CHAPTER 11

CONSTRUCTED TYPES

WHAT STRUCTURES LOOKS LIKE

	number	name	phone	age
emp	1001	merriweather, tammy\0	818-555-5741\0	26

prstruct.c

(Figure 11.1 – Page 255)

```
/*
 * Display employee information.  Uses:
 *   write_emp - write employee given a structure.
 *   print_emp - write employee given pointer to the structure.
 */
#include <stdio.h>
#include <string.h>

#define MAXNAME   40                      /* longest name */
#define MAXDIGITS 27                      /* longest phone */

struct employee
{
  long number;                            /* employee number */
  char name[MAXNAME];                     /* first and last name */
  char phone[MAXDIGITS];                  /* at least xxx-xxx-xxxx\0 */
  int  age;                               /* age */
};

int main()
{
  void write_emp(struct employee e);
  void print_emp(struct employee *p);
  struct employee emp;
  struct employee other_emp = {1023, "perl, doris", "213-555-6917", 26};

  emp.number = 1001;
  emp.age = 26;
  strcpy(emp.name, "merriweather, tammy");
  strcpy(emp.phone, "818-555-5741");
  write_emp(emp);                        /* write the first employee */
  print_emp(&other_emp);                 /* write the second employee */
  return 0;
}
```

prstruct.c

(Figure 11.1 – Page 255)

```
void write_emp(struct employee e)
{
  printf("Employee: %li\n", e.number);
  printf("Name:     %s\n",  e.name);
  printf("Age:      %i\n",  e.age);
  printf("Phone:    %s\n",  e.phone);
}

void print_emp(struct employee *ep)
{
  printf("Employee: %li\n", ep->number);
  printf("Name:     %s\n",  ep->name);
  printf("Age:      %i\n",  ep->age);
  printf("Phone:    %s\n",  ep->phone);
}
```

date.h

(Figure 11.2 – Page 258)

```
/*
 * Provides DATE type and prototypes for functions that compare DATEs.
 */
typedef struct
{
  int month,
      day,
      year;
} DATE;

int date_equal(const DATE *xptr, const DATE *yptr);
int date_lt(const DATE *xptr, const DATE *yptr);
int date_gt(const DATE *xptr, const DATE *yptr);
```

date.c

(Figure 11.3 – Page 258)

```
/*
 * Functions for comparing dates.
 *   date_equal - are two dates the same?
 *   date_lt - is the first date before the second?
 *   date_gt - is the first date after the second?
 */
#include "date.h"                          /* date type and prototypes */

int date_equal(const DATE *xptr, const DATE *yptr)
{
  return xptr->year  == yptr->year  &&
         xptr->month == yptr->month &&
         xptr->day   == yptr->day;
}

int date_lt(const DATE *xptr, const DATE *yptr)
{
  if (xptr->year < yptr->year)
    return 1;
  if (xptr->year > yptr->year)
    return 0;
  return (xptr->month < yptr->month ||
         (xptr->month == yptr->month && xptr->day < yptr->day));
}

int date_gt(const DATE *xptr, const DATE *yptr)
{
  return !date_lt(xptr, yptr) && !date_equal(xptr, yptr);
}
```

datetest.c

(Figure 11.4 – Page 259)

```
/*
 * A program to test our date comparison routines.  Uses:
 *   cmpdates - compare two dates and print message describing results.
 *   datestr - convert date to string format for output.
 */
#include <stdio.h>
#include "date.h"

#define MONTH_CHARS   2       /* number of chars in a month */
#define DAY_CHARS     2       /* number of chars in a day */
#define YEAR_CHARS    4       /* number of chars in a year */

int main()
{
  void cmpdates(const DATE *ptr1, const DATE *ptr2);
  DATE day1 = {10,4,1965};
  DATE day2 = {10,18,1963};
  DATE day3 = day1;
  DATE day4 = {6,24,1963};

  cmpdates(&day1,&day2);         /* lots of test cases */
  cmpdates(&day1,&day3);
  cmpdates(&day2,&day1);
  cmpdates(&day4,&day2);
  cmpdates(&day2,&day4);
  return 0;
}
```

