Structures Lecture

CSE 1320

Introduction to Structures

Aggregate Types

Aggregate types are designed to hold multiple data values

Arrays can hold many data values of the same type

```
int GradeArray[10] = {100,99,98,34,89,99,70,99,88,100};
```

Introduction to Structures

Structure

A structure can concurrently hold multiple data values of different types.

```
struct tshirt
{
     char  size[5];
     char  color[10];
     char  design[100];
     char  fittype;
     float price;
     int   inventory_level;
};
```

Introduction to Structures

`struct` is a keyword in C - it signals the declaration of a structure

keyword

user defined

```
struct tshirt
{
        char   size[5];
        char   color[10];
        char   design[100];
        char   fittype;
        float  price;
        int    inventory_level;
};
```

the compiler knows what a `struct tshirt` looks like but has not created one

`struct tshirt` has 6 members

Introduction to Structures

`struct tshirt` is now a user-define type that can be used to declare variables of that type

```
struct tshirt MyTShirts;
struct tshirt YourTShirts;
struct tshirt TheirTShirts;
struct tshirt OurTShirts;
struct tshirt NobodysTShirts;
```

the compiler has now allocated memory for several `struct tshirt` variables

```
struct tshirt
{
  char  size[5];
        color[10];
        design[100];
        fittype;
      t price;
  int   inventory_level;
};
```

```
Breakpoint 1, main () at struct1Demo.c:6
6                int GradeArray[10] =
{100,99,98,34,89,99,70,99,88,100};
(gdb) step
18               struct tshirt YourTShirts = {};

(gdb) ptype GradeArray
type = int [10]

(gdb) ptype YourTShirts
type = struct tshirt {
    char size[5];
    char color[10];
    char design[100];
    char fittype;
    float price;
    int inventory_level;
}
```

int
fundamental type

struct tshirt
user define type

struct1Demo.c

Introduction to Structures

A variable in a structure type can be initialized at the same time that the struct is declared.

```
struct tshirt NobodysTShirts;

struct tshirt YourTShirts = {};

struct tshirt TheirTShirts = {"S"};

struct tshirt OurTShirts = {"","GREEN"};

struct tshirt MyTShirts = {"XS", "BLUE", "DISNEY", 'Y', 14.99, 1987};
```

```
struct tshirt
{
   char  size[5];
   char  color[10];
   char  design[100];
   char  fittype;
   float price;
   int   inventory_level;
};
```

```
struct tshirt NobodysTShirts;
struct tshirt YourTShirts = {};
struct tshirt TheirTShirts = {"S"};
struct tshirt OurTShirts = {" ","GREEN"};
struct tshirt MyTShirts = {"XS", "BLUE", "DISNEY", 'Y', 14.99, 1987};
```

```
18      struct tshirt NobodysTShirts;
(gdb) p NobodysTShirts
$2 = {
  size = "v\000\000\000",
  color = "\000\000\000\000\000\000\000\000\000",
  design = '\000' <repeats 25 times>"\377,
\265\360\000\000\000\000\000\302\000\000\000\000\000\000\000\377\265\360\00
0\000\000\000\000\206\347\377\377\377\177\000\000\207\347\377\377\377\177\0
00\000\000\000\000\000\000\000\000\000\300\313!\311>\000\000\000`\006@\000\
000\000\000\000\203\003@\000\000\000\000\000\001\000",
  fittype = 0 '\000',
  price = 1.79079218e-38,
  inventory_level = 4195991
}
```

```
19      struct tshirt YourTShirts = {};
(gdb) p YourTShirts
$2 = {
  size = "\000\000\000\000",
  color = "\000\000\000\000\000\000\000\000\000",
  design = '\000' <repeats 99 times>,
  fittype = 0 '\000',
  price = 0,
  inventory_level = 0
}
```

```
struct tshirt
{
   char  size[5];
   char  color[10];
   char  design[100];
   char  fittype;
   float price;
   int   inventory_level;
};
```

```
20      struct tshirt TheirTShirts = {"S"};
(gdb) p TheirTShirts
$4 = {
  size = "S\000\000\000",
  color = "\000\000\000\000\000\000\000\000\000",
  design = '\000' <repeats 99 times>,
  fittype = 0 '\000',
  price = 0,
  inventory_level = 0
}
```

```
21    struct tshirt OurTShirts = {"","GREEN"};
(gdb) p OurTShirts
$4 = {
  size = " \000\000\000",
  colo
  desi
  fitt
  pric
  inve
}
```

```
struct tshirt OurTShirts = {,"GREEN"};
[frenchdm@omega ~]$ gcc struct1Demo.c
struct1Demo.c: In function ‘main’:
struct1Demo.c:27: error: expected expression before ‘,’ token
```

```
struct tshirt
{
                e[5];
                or[10];
                ign[100];
                type;
                ce;
                entory_level;
};
```

```
18    struct tshirt MyTShirts = {"XS", "BLUE", "DISNEY", 'Y', 14.99, 1987};
(gdb) p MyTShirts
$1 = {
  size = "XS\000\000",
  color = "BLUE\000\000\000\000\000",
  design = "DISNEY", '\000' <repeats 93 times>,
  fittype = 89 'Y',
  price = 14.9899998,
  inventory_level = 1987
}
```

