
   int      last;   /* index of last element */
   int      curr;   /* index of iterator */
   bool     (*lm_cmp) (Element,Element);
   Element  (*lm_cpy) (Element);
   char*    (*lm_lbl) (Element);
   void     (*lm_fre) (Element);
};

#define CREATE_AS(s)  \
  SetCreate((s)->lm_cmp,(s)->lm_cpy,(s)->lm_lbl,(s)->lm_fre)
```

Copyright 1995 - 2003
C. Gotsman & Y.M. Kimchi
Computer Science Dept. Technion

For personal use only

81

set.c - Internal functions

```
                    /* find element in set */
static int find(Set s, Element e)
{   for (int j = 0; j < s->last; j++) { // C99
      if (s->elem[j]!=NULL && s->lm_cmp(s->elem[j],e))
         return j;
    }
    return FAILURE;
}

             /* find first empty position in array */
static int find_empty(Set s)
{   int j;
    if (s->n_elem == MAX_SET_SIZE) return FAILURE;
    for (j = 0; j < s->last && s->elem[j] != NULL; j++)
      ;
    return j;  // j  cannot be local in for-loop
}
```

82

set.c - Interface Functions

```
Set SetCreate(bool    lm_cmp(Element,Element),
              Element lm_cpy(Element),
              char*   lm_lbl(Element),
              void    lm_fre(Element))
{   Set s = (Set) malloc(sizeof(struct set_rec));
    if (s == NULL)  return NULL;
    s->lm_cmp = lm_cmp;
    s->lm_cpy = lm_cpy;
    s->lm_lbl = lm_lbl;
    s->lm_fre = lm_fre;
    s->n_elem = s->curr = s->last = 0;
    return s;   /* A pointer to a record on the heap   */
}

void SetDestroy(Set s)
{   int i;
    if (s == NULL) return;
    for (i = 0; i < s->last; i++)
      if (s->elem[i] != NULL) s->lm_fre(s->elem[i]);
    free(s);   return;
}
```

Copyright 1995 - 2003
C. Gotsman & Y.M. Kimchi
Computer Science Dept. Technion

For personal use only

83

set.c

Note the semantics for SUCCESS

A hidden bug in this function

```
Result SetAdd(Set s, Element e)
{  int empty;
   if (find(s,e) != FAILURE) return SUCCESS;
   if ((empty = find_empty(s)) == FAILURE) return FAILURE;
   s->elem[empty] = s->lm_cpy(e);
   s->n_elem++;
   if (empty == s->last) s->last++;
   return SUCCESS;
}

Result SetRemove(Set s, Element e)
{  int j = find(s,e);
   if (j == FAILURE) return FAILURE;
   s->lm_fre(s->elem[j]);
   s->elem[j] = NULL;
   s->n_elem--;
   return SUCCESS;
}
```

84

set.c

```
bool SetIsIn(Set s, Element e)
{ return (bool)(find(s,e) != FAILURE);
}

int SetSize(Set s)
{ return s->n_elem;}

int SetMaxSize(Set s)      /* returns 0 (or -1) if unlimited */
{ return MAX_SET_SIZE;}    /* Cannot be a MACRO */

Element SetNext(Set s)
{ while (s->curr<s->last && s->elem[s->curr] == NULL) s->curr++;
  if (s->curr == s->last) return NULL;
  return s->elem[s->curr++];
}

Element SetFirst(Set s)
{ s->curr = 0;
  return SetNext(s);
}
```

Copyright 1995 - 2003
C. Gotsman & Y.M. Kimchi
Computer Science Dept. Technion

For personal use only

85

set.c