datetest.c

(Figure 11.4 – Page 259)

```
void cmpdates(const DATE *d1ptr, const DATE *d2ptr)
{
  char *datestr(const DATE *ptr);

  printf("Comparing %s ", datestr(d1ptr));
  printf("with %s: ", datestr(d2ptr));
  if (date_lt(d1ptr,d2ptr))
    printf("it's earlier.\n");
  if (date_gt(d1ptr,d2ptr))
    printf("it's later.\n");
  if (date_equal(d1ptr,d2ptr))
    printf("they're the same.\n");
}

char *datestr(const DATE *dptr)
{
  static char d[MONTH_CHARS + DAY_CHARS + YEAR_CHARS + 3];

  sprintf(d, "%0*i/%0*i/%*i", MONTH_CHARS, dptr->month,
                               DAY_CHARS,   dptr->day,
                               YEAR_CHARS,  dptr->year);
  return d;                    /* ptr to first character of date */
}
```

WHAT A TABLE OF STRUCTURES LOOKS LIKE

1001	merriweather, tammy\0	818-555-5741\0	26
1023	perl, doris\0	213-555-6917\0	26
1033	pham, gisele\0	714-555-2559\0	29
1036	borromeo, daphne\0	818-555-2042\0	22
1039	borromeo, irene\0	818-555-2718\0	26
1048	goebel, diane\0	213-555-4854\0	26
1047	ockert, veronica\0	213-555-4741\0	29

premps.h

(Figure 11.6 – Page 261)

```
/*
 * Define a struct employee.
 */
#define MAXNAME    40
#define MAXDIGITS  27

struct employee              /* same definition as before */
{
  long number;                      /* employee number */
  char name[MAXNAME];               /* first and last name */
  char phone[MAXDIGITS];            /* at least xxx-xxx-xxxx\0 */
  int  age;                         /* age */
};
```

premps.c

(Figure 11.7 – Page 261)

```
/*
 * Print the name field of each element in a structure.
 */
#include <stdio.h>
#include "premps.h"

int main()
{
  void print_emps(struct employee table[], int);
  struct employee emptab[] =
  {
    {1001, "merriweather, tammy", "818-555-5741", 24},
    {1023, "perl, doris",         "213-555-6917", 24},
    {1033, "pham, gisele",        "714-555-2559", 28},
    {1036, "borromeo, daphne",    "818-555-2042", 22},
    {1039, "borromeo, irene",     "213-555-2718", 24},
    {1047, "goebel, diane",       "213-555-4854", 24},
    {1048, "ockert, veronica",    "213-555-4741", 27}
  };

  print_emps(emptab,sizeof(emptab)/sizeof(emptab[0]));
  return 0;
}
```

premps1.c

(Figure 11.8 – Page 262)

```
/*
 * Array printing function, using array subscripting.
 */
#include <stdio.h>
#include "premps.h"                     /* for structure definition */

void print_emps(struct employee table[], int n)
{
  int i;                                /* array index */

  for (i = 0; i < n; i++)
    printf("Name: %s\n", table[i].name);
}
```

premps2.c

(Figure 11.9 – Page 262)

```
/*
 * Better array printing functions, using pointer indexing.
 */
#include <stdio.h>
#include "premps.h"

void print_emps(struct employee *ptr, int n)
{
  struct employee *endptr = ptr + n;

  for (; ptr < endptr; ptr++)
    printf("Name: %s\n", ptr->name);
}
```

wordemp.c

(Figure 11.10 – Page 264)

