Chapter 14

Dynamic Data Structures

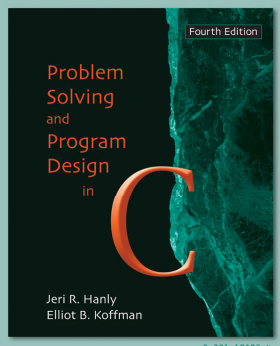
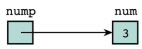


Figure 14.1 Comparison of Pointer and Nonpointer Variables



Reference	Explanation	Value
num	Direct value of num	3
nump	Direct value of nump	Pointer to location containing 3
*nump	Indirect value of nump	3

Figure 14.2 Function with Pointers as Output Parameters

```
1.  #include <stdio.h>
2.
3.  void long_division(int dividend, int divisor, int *quotientp,
4.                     int *remainderp);
5.
6.  int
7.  main(void)
8.  {
9.     int quot, rem;
10.
11.    long_division(40, 3, &quot, &rem);
12.    printf("40 divided by 3 yields quotient %d ", quot);
13.    printf("and remainder %d\n", rem);
14.    return (0);
15. }
16.
17. /*
18.  *  Performs long division of two integers, storing quotient
19.  *  in variable pointed to by quotientp and remainder in
20.  *  variable pointed to by remainderp
21.  */
22. void long_division(int dividend, int divisor, int *quotientp,
23.                    int *remainderp)
24. {
25.    *quotientp = dividend / divisor;
26.    *remainderp = dividend % divisor;
27. }
```


Figure 14.3 Data Area of a Function with Three Pointer-Type Local Variables

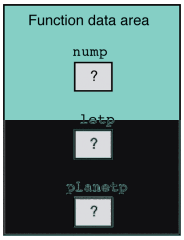


Figure 14.4 Dynamic Allocation of Variables for an int, a char, and a Five-Component planet_t Structure

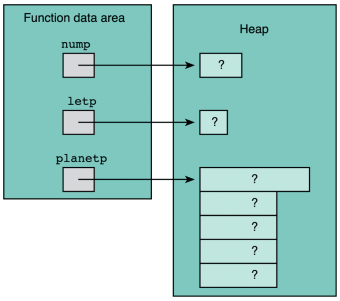


Figure 14.5 Assignment of Values to Dynamically Allocated Variables

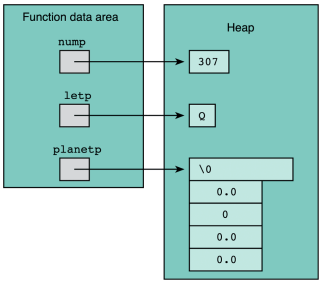


Figure 14.6 Referencing Components of a Dynamically Allocated Structure

```
printf("%s\n", planetp->name);
printf("  Equatorial diameter:  %.0f km\n", planetp->diameter);
printf("  Number of moons:  %d\n", planetp->moons);
printf("  Time to complete one orbit of the sun:  %.2f years\n",
       planetp->orbit_time);
printf("  Time to complete one rotation on axis:  %.4f hours\n",
       planetp->rotation_time);
```


Figure 14.7 Allocation of Arrays with calloc

```
#include <stdlib.h>  /* gives access to calloc */
int scan_planet(planet_t *plnp);

int
main(void)
{
      char     *string1;
      int      *array_of_nums;
      planet_t *array_of_planets;
      int      str_siz, num_nums, num_planets, i;
      printf("Enter string length and string> ");
      scanf("%d", &str_siz);
      string1 = (char *)calloc(str_siz, sizeof (char));
      scanf("%s", string1);

      printf("\nHow many numbers?> ");
      scanf("%d", &num_nums);
      array_of_nums = (int *)calloc(num_nums, sizeof (int));
      array_of_nums[0] = 5;
      for (i = 1;  i < num_nums;  ++i)
            array_of_nums[i] = array_of_nums[i - 1] * i;

      printf("\nEnter number of planets and planet data> ");
      scanf("%d", &num_planets);
      array_of_planets = (planet_t *)calloc(num_planets,
                                            sizeof (planet_t));
```

(continued)