```
Set SetIntersection(Set s1, Set s2)
{  int i;
   Set s3 = CREATE_AS(s1);
   if (s3 == NULL)  return NULL;

   for (i = 0; i < s1->last; i++) {/* No SET_FOREACH */
     Element e = s1->elem[i];
     if (e == NULL) continue;
     if (SetIsIn(s2,e)) {
       if (SetAdd(s3,e) == FAILURE) {     /* clean up */
         SetDestroy(s3);
         return NULL;
       }
     }
   }
   return s3;
}
```

Are s1 and s2 sets of the same kind ?

These lines are going to be duplicated.
Look at set.c (addendum)
for an improved implementation

86

set.c

```
Set SetUnion(Set s1, Set s2)
{  int i;
   Set s3 = CREATE_AS(s1);
   if (s3 == NULL) return NULL;
   for (i = 0; i < s1->last; i++) {
     Element e = s1->elem[i];
     if (e == NULL) continue;
     if (SetAdd(s3,e) == FAILURE) {
       SetDestroy(s3);
       return NULL;
   } }
   for (i = 0; i < s2->last; i++) {
     Element e = s2->elem[i];
     if ((e == NULL) continue;
     if (SetAdd(s3,e) == FAILURE) {
       SetDestroy(s3);
       return NULL;
   } }
   return s3;
}
```

Are s1 and s2 sets of the same kind ?

Yes, they are duplicated.
Look at set.c (addendum)

Copyright 1995 - 2003
C. Gotsman & Y.M. Kimchi
Computer Science Dept. Technion

For personal use only

87

set.c

```
void SetPrint(Set s)
{  Element e;
   int i;

   for (i = 0; i < s->last; i++) { /*DO NOT use SET_FOREACH */
     char *label;
     if ((e = s->elem[i]) == NULL) continue;
     label = s->lm_lbl(e);
     if (label==NULL) {
       printf("ILLEGAL ");
     }else {
       printf("%s ",label);
       free(label);       /* A constraint on s->lm_lbl() */
     }
   }
   printf("\n");
}
```

Look at set.c (addendum) for an improved version

88

set.c (addendum)

A helper function

```
static Set addOrDestroy(Set s, Element e)
{ if (SetAdd(s,e) != FAILURE) return s; /* s is never NULL */
  SetDestroy(s);
  return NULL;
}
```

Yes, the set s still exists

```
Set SetIntersection(Set s1, Set s2)
{ int i;
  Set s3 = CREATE_AS(s1);
  if (s3 == NULL)  return NULL;
  for (i = 0; i < s1->last; i++) { /* NO SET_FOREACH */
    Element e = s1->elem[i];
    if (e != NULL && SetIsIn(s2,e))
      if (!addOrDestroy(s3,e)) return NULL;
  }
  return s3;
}
```

Are s1 and s2 sets of the same kind ?

Copyright 1995 - 2003
C. Gotsman & Y.M. Kimchi
Computer Science Dept. Technion

For personal use only

89

set.c (addendum)

A helper function

```
static Set addToNew(Set s0, Set s1)
{    /*  C99 */
  for (int i = 0; i < s1->last; i++) { /* NO SET_FOREACH */
    Element e = s1->elem[i];
    if (e != NULL && !addOrDestroy(s0,e)) return NULL;
  }
  return s0;
}
```

Yes, the set `s0` still exists

```
Set SetUnion(Set s1, Set s2)
{ Set s3 = CREATE_AS(s1);
  if (s3 != NULL)
    if (!addToNew(s3,s1) || !addToNew(s3,s2)) return NULL;
  return s3;
}
```

Are s1 and s2 sets of the same kind ?

90

set.c (addendum)

```
void SetPrint(Set s) /* Do not use SET_FOREACH */
{ for (int i = 0; i < s->last; i++) { // C99
    Element e = s->elem[i];
    if (e != NULL) {
      char *label = s->lm_lbl(e);
      printf("%s ", label ? label : "ILLEGAL");
      free(label); /*A constraint on s->lm_lbl() */
    }
  }
  printf("\n");
  return;
}
```

`free()` is null-pointer protected

Copyright 1995 - 2003
C. Gotsman & Y.M. Kimchi
Computer Science Dept. Technion

For personal use only