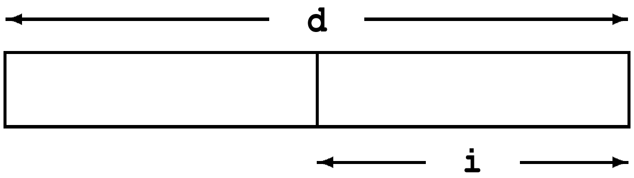
```c
/*
 * New version of our earlier program to pack employee info
 * into a single word.  This time we use bitfields.
 */
#include <stdio.h>

#define SINGLE    0                /* marital status flags */
#define MARRIED   1
#define SEPARATED 2
#define DIVORCED  3
#define MALE      1                /* sex */
#define FEMALE    0

struct emp
{
  unsigned int mstat : 2;          /* marital status */
  unsigned int age   : 7;          /* employee age */
  unsigned int sex   : 1;          /* male or female */
  unsigned int years : 6;          /* years employeed */
};
```

wordemp.c

(Figure 11.10 – Page 264)

```
int main()
{
  unsigned int mstat, sex, age, years;
  struct emp info;                /* holds info on one person */

  printf("male=%i, female=%i\? ", MALE, FEMALE);
  scanf("%i", &sex);
  info.sex = sex;

  printf("age\? ");
  scanf("%i", &age);
  info.age = age;

  printf("single=%i, married=%i, separated=%i, divorced=%i\? ",
         SINGLE, MARRIED, SEPARATED, DIVORCED);
  scanf("%i", &mstat);
  info.mstat = mstat;

  printf("years employed\? ");
  scanf("%i", &years);
  info.years = years;

  printf("Here's the info:\n");
  printf("Sex: %i\n", info.sex);
  printf("Age: %i\n", info.age);
  printf("Marital status: %i\n", info.mstat);
  printf("Years employed: %i\n", info.years);
  return 0;
}
```

WHAT A UNION LOOKS LIKE



Sample Output – union

```
69
1234
-123.345000
98.760000
1078384066
```

union.c

(Figure 11.11 – Page 267)

```
/*
 * Program to test basic union operations.
 */
#include <stdio.h>

union number                          /* define the union type */
{
  int i;
  double d;
};

int main()
{
  union number x = {69};              /* a union and a pointer to it */
  union number *xptr;

  printf("%i\n", x.i);                /* print the initial value */
  x.i = 1234;
  printf("%i\n", x.i);                /* print the int we put in it */
  x.d = -123.345;
  printf("%f\n", x.d);                /* print the double we put in it */
  xptr = &x;
  xptr->d = 98.76;
  printf("%f\n", xptr->d);            /* access it through the pointer */
  x.d = 45.67;
  printf("%i\n", x.i);                /* oops: print the double as int */
  return 0;
}
```

getvalue.h
(Figure 11.12 – Page 268)

```
/*
 * Define types for getvalue.
 */
#define MAXLEN 80              /* longest input line */

#define INTEGER 0
#define REAL    1
#define STRING  2

union value                    /* can hold a long, double, or char */
{
  int    i;
  double d;
  char  *s;
};

union value *getvalue(const char *prompt, int type);
```

getvalue.c

(Figure 11.13 – Page 269)

```
/*
 * Read a value from the user.
 */
#include <stdio.h>
#include <string.h>
#include <stddef.h>
#include <stdlib.h>
#include "getvalue.h"

union value *getvalue(const char *prompt, int type)
{
  int getline(char *buf, int bufsize);
  static union value v;
  char line[MAXLEN + 1];
  int len;

  printf("%s", prompt);
  if ((len=getline(line, MAXLEN)) == -1)
    return NULL;
  switch(type)
  {
    case INTEGER:  if (sscanf(line, "%i", &v.i) != 1)
                     return NULL;
                   break;
    case REAL:     if (sscanf(line, "%lf", &v.d) != 1)
                     return NULL;
                   break;
    case STRING:   if ((v.s = malloc(len + 1)) == NULL)
                     return NULL;
                   strcpy(v.s, line);
                   break;
    default:       return NULL;
  }
  return &v;      /* succeeded */
}
```

useit.c

(Figure 11.14 – Page 270)

