Reading Assignment #10

Topic: C

Textbook: Problem Solving & Program Design in C
by Jeri R. Hanly and Elliot B. Koffman

Chapter (11, 14)

Structures and Linked Lists
Dynamic Memory allocation
Name: Jupiter
Diameter: 142,800 km
Distance from sun (average): 778.3 million km
Orbital period: 11.9 yr
Axial rotation period: 9.925 hr
Structure Type Definition

SYNTAX:  typedef struct struct_tag {
            type_1 id_list_1;
            type_2 id_list_2;
            .
            .
            .
            type_n id_list_n;
        } struct_type;

EXAMPLE:  typedef struct complex_s { /* complex number structure */
            double real_pt,
                    imag_pt;
        } complex_t;

INTERPRETATION: The identifier struct_type is the name of the structure type being defined. Each id_list_i is a list of one or more component names separated by commas; the data type of each component in id_list_i is specified by type_i. The struct_tag is an identifier that is an optional part of the structure type definition. When the tag is included, the term struct struct_tag can be used interchangeably with the identifier struct_type.

NOTE: type_i can be any standard or previously specified user-defined data type.
#define STRSZ 10

typedef struct planet_s {
    char name [STRSZ];
    double diameter, /* equatorial distance in km */
    dist_sun, /* average distance from sun in km */
    orbital_prd, /* years to orbit sun once */
    axial_rot_prd; /* hours to complete one revolution on
    axis */
} planet_t;

planet_t current_planet,
    previous_planet,
    blank_planet = {"", 0, 0, 0, 0};
...

typedef struct complex_s {
    double real_part;
    double img_part;
} complex_t;

complex_t cn1, cn2;
Variable `blank_planet`, a structure of type `planet_t`:

<table>
<thead>
<tr>
<th>.name</th>
<th>\Q ????????????</th>
</tr>
</thead>
<tbody>
<tr>
<td>.diameter</td>
<td>0.0</td>
</tr>
<tr>
<td>.dist_sun</td>
<td>0.0</td>
</tr>
<tr>
<td>.orbital_prd</td>
<td>0.0</td>
</tr>
<tr>
<td>.axial_rot_prd</td>
<td>0.0</td>
</tr>
</tbody>
</table>

A user-defined type like `planet_t` can be used to declare both simple and array variables and to declare components in other structure types. A structure containing components that are themselves structures is sometimes called a `hierarchical` structure. The following definition of a structure type includes a component that is an array of planets.

```c
typedef struct solar_sys_s {
    double diameter;
    planet_t planets[9];
    char galaxy[STRSIZ];
} solar_sys_t;
```
Manipulating Individual Components of a Structured Data Object

strassign(current_planet.name, "Jupiter", STRSZ);  
current_planet.diameter = 142800;  
current_planet.dist_sun = .7783e+9;  
current_planet.orbital_prd = 11.9;  
current_planet.axial_rot_prd = 9.925;

Variable current_planet, a structure of type planet_t

<table>
<thead>
<tr>
<th>.name</th>
<th>Jupiter \0 ??</th>
</tr>
</thead>
<tbody>
<tr>
<td>.diameter</td>
<td>142800.0</td>
</tr>
<tr>
<td>.dist_sun</td>
<td>.7783e+9</td>
</tr>
<tr>
<td>.orbital_prd</td>
<td>11.9</td>
</tr>
<tr>
<td>.axial_rot_prd</td>
<td>9.925</td>
</tr>
</tbody>
</table>

Assigning Values to Components of Variable current_planet
planet_t   new_planet;
new_planet.dist_sun = 1e + 9;
new_planet.diameter = 180000;

strcpy("Earth", new_planet.name);

planet_t   another_planet;
another_planet.dist_sun = ....
current_planet.name = "Jupiter";
char name[25];

name = "Jupiter";
strcpy("Jupiter", name);
strcpy("Jupiter", current_planet.name);

Suppose we want to print current_planet's name and diameter

printf("%s %f", current_planet.name, current_planet.diameter);
Function with a Structured Input Parameter

/*
 * Prints with labels all components of a planet_t structure
 */

void
print_planet(planet_t p1) /* input – one planet structure */
{
    printf("%s\n", pl.name);
    printf("Equatorial diameter : %.0f km\n", pl.diameter);
    printf("Average distance from the sun: %.4e km\n", pl.dist_sun);
    printf("Time to complete one orbit of the sun : %.2f year\n", pl.orbital_prd);
    printf("Time to complete one rotation of the sun : %.4f hour\n", pl.axial_rot_prd);
}