Figure 14.7 Allocation of Arrays with calloc (cont'd)

```
      for (i = 0;  i < num_planets;  ++i)
            scan_planet(&array_of_planets[i]);
      . . .
}

Enter string length and string> 9 enormous

How many numbers?> 4

Enter number of planets and planet data> 2
Earth    12713.5 1  1.0 24.0
Jupiter 142800.0 4 11.9  9.925
```


Figure 14.8 Stack and Heap After Program Fragment in Fig. 14.7

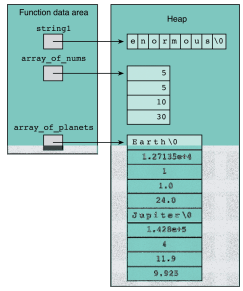


Figure 14.9 Multiple Pointers to a Cell in the Heap

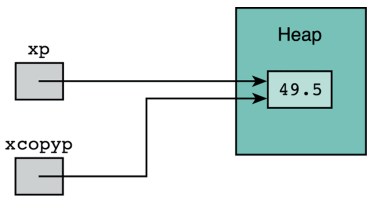


Figure 14.10 Children's Pop Beads in a Chain

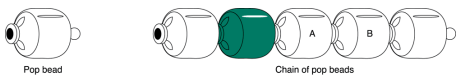


Figure 14.11 Multiple Pointers to the Same Structure

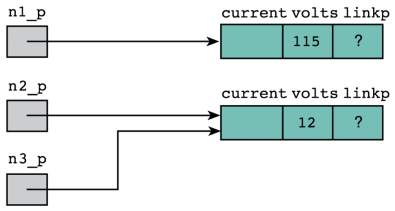


Figure 14.12 Linking Two Nodes

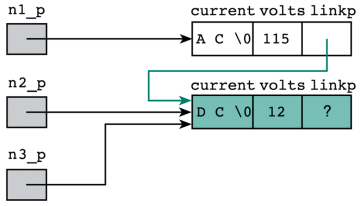


Figure 14.13 Three-Node Linked List with Undefined Final Pointer

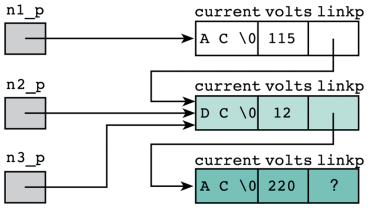


Figure 14.14 Three-Element Linked List Accessed Through n1_p

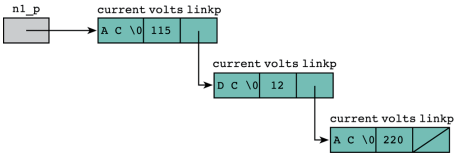


Figure 14.15 Linked List After an Insertion

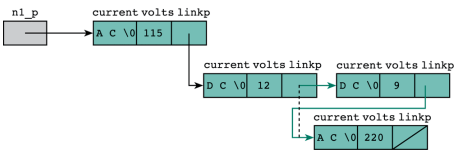


Figure 14.16 Linked List After a Deletion

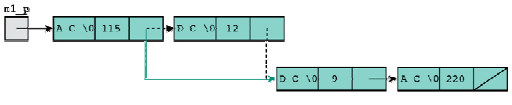


Figure 14.17 Function print_list

```
1.  /*
2.   *  Displays the list pointed to by headp
3.   */
4.  void
5.  print_list(list_node_t *headp)
6.  {
7.        if (headp == NULL) {    /* simple case - an empty list                  */
8.              printf("\n");
9.        } else {                /* recursive step - handles first element       */
10.             printf("%d", headp->digit);  /*      leaves rest to             */
11.             print_list(headp->restp);    /*      recursion                  */
12.       }
13. }
```


Figure 14.18 Comparison of Recursive and Iterative List Printing

```
/* Displays the list pointed to by headp */
void
print_list(list_node_t *headp)
{
 if (headp == NULL) {/* simple case */
    printf("\n");
 } else {          /* recursive step */
    printf("%d", headp->digit);
    print_list(headp->restp);
 }
}
```

```
{ list_node_t *cur_nodep;

  for  (cur_nodep = headp; /* start at
                              beginning  */
        cur_nodep != NULL; /* not at
                              end yet    */
        cur_nodep = cur_nodep->restp)
      printf("%d", cur_nodep->digit);
  printf("\n");
}
```