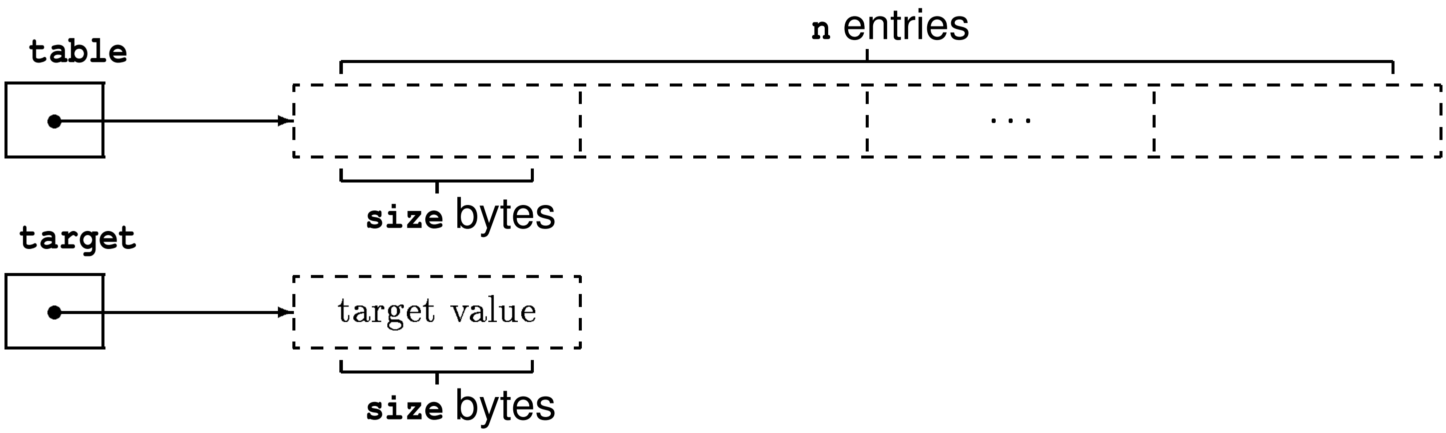
# CHAPTER 16

# GENERIC FUNCTIONS

# BSEARCH'S PARAMETERS

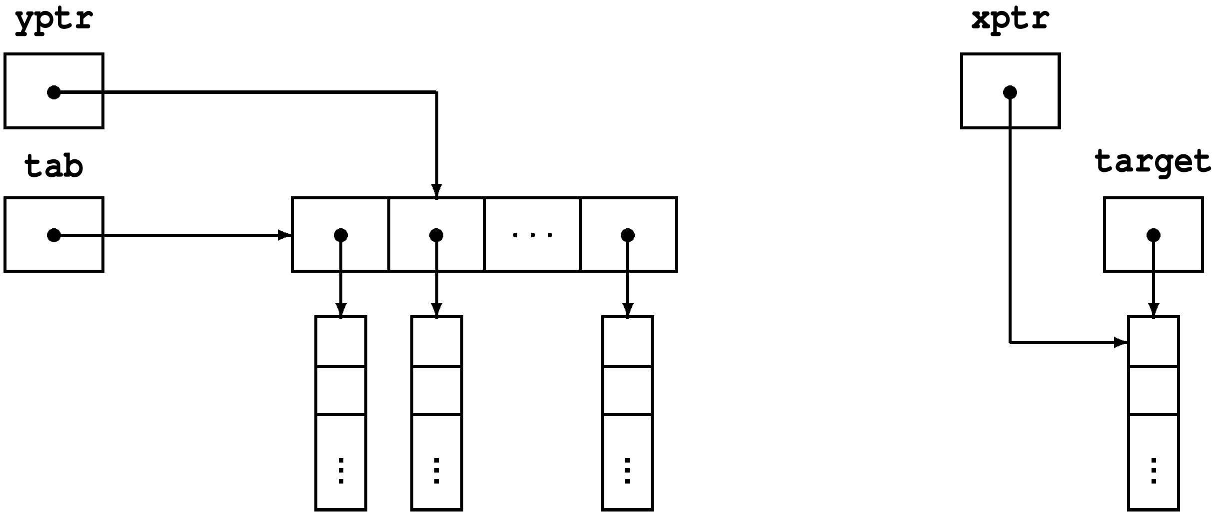


# lbsearch.c

(Figure 16.2 – Page 405)

```
/*
 * Use generic binary search to search a table of longs.
 */
#include <stdio.h>
#include <stddef.h>
#include <stdlib.h>

#define MAX    100

int main(int argc, char *argv[])
{
  int  cmplong(const void *xptr, const void *yptr);
  long tab[MAX];                       /* array to search */
  long *ptr;                           /* ptr to found item */
  int  i;                              /* temporary index */
  long t;                              /* target */

  for (i = 0; i < MAX; i++)            /* initialize table with */
    tab[i] = i;                        /*    0,1,2,3,... */
  for (i = 1; i < argc; i++)           /* are arguments in table? */
    if (sscanf(argv[i], "%li", &t) == 1)
      if ((ptr = bsearch(&t, tab, MAX, sizeof(tab[0]), cmplong)) != NULL)
        printf("Found %li as element %i.\n", t, ptr - tab);
      else
        printf("Didn't find %li.\n", t);
  return 0;
}

/* Compare target long with table entry */

int cmplong(const void *xptr, const void *yptr)
{
  long x = * (long *) xptr;
  long y = * (long *) yptr;

  return (x == y) ? 0 : ((x < y) ? -1 : 1);
}
```

# POINTERS USED TO SEARCH STRING TABLE



# sbsearch.c

(Figure 16.3 – Page 406)

```
/*
 * Uses generic binary search to search an array of strings.
 */
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

int main(int argc, char *argv[])
{
  int   cmpstr(const void *xptr, const void *yptr);
  const char  *tab[] =
              {"alex",   "barbi", "daphne", "diane", "doris",
               "gisele", "irene", "sally", "tammy", "veronica"};
  const int n  = sizeof(tab)/sizeof(tab[0]);
  const char  **ptr;                     /* pointer to found item */
  int   i;

  for (i = 1; i < argc; i++)
    if ((ptr = bsearch(argv[i], tab, n, sizeof(tab[0]), cmpstr)) != NULL)
      printf("Found %s in position %i\n", argv[i], ptr - tab);
