Pointers, arrays, and structures

Prof. (Dr.) K.R. Chowdhary, Director SETG
Email: kr.chowdhary@jietjodhpur.ac.in
webpage: http://www.krchowdhary.com

Jodhpur Institute of Engineering and Technology, SETG

September 17, 2015

Pointers

- Pointer is data that holds the address of another memory item
- A pointer itself can store the address of another pointer

  $\texttt{int var1}, \texttt{var2}, *\texttt{ptr};$

  $*\texttt{ptr} = 1234;$

  $\texttt{var1} = *\texttt{ptr};$

  $\texttt{var2} = 1235;$

  $\texttt{ptr} = \&\texttt{var2};$

- The operator & can be legally used only for variables and array elements, but not for compound expressions and constants: $\texttt{ptr} = \&\mathsf{a}[5];$ is valid, and $\texttt{ptr} = \&(\mathsf{a} + \texttt{b});$ is invalid.
- The operator * can only be applied to pointer variables and expressions.

Pointers

```c
/* cptr1.c*/
#include <stdio.h>
int main(){
   int i=18, *ip , **ipp;
   ip = &i;
   ipp=&ip;
   printf("ip = %u, &ip = %u, ipp = %u \n", ip, &ip, ipp);
   printf("&i = %u, i = %d \n", &i, i);
   printf("hello\n");
   return 0;
}
```

Pointers

- A pointer variable may be assigned to a pointer variable of same type
- Pointer variable can be incremented or decremented
- difference between two pointer variable can be obtained by ptr1 - ptr2.

  Array pointer variables:
- $\texttt{ptr} = \&\texttt{arr}[0]$;
- Its contents can be accessed by: $\texttt{arr}[0]$ or $*\texttt{ptr}$
- The $\texttt{arr}[1]$ can be accessed by $*(\texttt{ptr} + 1)$

Pointers

```
/* cptr2.c*/
#include <stdio.h>
char carr[4] = "ABC";
double darr[3] = {1.2, 3.4, 5.6};
int main(){
 char *cptr = &carr[0];
 double *dptr = &darr[0];
 for(; *cptr; cptr++, dptr++) {
    printf("*cptr: %c, cptr: %u ", *cptr, cptr);
    printf("*dptr: %g, dptr: %u\n", *dptr, dptr);
    ;}
 return 0;
}
```

Pointers

Representation or the array “arr” in memory:

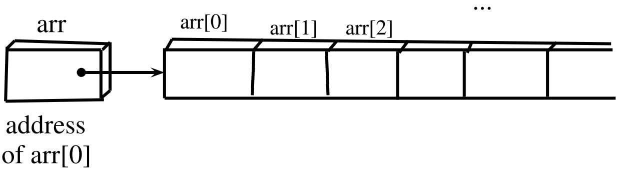

Figure 1: Pointers and arrays

Structures

- Structure in C is a combination of several variables of different types
- structures of C are named by keyword "struct", and behave like records of language PL/1, Ada, Cobol. etc.

```
....
struct date {
      int day;
      char mon_name[3];
      int year;
};  /*type definition*/
....
struct date birthday, exam, wedding;
                    /* variable definition*/
...
```

Structures

You may combine the type and data definition

```
....
struct date {
      int day;
      char mon_name[3];
      int year;
} birthday, exam, wedding;
 ...
 Initialization of structures:
 .....
 struct date birthday = {18, "Jan", 1995};
```

Those structures which are “static” can be initialized.

Structures

Referencing structures:

- .¡element-name¿; The month[0] is first letter of the month.

```
birthday.year
birthday.month[0]
```

- As opposed to arrays, entire structure variables can be transferred to and from functions, as parameters, and function values.

Pointers to structures:

- Useful for creating structures of structures types
- We can also use pointers to access structures types

```
....
struct date *pbirthday; /*structure pointer variable*/
....
```

Structures

```
subsequently, we assign to pointer variable,
the address of structure:
...
pbirthday = &birthday;
.....
elements are eferred by:
...
(*pbirthday).year
....
```

Increment of a structure increases the pointer by the size of the pointer.