In calling function :
    print_planet(current_planet);
Just as with strings, we compare structures component by component

```c
if (planet1 == planet2)
    printf(“The planets are equal”);
```

Recall strings

```c
char name1[25];
char name2[25];

if (name1 == name2)
    printf(“The names are the same”);
```
Function Comparing Two Structured Values for Equality

```c
#include <string.h>
/* Determines whether or not the components of planet_1 and planet_2 are equal */
int p_equal(planet_t planet_1, /* input – planets to */
            planet_t planet_2) /* compare */
{
    return ((strcmp(planet_1.name, planet_2.name) == 0) &&
             (planet_1.diameter == planet_2.diameter) &&
             (planet_1.dist_sun == planet_2.dist_sun) &&
             (planet_1.orbital_prd == planet_2.orbital_prd) &&
             (planet_1.axial_rot_prd == planet_2.axial_rot_prd));
}
```

In call function:

```c
if (p_equal(current_planet, blank_planet))
    printf("The planets are equal");
```
void main( )
{
    typedef struct planet_s {
        ;
    } planet_t ;

    planet_t     new_planet ;
    printf( "Enter name, diameter, distance to sun, orbital period and axial rotation period : ") ;
    scanf("%s", new_planet.name ) ;
    scanf("%lf", &new_planet.diameter ) ;
    scanf("%lf", &new_planet.distance );
    scanf("%lf", &new_planet.orbital_prd );
    scanf("%lf", &new_planet.axial_rot_prd ) ;

}
void scan_planet(planet_t *pln)
{
    printf("Enter name, diameter, distance to sun, orbital period and axial rotation period : ");
    scanf("%s", (*pln).name);
    scanf("%lf", &(*pln).diameter);
    scanf("%lf", &(*pln).distance);
    scanf("%lf", &(*pln).orbital_prd);
    scanf("%lf", &(*pln).axial_prd);
}

void main()
{
    typedef struct planet_s {
        
    } planet_t;

    planet_t new_planet;

    scan_planet(&new_planet);
}
Data Areas of main and scan_planet during Execution of 
status = scan_planet (&current_planet);

<table>
<thead>
<tr>
<th>Reference</th>
<th>Type</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>pln;</td>
<td>planet_t *</td>
<td>address of structure that main refers to as new_planet</td>
</tr>
<tr>
<td>*pln</td>
<td>planet_t</td>
<td>structure that main refers to as new_planet</td>
</tr>
<tr>
<td>(*pln).diameter</td>
<td>double</td>
<td>12713.5</td>
</tr>
<tr>
<td>&amp;(*pln).diameter</td>
<td>double *</td>
<td>address of colored component of structure that main refers to as new_planet</td>
</tr>
</tbody>
</table>

Step-by-Step Analysis of Reference &(*pln).diameter
### Precedence and Associativity of Operators Seen So Far

<table>
<thead>
<tr>
<th>Precedence</th>
<th>Symbols</th>
<th>Operator Names</th>
<th>Associativity</th>
</tr>
</thead>
<tbody>
<tr>
<td>highest</td>
<td>a[j] f(...) .</td>
<td>subscripting, function calls, direct component selection</td>
<td>left</td>
</tr>
<tr>
<td></td>
<td>++ --</td>
<td>postfix increment and decrement</td>
<td>left</td>
</tr>
<tr>
<td></td>
<td>++ -- !</td>
<td>prefix increment and decrement, logical not, unary negation and plus, address of, indirect</td>
<td>right</td>
</tr>
<tr>
<td></td>
<td>- + &amp; *</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>( type name )</td>
<td>casts</td>
<td>right</td>
</tr>
<tr>
<td></td>
<td>* / %</td>
<td>multiplicative operators (multiplication, division, remainder)</td>
<td>left</td>
</tr>
<tr>
<td></td>
<td>+ -</td>
<td>binary additive operators (addition and subtraction)</td>
<td>left</td>
</tr>
<tr>
<td></td>
<td>&lt; &gt; &lt;= =&gt;</td>
<td>relational operators</td>
<td>left</td>
</tr>
<tr>
<td></td>
<td>== !=</td>
<td>equality / inequality operators</td>
<td>left</td>
</tr>
<tr>
<td></td>
<td>&amp;&amp;</td>
<td>logical and</td>
<td>left</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>lowest</td>
<td>= += -=</td>
<td>assignment operators</td>
<td>right</td>
</tr>
<tr>
<td></td>
<td>*= /= %=</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Declaring an Array of Structures

