Structures

Hour 19

Objectives

- Defining and declaring structures
- Accessing structure members or fields
- Structures and pointers
- Structure return values
- Structures containing arrays
- Arrays of structures
- Structures containing pointers

Copyright © 1998-2002 Delroy A. Brinkerhoff. All Rights Reserved.

Structures

User-created aggregate types

- Collection of variables referenced with one name
- A structure declaration is a blueprint, or “cookie cutter” (i.e., does not allocate memory); often put in header files
- struct tag {
  variable declarations;
};
- Taken together, struct tag creates a new type specifier
- Variable declarations may include any type except the struct-type being created
- Each element or variable is called a member or field
- The tag can be any legal identifier

struct Examples

struct variables and struct pointers

struct date {
  int year;
  int mon;
  int day;
};

struct date when;
struct date* datePtr;

Space is not allocated by struct declarations, only by variable definitions
struct declarations often go in header files

Member Access Operators

Accessing the individual data contained in a struct

- If the LHS is a struct object, use the . (the period, pronounced “dot”)
- If the LHS is a pointer to a struct, use the -> (formed by the minus and greater than symbols, pronounced “arrow”)
- Members are legal anywhere a variable of that type is legal (i.e., they can be either l-values or r-values)
- Members are in a unique scope
- Examples
  - when.year = 1998;
  - int mon = when.mon;
  - datePtr->day = 14;
  - int year = datePtr->year;

struct Facts

Assignment, pass, and return copies a “patch of bits”

- Initialization
  struct date today = { 1998, 2, 14 };
- Assignment (bit-by-bit copy)
  when = today;
- Member by member assignment (if you only want some members)
  when.year = today.year;
  when.mon = today.mon;
  when.day = today.day;
- Function arguments are passed by value (bit-by-bit copy)
- Functional return values are returned by value (bit-by-bit copy)

Manipulating structs

Miscellaneous syntax

- Finding the address: &today;
- Finding the size (in bytes): sizeof(struct date);
- Allocating structs dynamically:
  struct date* dateVar;
  dateVar = (struct date*) malloc(sizeof(struct date));
- Deallocating dynamic memory or variables: free(dateVar);
- A common error: memory overflow
  - struct date* dPointer;
  - dPointer->year = 1998;
  - fread(dPointer, sizeof(struct date), 1, fp);
Structure Function Arguments

Passing structs by value and by "reference"

```c
void printDate1(struct date D)
{
 printf("%s\n", D.year);
 printf("%s\n", D.mon);
 printf("%s\n", D.day);
}

struct date newYears = { 1, 1, 1998 };
printDate1(newYears);
```

```c
void printDate2(struct date *D)
{
 printf("%s\n", D->year);
 printf("%s\n", D->mon);
 printf("%s\n", D->day);
}

struct date newYears = { 1, 1, 1998 };
printDate2(&newYears);
```

Structure Return Values

Return by value

```c
struct date getDate(void)
{
 struct date D;
 printf("enter a year: ");
 scanf("%d", &D.year);
 printf("enter a month: ");
 scanf("%d", &D.mon);
 printf("enter a day: ");
 scanf("%d", &D.day);
 return D;
}

struct date now;
now = getDate();
```

Arrays and structs

 structs containing arrays and arrays of structs

```c
struct date {  char year[5];
               char mon[4];
               char day[3];
             };
struct date Important[100];
strcpy(Important[5].year, "1776");
strcpy(Important[5].mon, "Jul");
strcpy(Important[5].day, "4");
if (Important[5].year[0] == '1') ...
```

structs Containing Pointers

The reference data is not part of the struct

```c
struct date Holiday;
struct date temp;
temp = Holiday;
Holiday.year = strdup("1847");
Holiday.mon = strdup("Jul");
Holiday.day = strdup("24");
```

Nested structs

 struct within structs

```c
struct range {
               struct date Start;
               struct date End;
             };
struct date {  int year, mon, day;
               struct date end;  /*error*/
             };
struct ListNode {  char* Name;
               struct ListNode* Next;
             };
```

Alternate Declarations and Definitions

Alternate syntax

```c
#Create a type and define variables:
 struct tag {
               variable declarations;
             } var1, var2;
#Define variables only:
struct {
               variable declarations;
             } var1, var2;
#Create a type that doesn't require the struct keyword:
typedef struct {
               variable declarations;
             } short_name;
```