Structures

Lecture 4

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 info
{
   int      number;
   char     character;
   char     string[10];
   int      array[10];
};
```

- Defines a structure type.
- The name of the type (struct info) 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 info" may now be used just like any predefined type: int, char, etc.

Another Example

```
struct node
{
   int          number;
   struct node  *link;
};
```

- Structure tag, new type ==> struct node
- Members ==> number, link

Declaring Structure Variables

```
struct info   real_info;
struct info   *p;
```

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

```
p = &real_info;
real_info.number = 2;
real_info.character = 'A';
```

Pointers to Structures

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

```
struct info *p, real_info;
p= &real_info;

p->number= 2;
p->character = 'B';
p->array[0] = 3;
```

Declaring Structure Variables

- Shortcuts: Typedef

```
typedef struct info
{
   int      number;
   char     character;
   char     string[10];
   int      array[10];
} INFO;

…

INFO     real_info;
INFO     *p;
```

Declaring Structure Variables

- Shortcuts: define

```
#define INFO struct info
…
struct info
{
    int         number;
    char        character;
    char        string[10];
    int         array[10];
};
…
INFO    real_info;
INFO    *p;
```

Initializing Structures

```
struct info
{
    int         number;
    char        character;
    char        string[10];
    int         array[10];
};

struct info i1 = {99, 'Z', "abc", {0, 1, 2, 3, 4, 5, 6, 7, 8, 9}};
```

Practice!

- Write the C statements necessary to output the value of the structure i1 defined in the previous slide.

Assignment operator

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

```
struct info info1, info2;
…
info1.number = 99;
info1.character = 'Z';
strcpy (info1.string, "abc");
for (i = 0; i < 10, i++)
        info1.array[i] = i;
…
/*  Copy all data from info1 to info2.  */
info2 = info1;
```

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 info lots_of_info[20];

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

  lots_of_info[0].number = 17;

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;
}
```

Structures as Return Values

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

int main(void)
{
   struct simple swap (struct simple);

   struct simple s1 = {0, 1}, s2;
   s2 = swap (s1);
   .
   .
```

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 of people