#define MAX_STU 50
#define NUM_PTS 10

typedef struct student_s {
    char name[20];
    int id;
    double gpa;
} student_t;

typedef struct point_s {
    double x, y;
} point_t;

...
student_t ece275students[50] ;
strcpy("Johnson",ece275students[0].name );
ece275students[3].id = 0 ;
ece275students[3].gpa = 4.0 ;

complex_t cpoints[100] ;
cpoints[0].real_part =1 ;
cpoints[0].img_part =2 ;

1 + 2j
Dynamic Memory Allocation
int *nump;
int num = 10;
nump = &num;
Storing Data when a Pointer is given:

```c
int *nump;

nump = (int *) malloc(sizeof(int));

*nump = 10;
```
How do we store new data starting with a pointer?
int *nump;
char *letp;
planet_t *planetp;

int num;
nump = &num;
planet_t planet1;
planet_t *planetp;
planetp = &planet1;

malloc(sizeof(int))

nump = (int *)malloc(sizeof(int));
letp = (char *)malloc(sizeof(char));
planetp = (planet_t *)malloc(sizeof(planet_t));
Values may be stored in the newly allocated memory using the indirection operator (*), the same operator we used to follow pointers representing function output parameters. The statements

*\text{nump} = 307;
*\text{letp} = 'Q';
*\text{planetp} = \text{blank\_planet};
Declare a planet pointer and assign this pointer to the values for planet earth.

```c
planet_t  *planet_earth;
planet_earth = ( planet_t *)malloc( sizeof (planet_t)) ;
strcpy("Earth", (*planet_earth).name ) ;
(*planet_earth).dist_sun = 200000 ;
    |  
    |  
(*planet_earth).axial_rot_prd = 1 ;
```
Returning Cells to the Heap

Execution of a call to the function *free* returns memory cells to the heap so they can be reused later in response to calls to *calloc* and *malloc*. For example,

```c
free(lep);
free(planetp);
```

returns to the heap the cell whose address is in *lep*, that is, the cell in which we stored a 'Q'.
Allocation of Arrays with calloc

```c
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 ("n How 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 ("n Enter number of planets and planet data > ");
    scanf ("%d", &num_planets) ;
    array_of_planets = (planet_t *) calloc (num_planets, sizeof (planet_t)) ;

    for (i = 0 ; i < num_planets ; ++ i)
        scan_planets (&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    150000000.0    1.0    24.0
Jupiter  142800.00    778300000.0    11.9    9.925
Accessing a Component of a Dynamically Allocated Structure

```c
planet_t *planetp

(*planetp).name
```

Referencing Components of a Dynamically Allocated Structure

```c
printf("%s\n", planetp->name); // (*planetp).diameter
printf(" Equatorial diameter: %.0f km\n", planetp->diameter);
printf(" Average distance from the sun: %.4e km\n", planetp->dist_sun);
printf(" Time to complete one orbit of the sun: %.2f years\n", planetp->orbital_prd);
printf(" Time to complete one rotation on axis: %.4f hours\n", planetp->axial_rot_prd);
```
Linked List

