Structures

Lecture 4

Yi Fang, Ph.D.
yfang@scu.edu

Structure Basics

- A structure is a collection of values, called *members*, forming a single unit.
- Unlike arrays, the data members can be of different types.

Structure Definition

```
struct name
{
    variable declaration;
    variable declaration;
      .
      .
};
```

- The keyword struct announces that this is a structure definition, which defines a new type.
- No memory is allocated.

Example

```
struct birthday
{
    char    name[20];
    int     day;
    int     month;
    int     year;
};
```

- Defines a structure type.
- The name of the type (struct birthday) is called the structure tag.
- The identifiers declared inside the braces are the members. Members can be declared to be of any valid C data type.
- The tag "struct birthday" may now be used just like any predefined type: int, char, etc.

Declaring Structure Variables

```
struct birthday      real_birthday;
struct birthday      *p;
```

- Variable real_birthday is a collection of variables called member variables
  - Memory is allocated for real_birthday
- The member variables are accessed using the dot (.) operator

```
real_birthday.year = 1993;
real_birthday.month = 3;
real_birthday.day = 10;
```

Pointers to Structures

- When using pointers to access structure members, the arrow operator is used.
- Example:

```
struct birthday *p, real_birthday;
p= &real_birthday;

p->year = 1993;
p->month = 3;
p->day = 10;
```

Declaring Structure Variables

- Shortcuts: Typedef

```
typedef struct birthday
{
    char      name[20];
    int       day;
    int       month;
    int       year;
} BIRTHDAY;

…

BIRTHDAY   real_birthday;
BIRTHDAY   *p;
```

Declaring Structure Variables

- Shortcuts: define

```
#define BIRTHDAY struct birthday
…
struct birthday
{
    char      name[20];
    int       day;
    int       month;
    int       year;
} ;
…
BIRTHDAY  real_birthday;
BIRTHDAY  *p;
```

Initializing Structures

```
struct birthday
{
    char      name[20];
    int       day;
    int       month;
    int       year;
} ;

struct birthday b1 = {"John", 10, 3, 1993};
```

Assignment operator

- Assignment operator is defined for structure of the same type.

```
struct birthday b1, b2;
…
b1.year = 1992;
strcpy (b1.name, “Mary”);
…
/*  Copy all data from b1 to b2.  */
b2 = b1;
```

Scope of a Structure

- Member variables are local to the structure.
- Member names are not known outside the structure.

Arrays of Structures

- Arrays of structures may be declared in the same way as other C data types.

  struct birthday lots_of_birthday[20];

- lots_of_birthday[0] references first structure of lots_of_birthday array.

  lots_of_birthday[0].year = 1986;

Structures as Arguments to Functions

- When a structure is passed as an argument to a function, it is a call-by-value.
  - Changes made to the struct received do not change the argument.
- A pointer to a structure may also be passed as an argument to a function.
  - Changes made to the pointed struct change the argument.

Call by Value - example

Example:

```
struct simple
{
            int         value1;
            int         value2;
};

int main(void)
{
            struct simple s1 = {10, 15};
            fun1 (s1);
            printf("%d %d", s1.value1, s1.value2);
            return 0;
}
void fun1(struct simple s)
{
            s.value1++;
            s.value2 *= 2;
}
```

Call by Reference - example

- Example:

```
struct simple
{
    int    value1;
    int    value2;
};

int main(void)
{
    struct simple s1 = {10, 15};
    fun1 (&s1);
    printf (“%d %d”, s1.value1 , s1.value2);
    return 0;
}

void fun1(struct simple *s)
{
    s->value1++;
    s->value2 *= 2;
}
```

Nested Structures

- Structure definitions may contain data members that are other structures:

- Example:

```
struct more_info
{
    int    c;
    int    d;
};
struct info
{
    int                 a;
    int                 b;
    struct more_info    cd;
};
```

Lab 3

- Change your lab 2 to have an array of structs instead of 2 arrays
- Each struct
  - Name
  - Number