Lecture10: Structures, Unions and Enumerations
11/26/2012

Slides modified from Yin Lou, Cornell CS2022: Introduction to C

1

Administrative Items

- Assignment #10 on the course webpage

2

Complex Types

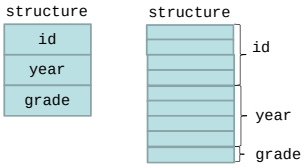
- Structure
  - User-defined combinations of other types
- Enumeration
  - An integer type whose values are named by the programmer
- Union
  - Same data, multiple interpretations

3

Structure

- A structure is a collection of variables of (different) types
  - Defined with the keyword `struct`
- Example

```
struct student {
   int id;
   int year;
   char grade;
};

void main() {
   struct student s;
   s.id = 10001;
   s.year = 2011;
   s.grade = 'A';
}
```



4

Structure Syntax

To define a structure

Definition

```
struct structure_name
{
   /* list of variable
   declarations */
};
```

To declare a variable of the new structure type

Example

```
void main()
{
      struct structure_name
variable_name;
}
```

The struct keyword is required in both places

5

Initializing Structure Variable

```
#define NAME_LEN 20
struct part {
   int number;
   char name[NAME_LEN+1];
   int on_hand;
};

int main() {
   struct part part1 = {528, "Disk drive", 10};
   struct part part2 = {.number=914, .name="Printer cable",
   .on_hand=5};
   struct part part3;
   part3.number = 321;
   strcpy(part3.name, "Computer Display");
   part3.on_hand = 8;
}
```

6

Operations on Structure

```
#define NAME_LEN 20
struct part {
   int number;
   char name[NAME_LEN+1];
   int on_hand;
};
int main() {
   struct part part1 = {528, "Disk drive", 10};
   struct part part2 = {914, "Printer cable", 5};
    struct part temp;

   temp = part1;
   part1 = part2;
   part2 = temp;

   printf("Part number: %d\n", part1.number);
   printf("Part name: %s\n", part1.name);
   printf("Quantiy on hand: %d\n", part1.on_hand);
   return 0;
```

7

Defining Structure Types

```
struct part {
   int number;
   char name[NAME_LEN+1];
   int on_hand;
};

part temp;    /* wrong, part is just the structure name,
                 * not a type */

struct part temp; /* okay */
```

8

Defining Structure Types: typedef

```
typedef struct {
   int number;
   char name[NAME_LEN+1];
   int on_hand;
} Part;             /* now Part becomes the name of
                     * a type. Use it in the same way
                     * as other built-in types */

Part part1, part2;
```

9

Structures as Arguments

```
struct part {
   int number;
   char name[NAME_LEN+1];
   int on_hand;
};

void print_part(struct part p)
{
    printf("Part number: %d\n", p.number);
    printf("Part name: %s\n", p.name);
    printf("Quantiy on hand: %d\n", p.on_hand);
}

int main() {
   struct part part1 = {528, "Disk drive", 10};
   print_part(part1);
   return 0;
}
```

10

Structures as Return Values

```
struct part {
   int number;
   char name[NAME_LEN+1];
   int on_hand;
};

struct part build_part(int number, char *name, int on_hand) {
    struct part p;
    p.number = number;
    strcpy(p.name, name);
    p.on_hand = on_hand;
    return p;
}

int main() {
   struct part part1;
   part1 = build_part(528, "Disk drive", 10);
   return 0;
}
```

11

Nested Structures

```
struct person_name {
    char first[NAME_LEN+1];
    char middle_initial;
    char last[NAME_LEN+1];
};

struct student {
    struct person_name name;
    int id, age;
    char sex;
};
```

12

Array of Structures

```
struct part {
   int number;
   char name[NAME_LEN+1];
   int on_hand;
};

int main() {
   struct part inventory[100];
}
```

13

In-Class Exercise 10.1

Maintain a parts database for a warehouse. the program is built around an array of structures, with each structure containing information, part number, and quantity – about one part. Our program will support the following operations:

- `void insert()`: Ask the user to add information about a new part number, part name, and initial quantity on hand.
- `void print()`: Print all the parts in a database.
- `void search()`: Ask th user to enter a part number. If the part exists, print the name and quantity on hand. If not, print an error message.

Your program ought to have the following input & output (next slide):

14

In-Class Exercise #1

```
Enter operation code: i
Enter part number: 528
Enter part name: Disk drive
Enter quantity on hand: 10

Enter operation code: s
Enter part number: 528
Part name: Disk drive
Quantity on hand: 10

Enter operation code: s
Enter part number: 914
Part not found.

Enter operation code: i
Enter part number: 914
Enter part name: Printer cable
Enter quantity on hand: 5

Enter operation code: p
Part number      Part Name        Quantity on Hand
528              Disk drive       8
914              Printer cable    5

Enter operation code: q
```

15

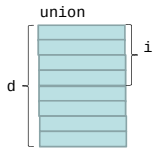
Pointers to Structures

```
typedef struct {
   char name[10000];
   int id;
} person;

void print_person(person *p)
{
   printf("%d %s\n", p->id, p->name);
   /* same as (*p).id and (*p).name */
}

void main()
{
   person a = {"Mike Smith", 1234};
   print_person(&a);
}
```

16

Union

- A union is like struct has many members, possible of different types. Computer allocates only enough space for the largest of its member.
- Example




```
union {
    int i;
    double d;
} u;

u.i = 82;
u.d = 74.8;  // over-writing the value of i
```

17

Why Union? Save space

```
struct catalog_item {
    int stock_number;
    double price;
    int item_type;

    /* books */
    char title[TITLE_LEN+1];
    char author[AUTHOR_LEN+1];
    int num_pages;

    /* mugs & shirts */
    char design[DESIGN_LEN+1];
    int colors;
    int sizes;
}
```

```
struct catalog_item {
    int stock_number;
    double price;
    int item_type;

    union {
        struct {
            char title[TITLE_LEN+1];
            char author[AUTHOR_LEN+1];
            int num_pages;
        } book; /* books */

        struct {
            char design[DESIGN_LEN+1];
        } mug; /* mug */

        struct {
            char design[DESIGN_LEN+1];
            int colors;
            int sizes;
        } shirt; /* shirt */
    }
}
```

18

In-Class Exercise 10.2

```
#include <stdio.h>
#define INT_KIND 0
#define DOUBLE_KIND 1

typedef struct {
    int kind;
    union {
        int i;
        double d;
    } u;
} Number;
```

```
void print_number(Number n)
{
    /* fill your code here */
}

int main() {
       Number n;
       n.kind = INT_KIND;
       n.u.i = 82;
       print_number(n);

       n.kind = DOUBLE_KIND;
       n.u.d = 82.56;
       print_number(n);
}
```

19

Enumerations: basically an int

```
enum days {mon, tue, wed, thu, fri, sat, sun};
/* You can name the values for 0, 1, 2, 3, 4, …
 * make the code easy to read
 * #define mon 0
 * #define tue 1
 * ...
 * #define sun 6
 */

enum days {mon = 3, tue = 8, wed, thu, fri, sat, sun};
/* Same as:
 * #define mon 3
 * #define tue 8
 * #define wed  9
 * ...
 * #define sun 13
 */
```

20

Enumerations

```
enum days day;
// Same as: int day;

for (day = mon; day <= sun; ++day)
{
   if (day == sun)
   {
       printf("Sun\n");
   }
   else
   {
       printf("day = %d\n", day);
   }
}
```

21