    else
      printf("Couldn't find %s in table\n", argv[i]);
  return 0;
}

/* Compare target string with string in table */

int cmpstr(const void *xptr, const void *yptr)
{
  return strcmp((char *) xptr, * (char **) yptr);
}
```

# ebsearch.c

(Figure 16.5 – Page 408)

```
/*
 * Using bsearch to search an array of structures.
 */
#include <stdio.h>
#include <string.h>
#include <stdlib.h>

typedef struct
{
  long number;
  char *name, *phone;
  double salary;
} EMPLOYEE;

int main(int argc, char *argv[])
{
  int empnocmp(const void *xptr, const void *yptr);
  EMPLOYEE tab[] =
            {{1001, "Daphne Borromeo",    "213-555-2134", 8.78},
             {1011, "Tammy Merriweather", "213-555-1212", 4.50},
             {1140, "Doris Perl",         "213-555-1215", 5.60},
             {2045, "Barbara Wong",       "213-555-1219", 7.80},
             {2945, "Diane Goebel",       "213-555-1235", 9.00},
             {3300, "Irene Borromeo",     "213-555-1210", 3.97},
             {4011, "Veronica Ockert",    "213-555-1214", 4.50},
             {4140, "Tony Quilici",       "213-555-1295", 5.60},
             {5045, "Brian Hight",        "213-555-6219", 7.80},
             {5945, "Gisele Pham",        "213-555-1545", 9.00}};
  const int n = sizeof(tab)/sizeof(tab[0]);
  int       i;                       /* index into values to search */
  long      target;                  /* value we're trying to find */
  EMPLOYEE  *matchptr;               /* pointer to found item */
```

# ebsearch.c

(Figure 16.5 – Page 408)

```
  for (i = 1; i < argc; i++)
    if (sscanf(argv[i],"%li",&target) == 1)
    {
      matchptr = bsearch(&target, tab, n, sizeof(tab[0]), empnocmp);
      if (matchptr != NULL)
        printf("%li is in record %i\n", target, matchptr - tab);
      else
        printf("%li wasn't found\n", target);
    }
  return 0;
}

/* Comparison function for employee numbers */

int empnocmp(const void *tarptr, const void *entryptr)
{
  return * (long *) tarptr - ((EMPLOYEE *) entryptr)->number;
}
```