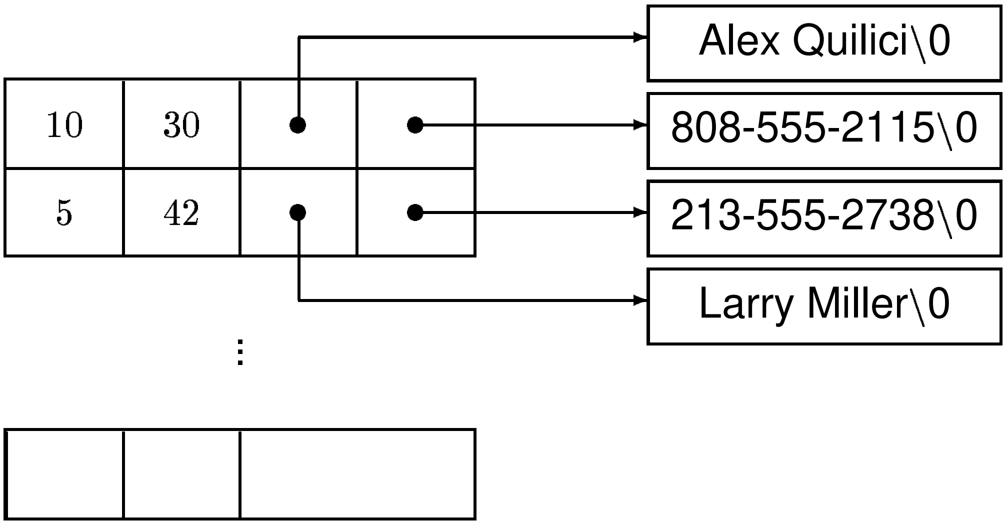
```
/*
 * A program that uses getvalue (quits on first error).
 */
#include <stdio.h>
#include <stddef.h>
#include "getvalue.h"

int main()
{
  union value *uptr;             /* return value from getvalue */
  char        *name;             /* holds name */
  int         age;               /* holds age */
  double      salary;            /* holds salary */
  int         status;            /* did program succeed? */
```

useit.c

(Figure 11.14 – Page 270)

```
  if ((uptr=getvalue("Enter name: ", STRING)) == NULL)
  {
    printf("Couldn't get name\n");
    status = 1;                  /* name failure! */
  }
  else
  {
    name = uptr->s;              /* save name before getting age */
    if ((uptr=getvalue("Enter age: ", INTEGER)) == NULL)
    {
      printf("Couldn't get age\n");
      status = 1;                /* age failure! */
    }
    else
    {
      age = uptr->i;             /* save age before getting salary */
      if ((uptr=getvalue("Enter salary: ", REAL)) == NULL)
      {
        printf("Couldn't get salary\n");
        status = 1;              /* salary failure! */
      }
      else
      {
        salary = uptr->d;
        printf("Name: %s, Age: %i, Salary: %7.2f\n",
                name, age, salary);
        status = 0;
      }
    }
  }
  return status;
}
```

THE NEW EMPLOYEE STRUCTURE



Sample I/O – db

```
Main Menu
  (1) Add
  (2) Delete
  (3) Print
  (0) Quit
Enter choice: 1
Add employee number? 10
Employee name: Alex Quilici
Employee phone: 808-555-2115
Employee age: 30
   ...
Enter choice: 1
Add employee number? 5
Employee name: Larry Miller
Employee phone: 213-555-2738
Employee age: 42
   ...
Enter choice (0 to exit): 1
Add employee number? 29
Employee name: Tony Quilici
Employee phone: 408-555-7764
Employee age: 28
   ...
Enter choice: 2
Delete employee number? 29
   ...
Enter choice: 0

10, Alex Quilici, 30, 808-555-2115
5, Larry Miller, 42, 213-555-2738
2 employees
```