```c
planet_t  current_planet;
planet_t  previous_planet;
```

```plaintext
headp

previous_planet   current_planet

NULL

name  dist_sun
name  dist_sun
name  dist_sun
```

name  dist_sun

```plaintext
NULL
```
Linked Lists

#define STRSIZ 10

typedef struct planet_s {
    char name[STRSIZ] ;
    double diameter, dist_sun, orbital_prd ;
    struct planet_s *linkp ;
} planet_t;

planet_t *n1_p, *n2_p, *n3_p ;
n1_p = (planet_t *)malloc(sizeof(planet_t));
n1_p->dist_sun = 142800 ;
strcpy (n1_p->name, “Jupiter”);
#define STRSIZ 10

typedef struct planet_s {
    char name[STRSIZ] ;
double   diameter, dist_sun, orbital_prd ;
struct planet_s *linkp ;
} planet_t;

planet_t *n1_p, *n2_p, *n3_p ;
n1_p  = (planet_t *)malloc(sizeof(planet_t));
n1_p->dist_sun = 142800 ;
strcpy (n1_p->name, “Jupiter”);

n2_p  = (planet_t *)malloc(sizeof(planet_t));
strcpy(n2_p->name, “Earth”);

n2_p->dist_sun = 100200;

n1_p         n2_p
linkp          linkp

Jupiter
142800

Earth
100200

linkp
n1_p->linkp = n2_p;

n2_p->linkp = NULL;

"Jupiter"
142800

"Earth"
100200

n1_p

n2_p
n1_p->linkp = n2_p;
n2_p->linkp = NULL;
n3_p = n2_p;
Create a third node with the following values:
Name: Venus  dist_sun: 60000
n2_p->linkp = (planet_t *)malloc(sizeof(planet_t));
strcpy(n2_p->linkp->name, "Venus");
n2_p->linkp->dist_sun = 60000;
n2_p->linkp->linkp = NULL;
Suppose we want to read in a set of data and create a database with linked list. Assume we have the following for planets

<table>
<thead>
<tr>
<th>Name</th>
<th>dist sun</th>
<th>orbital prd</th>
</tr>
</thead>
<tbody>
<tr>
<td>Jupiter</td>
<td>142800</td>
<td>10000</td>
</tr>
<tr>
<td>Earth</td>
<td>100200</td>
<td>20000</td>
</tr>
<tr>
<td>Venus</td>
<td>60000</td>
<td>30000</td>
</tr>
<tr>
<td>Neptune</td>
<td>218100</td>
<td>7500</td>
</tr>
</tbody>
</table>

done

Now create a linked list pointed to by the pointer headp to hold the planet data.
headp

Jupiter
name
142800
dist_sun
10000
orbital_prd
linkp

Earth
name
100200
dist_sun
20000
orbital_prd
linkp

Venus
name
60000
dist_sun
30000
orbital_prd
linkp

Neptune
name
218100
dist_sun
7500
orbital_prd
linkp

NULL
/* Form a linked list pointed to by the pointer headp to hold the planet data */

#define STRSZ 10

typedef struct planet_s{
    char    name[STRSZ] ;
    double    dist_sun, orbital_prd ;
    struct  planet_s    *linkp ;
}planet_p ;

planet_p    *headp ;
headp = getlist( ) ;
# define SENTINEL "done"

```c
planet_t *getlist()
{
    char s[STRSIZ];
    planet_t *ptr;
    scanf("%s",s);
    if (strcmp(s, SENTINEL) == 0)
        ptr = NULL;
    else {
        ptr = (planet_t*)malloc(sizeof(planet_t));
        strcpy(ptr->name, s);
        scanf("%lf", &(ptr->dist_sun));
        scanf("%lf", &(ptr->orbital_prd));
        ptr->linkp = getlist();
    } /* else */
    return(ptr);
} /* getlist */
```
Printing the contents of a list

Display (print) the name of every planet in our linked list given the pointer to beginning of list

```c
void print_list ( planet_t  * nodep)
    /* input is pointer to beginning of list */
{
    if (nodep = = NULL) {  /* an empty list */
        printf("\n") ;
    }
    else {
        printf("%s", nodep->name ) ;
        /* print first element */
        print_list( nodep->linkp) ;
        /* handle the rest of the list */
    }
}
```
How to search a linked list

Suppose we want to search and find the node for a planet given its name (or any other attribute) Assume that the list starts with headp

```c
#define STRSIZ 10

void main()
{

typedef struct planet_s{
    char    name[STRSIZ] ;
    double   dist_sun, orbital_prd ;
    struct  planet_s    *linkp ;
}planet_t ;

char target_name [ ] = "Venus" ;
planet_t    *targetp ;
targetp = search(headp, target_name) ;
```
How to search a linked list (continued)

```c
planet_t *search(planet_t *headp, char target_name[])
{
    planet_t *current_nodep;
    for (current_nodep = headp; ((current_nodep != NULL) &&
        (strcmp(current_nodep -> name, target_name) != 0));
        current_nodep = current_nodep -> linkp);
    return( current_nodep);
}
```
Deleting a node from a linked list

Delete a node given the pointer pointing to that node

```c
void delete_node ( planet_p *linkp )  
    /* input is pointer to node to be deleted */
{
    planet_p  *temp_linkp ; /* temporary pointer */
    if ( linkp != NULL ) {
        temp_linkp = linkp ;
        linkp = linkp->linkp ;
        free(temp_linkp) ;
    }
}
```

```
<table>
<thead>
<tr>
<th>name</th>
<th>diameter</th>
<th>dist_sun</th>
<th>orbital_prd</th>
<th>linkp</th>
</tr>
</thead>
<tbody>
<tr>
<td>&quot;Jupiter&quot;</td>
<td>14280</td>
<td>0</td>
<td></td>
<td>NULL</td>
</tr>
<tr>
<td>&quot;Earth&quot;</td>
<td>100200</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>&quot;Venus&quot;</td>
<td>60000</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
```