Structures

```
 Array of struture:
 ...
 struct date anniversary[] = {
    {8, "MAr", 1980},
    {20, "Aug", 1981},
    {25, "Jul", 1982},
    {2, "Aug", 1984},
    {19, "Sep", 1985} };
....
```

As with array initializations, we ca omit the dimension size, as it can be derived. Following is self referencing structures:

```
 struct list_item {
    char *contents;
    ....
    struct list_item *sucessor;
    };
```

Structures

```
 /*str-p.c*/
#include <stdio.h>
#include <math.h>
int main(){
  struct point {
      int x, y;};
 int ht, len, temp;
 struct point maxpt = {1024, 768};
 struct point pt1 = {50, 40};
 struct point pt2 = {10, 10};
 double dist;
 ht = pt1.y-pt2.y;
 len = pt1.x-pt2.x;
 temp = ht*ht + len*len;
 dist = sqrt((double)temp);
 printf("%d, %d\n", pt1.x, pt1.y);
 printf("%d, %d\n", pt2.x, pt2.y);
 printf("%f\n", dist);
return 0;}
```

Structures

```
 /*str-p2.c passsing structure as a parameter in function call*/
#include <stdio.h>
#include <string.h>

struct student {
      int id;
      char name[20];
      float percent;
};
void func(struct student record);
int main(){
   struct student record;
   record.id = 1;
   strcpy(record.name, "raju");
   record.percent=86.4;

   func(record);
return 0;
}
```

Structures

```
 /*str-p2.c contd */

void func(struct student record) {
     printf("Id is = %d\n", record.id);
     printf("Name is=%s\n", record.name);
     printf("percent is=%f\n", record.percent);
}
```

Structures

```
 /*str-p3.c passing structure as a pointer in function call*/
#include <stdio.h>
#include <string.h>

struct student {
      int id;
      char name[20];
      float percent;
};
void func(struct student *record);
int main(){
   struct student record;
   record.id = 1;
   strcpy(record.name, "raju");
   record.percent=86.4;

   func(&record);
return 0;
}
```

Structures

```
void func(struct student *record) {
     printf("Id is = %d\n", record->id);
     printf("Name is=%s\n", record->name);
     printf("percent is=%f\n", record->percent);
}
```

User defined data structures, using structures

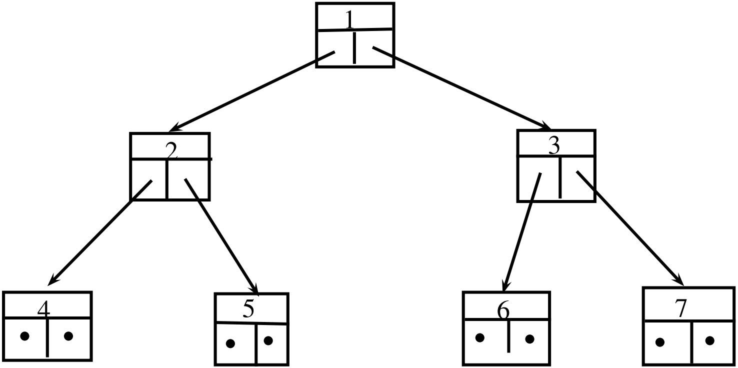

Figure 2: user-defined data structure.

Structures

```
 /*str-p4.c User defined data structures*/
#include <stdio.h>
#include <string.h>

struct student {
      int id;
      struct student *lnode;
      struct student *rnode;
};
int main(){
   struct student *node, *start, *temp;
   node = malloc(sizeof(struct student));
   node->id = 1;
   node->lnode=NULL;
   node->rnode=NULL;
   start=node;
```

Structures

```
 /*str-p4.c User defined data structures contd.*/

   node = malloc(sizeof(struct student));
   node->id = 2;
   node->lnode=NULL;
   node->rnode=NULL;
   start->lnode=node;

   node = malloc(sizeof(struct student));
   node->id = 3;
   node->lnode=NULL;
   node->rnode=NULL;
   start->rnode=node;
   ... continue constructing reaming nodes
   return 0;
}
```

By this we have constructed the root and two nodes, one each for left and right sub-tree.