db.h

(Figure 11.17 – Page 273)

```
/*
 * Prototypes for data base functions, externals for table.
 */
#define MAXEMPS  100

typedef struct employee                /* one personnel record */
{
  long number;
  char *name;
  char *phone;
  int  age;
} EMPLOYEE;

extern EMPLOYEE EmployeeTable[];       /* all personnel records */
extern int      Employees;             /* count of employees */

void AddEntry(void);
void DeleteEntry(void);
void PrintEntry(void);
void PrintEntries(void);
```

db.c

(Figure 11.18 – Page 274)

```
/*
 * Program to manage an employee data base.
 */
#include <stdio.h>
#include "db.h"
#include "inp.h"

EMPLOYEE EmployeeTable[MAXEMPS];          /* table of employees */
int      Employees;                       /* count of employees */

int main()
{
  for (;;)
  {
    printf("Main Menu\n"
           "  (1) Add\n"
           "  (2) Delete\n"
           "  (3) Print\n"
           "  (0) Quit\n");
    switch ((int) getlong("Enter choice? "))
    {
      case 0:  PrintEntries();
               return 0;
      case 1:  AddEntry();
               break;
      case 2:  DeleteEntry();
               break;
      case 3:  PrintEntry();
               break;
      default: printf("Bad choice.\n");
    }
  }
}
```

update.c

(Figure 11.19 – Page 275)

```
/*
 * Operations to update the database.
 *   AddEntry - add a new entry to the employee table.
 *   DeleteEntry - deleter an entry from the employee table.
 */
#include <stdio.h>
#include <stdlib.h>
#include "find.h"
#include "inp.h"

void AddEntry(void)
{
  EMPLOYEE *empptr;
  long     newnum;

  if (Employees == MAXEMPS)
    printf("Employee table is full.\n");
  else
  {
    newnum = getlong("Add employee number? ");
    if ((empptr = dont_find_employee(newnum)) == NULL)
    {                              /* add at end of table */
      empptr = &EmployeeTable[Employees++];
      empptr->number = newnum;
      empptr->name   = getstr("Employee name: ");
      empptr->phone  = getstr("Employee phone: ");
      empptr->age    = (int) getlong("Employee age: ");
    }
  }
}
```

update.c

(Figure 11.19 – Page 275)

```
void DeleteEntry(void)
{
  EMPLOYEE *empptr;
  long     empnum;

  if (Employees == 0)
    printf("Employee table is empty.\n");
  else
  {
    empnum = getlong("Delete employee number? ");
    if ((empptr = find_employee(empnum)) != NULL)
    {                              /* replace with last employee */
      free(empptr->name);
      free(empptr->phone);
      *empptr = EmployeeTable[--Employees];
    }
  }
}
```

display.c

(Figure 11.20 – Page 276)

```
/*
 * Functions to display entries in the employee data base.
 *   PrintEntry - print a single table entry.
 *   PrintEntries - print all table entries.
 */
#include <stdio.h>
#include "inp.h"
#include "find.h"

void PrintEntry(void)
{
  const EMPLOYEE *empptr;
  long           empnum;

  if (Employees == 0)
    printf("Employee table is empty.\n");
  else
  {
    empnum = getlong("Print employee number? ");
    if ((empptr = find_employee(empnum)) != NULL)
      printf("Employee #%li\n\tName: %s\n\tAge: %i\n\tPhone: %s\n",
              empptr->number, empptr->name,
              empptr->age, empptr->phone);
  }
}

void PrintEntries(void)
{
  const EMPLOYEE *ptr          = EmployeeTable;
  const EMPLOYEE *const endptr = ptr + Employees;

  for (; ptr < endptr; ptr++)
    printf("%li, %s, %i, %s\n",
            ptr->number, ptr->name, ptr->age, ptr->phone);
  printf("%i employee\n", Employees, Employees == 1 ? "" : "s");
}
```

find.h

(Figure 11.21 – Page 276)

```
/*
 * Prototypes for lookup functions.
 */
#include "db.h"

EMPLOYEE *find_employee(long employee_number);
EMPLOYEE *dont_find_employee(long employee_number);
```