# qsort1.c

(Figure 16.6 – Page 410)

```
/*
 * Using quicksort to sort both an array of longs and an array of
 * character strings.
 */
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

int main()
{
  int cmplong(const void *xptr, const void *yptr);
  int cmpstrs(const void *xptr, const void *yptr);
  long table[] =
         {17, 10, 89, 26, 30, 99, 78, 11, 91, 16,
          87, 45, 78, 11, 45, 98, 67, 20, 60, 8};
  char *names[] =
         {"barbi",  "irene",  "diane", "doris", "veronica",
          "gisele", "daphne", "alex",  "tammy", "sally"};
  const int nlongs = sizeof(table)/sizeof(table[0]);
  const int nnames = sizeof(names)/sizeof(names[0]);
  int i;

  /* sort and print table of longs */
  qsort(table, nlongs, sizeof(table[0]), cmplong);
  for (i = 0; i < nlongs; i++)
    printf("%li\n", table[i]);

  /* sort and print table of strings */
  qsort(names, nnames, sizeof(names[0]), cmpstrs);
  for (i = 0; i < nnames; i++)
    printf("%s\n", names[i]);
  return 0;
}
```

# qsort1.c

(Figure 16.6 – Page 410)

```
/* Compare two table longs */

int cmplong(const void *xptr, const void *yptr)
{
  long x = * (long *) xptr;
  long y = * (long *) yptr;

  return (x == y) ? 0 : ((x < y) ? -1 : 1);
}

/* Compare two table strings */

int cmpstrs(const void *xptr, const void *yptr)
{
  return strcmp(* (char **) xptr, * (char **) yptr);
}
```

# qsort2.c

(Figure 16.7 – Page 411)

```
/*
 * Using qsort to sort an array of structures.
 */
#include <stdio.h>
#include <string.h>
#include <stdlib.h>
#include <stddef.h>

typedef struct
{
  long number;
  char *name, *phone;
  double salary;
} EMPLOYEE;

int main()
{
  int cmpphone(const void *xptr, const void *yptr);
  int cmpname(const void *xptr, const void *yptr);
  EMPLOYEE table[] =
           {{1001, "Daphne Borromeo",    "213-555-2134", 8.78},
            {1011, "Tammy Merriweather", "213-555-1712", 4.50},
            {1140, "Doris Perl",         "213-555-8615", 5.60},
            {2045, "Barbara Wong",       "213-555-3419", 7.80},
            {2945, "Diane Goebel",       "213-555-9035", 9.00},
            {3300, "Irene Borromeo",     "213-555-3410", 3.97},
            {4011, "Veronica Ockert",    "213-555-7814", 4.50},
            {4140, "Tony Quilici",       "213-555-3435", 5.60},
            {5045, "Brian Hight",        "213-555-6219", 7.80},
            {5945, "Gisele Pham",        "213-555-1945", 9.00}};
  const int n = sizeof(table)/sizeof(table[0]);
  int i;
```

# qsort2.c

(Figure 16.7 – Page 411)

```
  qsort(table, n, sizeof(table[0]), cmpphone);  /* sort by phone */
  for (i = 0; i < n; i++)
    printf("%s, %s\n", table[i].name, table[i].phone);
  qsort(table, n, sizeof(table[0]), cmpname);   /* sort by name */
  for (i = 0; i < n; i++)
    printf("%s, %li\n", table[i].name, table[i].number);
  return 0;
}

int cmpphone(const void *xptr, const void *yptr)
{
  return strcmp(((EMPLOYEE *) xptr)->phone, ((EMPLOYEE *) yptr)->phone);
}

int cmpname(const void *xptr, const void *yptr)
{
  return strcmp(((EMPLOYEE *) xptr)->name, ((EMPLOYEE *) yptr)->name);
}
```

# isearch.c

(Figure 16.8 – Page 412)

```
/*
 * Use standard sequential search on an array of integers.
 */
#include <stdio.h>

int main(int argc, char *argv[])
{
  int       *intsearch(const int *ptr, int n, int target);
  const int tab[] = {15, 78, 45, 89, 12, 16, 19, 31, 68, 90};
  const int n     = sizeof(tab)/sizeof(tab[0]);
  const int *ptr;                /* returned pointer to element */
  int       i;                   /* index through arguments */
  int       target;              /* value we're looking for */

  for (i = 1; i < argc; i++)
    if (sscanf(argv[i], "%i", &target) == 1)
      if ((ptr = intsearch(tab, n, target)) != NULL)
        printf("%i in tab[%i]\n", *ptr, ptr - tab);
      else
        printf("Couldn't find %i\n", target);
  return 0;
}

int *intsearch(const int *ptr, int n, int target)
{
  const int *const endptr = ptr + n;

  for (; ptr < endptr; ptr++)
    if (target == *ptr)
      return (int *) ptr;              /* found it */
  return NULL;                         /* failure, return null */
}
```