find.c

(Figure 11.22 – Page 277)

```
/*
 * Functions to search employee table.
 *   find_employee - write error message if employee not found.
 *   dont_find_employee - write error message if employee found.
 */
#include <stdio.h>
#include "find.h"

static const EMPLOYEE *lookup(long target)
{
  const EMPLOYEE *ptr          = EmployeeTable;
  const EMPLOYEE *const endptr = ptr + Employees;

  for (; ptr < endptr; ptr++)        /* find employee in table */
    if (ptr->number == target)
      return ptr;
  return (EMPLOYEE *) NULL;
}

EMPLOYEE *find_employee(long employee_number)
{
  const EMPLOYEE *const temp = lookup(employee_number);

  if (temp == NULL)
    printf("There is no such employee.\n");
  return (EMPLOYEE *) temp;
}

EMPLOYEE *dont_find_employee(long employee_number)
{
  const EMPLOYEE *const temp = lookup(employee_number);

  if (temp != NULL)
    printf("There is an employee with that number.\n");
  return (EMPLOYEE *) temp;
}
```

inp.h

(Figure 11.23 – Page 277)

```
/*
 * Prototypes for functions built on top of getvalue.
 */
#include "getval2.h"

long getlong(const char *prompt);
char *getstr(const char *prompt);
```

inp.c
(Figure 11.24 – Page 278)

```
/*
 * Nice interface on top of getvalue.
 */
#include <stdio.h>
#include <stddef.h>
#include "inp.h"

long getlong(const char *prompt)    /* prompt for and read a long */
{
  union value *up;

  while ((up = getvalue(prompt, LONG)) == NULL)
    printf("Bad input.  Expected an integer.\n");
  return up->i;
}

char *getstr(const char *prompt)    /* prompt for and read a string */
{
  union value *up;

  while ((up = getvalue(prompt, STRING)) == NULL)
    printf("Bad input.  Expected a string.\n");
  return up->s;
}
```

getval2.h

(Figure 11.25 – Page 278)

```
/*
 * Define types for getvalue.
 */
#define MAXLEN 80            /* longest input line */

enum  itype {LONG, REAL, STRING};

union value                  /* can hold a long, double, or char */
{
  long   i;
  double d;
  char  *s;
};

union value *getvalue(const char *prompt, enum itype valtype);
```

getval2.c

(Figure 11.26 – Page 279)

```
/*
 * Read a value from the user.
 */
#include <stdio.h>
#include <string.h>
#include <stddef.h>
#include <stdlib.h>
#include "getval2.h"

union value *getvalue(const char *prompt, enum itype type)
{
  int   getline(char *buf, int bufsize);
  union value v;
  char  line[MAXLEN + 1];
  int   len;

  printf("%s", prompt);
  if ((len=getline(line, MAXLEN)) == -1)
    return NULL;
  switch(type)
  {
    case LONG:     if (sscanf(line, "%li", &v.i) != 1)
                     return NULL;
                   break;
    case REAL:     if (sscanf(line, "%lf", &v.d) != 1)
                     return NULL;
                   break;
    case STRING:   if ((v.s = malloc(len + 1)) == NULL)
                     return NULL;
                   strcpy(v.s, line);
                   break;
    default:       return NULL;
  }
  return &v;                                /* succeeded */
}
```

REMINDERS

- Don’t assume the size of a structure is the sum of the size of its fields.
- We’re not allowed to compare structures directly.
- Don’t assume fields are allocated in a particular way.
- Don’t assign to one field of a union and then retrieve from another.

CHAPTER SUMMARY

- Structures allow us to combine data of different types into a single variable.

- Unions allow us to store different types in the same variable location.

- Enumerated types allow us to have a variable whose values fall within a specific list of values, and provide a nice shortcut for defining a set of constants.

- We can have arrays of any of these types, and we can use pointers to traverse them.

- The only operations we're allowed to perform on a structure are accessing its fields, assigning it to another structure, taking its address, and determining its size.

- We can specify fields in a structure to have a particular size in bits, which allows us to lessen our program's need for space.