# search.c

(Figure 16.9 – Page 413)

```
/*
 * Generic sequential search function.
 */
#include <stddef.h>

void *search(const void *tarptr, const void *tabptr,
             int n, size_t size,
             int (*funcptr)(const void *, const void *))
{
  const char *ptr          = tabptr;
  const char *const endptr = ptr + size * n;

  for (; ptr < endptr; ptr += size)
    if (funcptr(tarptr, ptr) == 0)
      return (void *) ptr;     /* found it, return pointer */
  return NULL;                 /* failure, return null */
}
```

# ssearch.c

(Figure 16.10 – Page 414)

```
/*
 * Use generic sequential search to search a table of strings.
 */
#include <stdio.h>
#include <stddef.h>
#include <string.h>

int main(int argc, char *argv[])
{
  int   cmpstr(const void *elptr, const void *tarptr);
  void  *search(const void *tarptr, const void *tabptr,
                int n, size_t size,
                int (*cmpfp)(const void *, const void *));
  const char *stab[] =
                {"gisele", "alex",     "tammy", "irene", "diane",
                 "daphne", "veronica", "doris", "barbi", "sally"};
  const int n = sizeof(stab)/sizeof(stab[0]);
  const char  **ptr;            /* array, ptr to an item */
  int   i;                      /* temporary index */

  for (i = 1; i < argc; i++)
  {
    ptr = search(argv[i], stab, n, sizeof(stab[0]),cmpstr);
    if (ptr != NULL)
      printf("Found %s as element %i.\n", argv[i], ptr - stab);
    else
      printf("Didn't find %s.\n", argv[i]);
  }
  return 0;
}

/* Same comparison function we used for bsearch */

int cmpstr(const void *xptr, const void *yptr)
{
  return strcmp((char *) xptr, * (char **) yptr);
}
```

# isort1.c

(Figure 16.11 – Page 415)

```c
/*
 * Using insertion sort to sort an existing array of ints.
 */
#include <stdio.h>
#include <stddef.h>                /* for size_t */

int main()
{
  void  isort(int vals[], int n);
  int   table[] =
          {19, 56, 48, 99, 1, 87, 67, 14, 95, 34};
  const int n = sizeof(table)/sizeof(table[0]);
  int   i;

  isort(table, n);
  for (i = 0; i < n; i++)
    printf("%i\n", table[i]);
  return 0;
}

void isort(int vals[], int n)
{
  int i, j;
  int temp;

  for (i = 1; i < n; i++)
  {
    temp = vals[i];
    for (j = i; j > 0 && temp < vals[j-1]; j--)
      vals[j] = vals[j-1];
    vals[j] = temp;
  }
}
```

# isort2.c

(Figure 16.12 – Page 416)

```
/*
 * Sort integers using generic insertion sorting routine.
 *   isort - insertion sort an array of any data type.
 *   cmpint - comparison function for integers.
 */
#include <stdio.h>                           /* for printf */
#include <stddef.h>                          /* for size_t */
#include <stdlib.h>                          /* for malloc */
#include <string.h>                          /* for memcpy */

int main()
{
  void isort(void *tab, int n, size_t size,
             int (*cmpfp)(const void *, const void *));
  int  cmpint(const void *xptr, const void *yptr);
  int  table[] =
         {19, 56, 48, 99, 1, 87, 67, 14, 95, 34};
  const int n = sizeof(table)/sizeof(table[0]);
  int  i;

  isort(table, n, sizeof(int), cmpint);
  for (i = 0; i < n; i++)
    printf("%i\n", table[i]);
  return 0;
}
```

# isort2.c
(Figure 16.12 – Page 416)

```
void isort(void *table, int n, size_t size,
           int (*cmpfp)(const void *, const void *))
{
  char *startptr      = table;            /* ptrs to array start */
  char *const endptr = startptr + (n * size);     /* and end */
  void *const temp   = malloc(size);      /* space for 1 element */
  char *pi, *pj;                           /* loop pointers */

  for (pi = startptr + size; pi < endptr; pi += size)
  {
    memcpy(temp, pi, size);                 /* store into temp space */
    for (pj = pi; pj > startptr; pj -= size)
    {
      if (cmpfp(temp, pj - size) >= 0)
        break;
      memcpy(pj, pj - size, size);
    }
    memcpy(pj, temp, size);
  }
}

int cmpint(const void *xptr, const void *yptr)
{
  return * (int *) xptr - * (int *) yptr;
}
```

# GENERIC SEARCH EFFICIENCY

| Percent Time | Total Time (Seconds) | Number of Calls | Time per Call (Microseconds) | Routine |
|---|---|---|---|---|
| 29.6 | 1.47 | 242762 | 0.01 | **memcpy** |
| 27.2 | 2.82 | 1 | 1350.32 | **isort** |
| 21.4 | 3.88 | | | **mcount** † |
| 14.1 | 4.58 | 241749 | 0.00 | **cmpint** |
| 2.8 | 4.72 | | | **monstartup** † |
| 2.2 | 4.83 | 1001 | 0.11 | **doscan** † |
| 1.6 | 4.91 | 1000 | 0.08 | **doprnt** † |

# strgens.h

(Figure 16.13 – Page 419)

```
/*
 * Interface for searching and sorting tables of strings.
 */
#include <stdlib.h>

#define SEARCH_STRINGS(tar,t,n)  bsearch(tar,t,n,sizeof(char *),cmpstr)
#define SORT_STRINGS(t,n)        qsort(t,n,sizeof(char *),cmpstrs)

int cmpstr(const void *xptr, const void *yptr);
int cmpstrs(const void *xptr, const void *yptr);
```

# strgens.c

(Figure 16.14 – Page 419)

```
/*
 * Comparison functions for strings.
 *   cmpstr - compare target string with string in table.
 *   cmpstrs - compare two strings in table.
 */
#include <string.h>

int cmpstr(const void *xptr, const void *yptr)
{
  return strcmp((char *) xptr, * (char **) yptr);
}

int cmpstrs(const void *xptr, const void *yptr)
{
  return strcmp(* (char **) xptr, * (char **) yptr);
}
```

# srchsort.c

(Figure 16.15 – Page 420)

```
/*
 * Table searching and sorting using nice interface.
 */
#include <stdio.h>
#include <stddef.h>
#include "strgens.h"

int main(int argc, char *argv[])
{
  char *tab[] =
          {"gisele", "doris",  "tammy", "tony", "daphne",
           "irene",  "vivian", "barbi", "alex", "veronica"};
  const int n  = sizeof(tab)/sizeof(tab[0]);
  char **ptr;            /* pointer to target's position in table */
  int  i;

  SORT_STRINGS(tab, n);
#if defined(DEBUG)
  for (i = 0; i < n; i++)
    printf("%i: %s\n", i, tab[i]);
#endif
  for (i = 1; i < argc; i++)
    if ((ptr = SEARCH_STRINGS(argv[i], tab, n)) == NULL)
      printf("Search failed.\n");
    else
      printf("Found %s at position %i\n", argv[i], ptr - tab);
  return 0;
}
```

# bsearch.c

(Figure 16.16 – Page 421)

```
/*
 * An implementation of the library binary search.
 */
#include <stddef.h>

static int (*CmpFp)(const void *xptr, const void *yptr);
static int Size;          /* int rather than possibly unsigned size_t */

static void *
binsearch(const char *minptr, const char *maxptr, const void *tptr)
{
  const char *midptr = minptr + ((maxptr - minptr)/(Size * 2)) * Size;
  int cmpres = CmpFp(tptr, midptr);

  if (cmpres == 0)                  /* got it! */
    return (void *) midptr;
  if (maxptr <= minptr)             /* value's not in table? */
    return NULL;
  return (cmpres < 0) ? binsearch(minptr, midptr - Size, tptr)
                      : binsearch(midptr + Size, maxptr, tptr);
}

void *bsearch(const void *tptr, const void *ptr, size_t n, size_t s,
                    int (*fp)(const void *xptr, const void *yptr))
{
  CmpFp = fp;                  /* store in globals to simplify calls */
  Size = (int) s;
  return binsearch(ptr, (char *) ptr + (n - 1) * s, tptr);
}
```

# REMINDERS

- Only use **`bsearch`** to search sorted arrays.

# CHAPTER SUMMARY

- Generic functions are functions that can perform a particular task with any data type.

- C's standard library comes complete with a generic binary search function, **`bsearch`**, and a generic quicksort function, **`qsort`**.

- We can write our generic functions that process arrays by treating them as collections of bytes and carefully doing all of the pointer arithmetic ourselves.

- By using macros, we can write a nice interface on top of generic functions that allows us to forget the painful details of comparison functions and size computations.