Introduction to Structures

```
struct tshirt
{
   char  size[5];
   char  color[10];
   char  design[100];
   char  fittype;
   float price;
   int   inventory_level;
};

struct tshirt MyTShirts, YourTShirts;
```

the type `struct tshirt` is created

variables of type `struct tshirt` are declared

```
struct tshirt
{
   char  size[5];
   char  color[10];
   char  design[100];
   char  fittype;
   float price;
   int   inventory_level;
}
TheirTShirts, OurTShirts;
```

`struct tshirt` is declared and two variables are created of that type

Introduction to Structures

```
struct tshirt
{
    char  size[5];
    char  [illegible]10];
    float p[illegible]
    int   inventory_[illegible];
}
MyTShirts, YourTShirts;

struct tshirt NobodysTShirts;
struct tshirt OurTShirts;
```

reusable

```
struct [illegible]
{
    char  size[5];
    char  [illegible]10];
    float p[illegible]
    int   inventory_[illegible];
}
TheirTShirts, OurTShirts;
```

one time use

Cannot create more variables based on this structure because the `struct` was not named; therefore, cannot be reused.

Introduction to Structures

Restrictions on the types of the members of a structure

- a member of a structure cannot be a function
- a structure may not nest a structure of its own type

a member of a structure cannot be a function

```
struct tshirt
{
    int FunctionX(void);
    char  size[5];
    char  color[10];
    char  design[100];
    char  fittype;
    float price;
    int   inventory_level;
};

[frenchdm@omega ~]$ gcc struct1Demo.c -g
struct1Demo.c: In function ‘main’:
struct1Demo.c:15: error: field ‘FunctionX’ declared as a function
```

a structure may not nest a structure of its own type

```
struct tshirt
{
      char  size[5];
      char  color[10];
      char  design[100];
      char  fittype;
      float price;
      int   inventory_level;
      struct tshirt NobodysTShirts;
}
MyTShirts, YourTShirts;

struct2Demo.c: In function 'main':
struct2Demo.c:14: error: field 'NobodysTShirts' has incomplete type
```

Compiler error because `struct tshirt` does not exist as a type since it is being declared here

Introduction to Structures

```
struct tshirt
{
      char   size[5];
      char   color[10];
      char   design[100];
      char   fittype;
      float  price;
      int    inventory_level;
}
MyTShirts, YourTShirts;
```

MyTShirts has already been created so OK to use here

```
struct
{
      char   size[5];
      char   color[10];
      char   design[100];
      char   fittype;
      float  price;
      int    inventory_level;
      struct tshirt MyTShirts;
TheirTShirts, OurTShirts;
```

C:\Users\Donna\Desktop\UTA\Programs\CSE1320\struct5Demo.c - Notepad++

File Edit Search View Encoding Language Settings Tools Macro Run Plugins Window ?

struct4Demo.c | struct1Demo.c | struct2Demo.c | struct3Demo.c | struct5Demo.c

```c
#include <stdio.h>

int main(void)
{

    struct tshirt
    {
        char  size[5];
        char  color[10] = "blue";
        char  design[100];
        char  fittype = 'Y';
        float price = 11.99;
        int   inventory_level = 100;
    } MyTShirts, YourTShirts;

    struct tshirt hello;

    struct
    {
        char  size[5];
        char  color[10];
        char  design[100];
        char  fittype;
        float price;
        int   inventory_level;
        struct tshirt MyTShirts;
    } TheirTShirts, OurTShirts;

    return 0;
}
```

C source file | length : 478 lines : 31 | Ln : 8 Col : 23 Sel : 0 | 0 | Windows (CR LF) | UTF-8 | INS

Type here to search

12:54 AM
3/3/2019

```
struct tshirt
{
   char  size[5];
   char  color[10];
   char  design[100];
   char  fittype;
   float price;
   int   inventory_level;
};
```

```
struct tshirt MyTShirts = {"XS", "BLUE", "DISNEY", 'A', 11.10, 20};
```

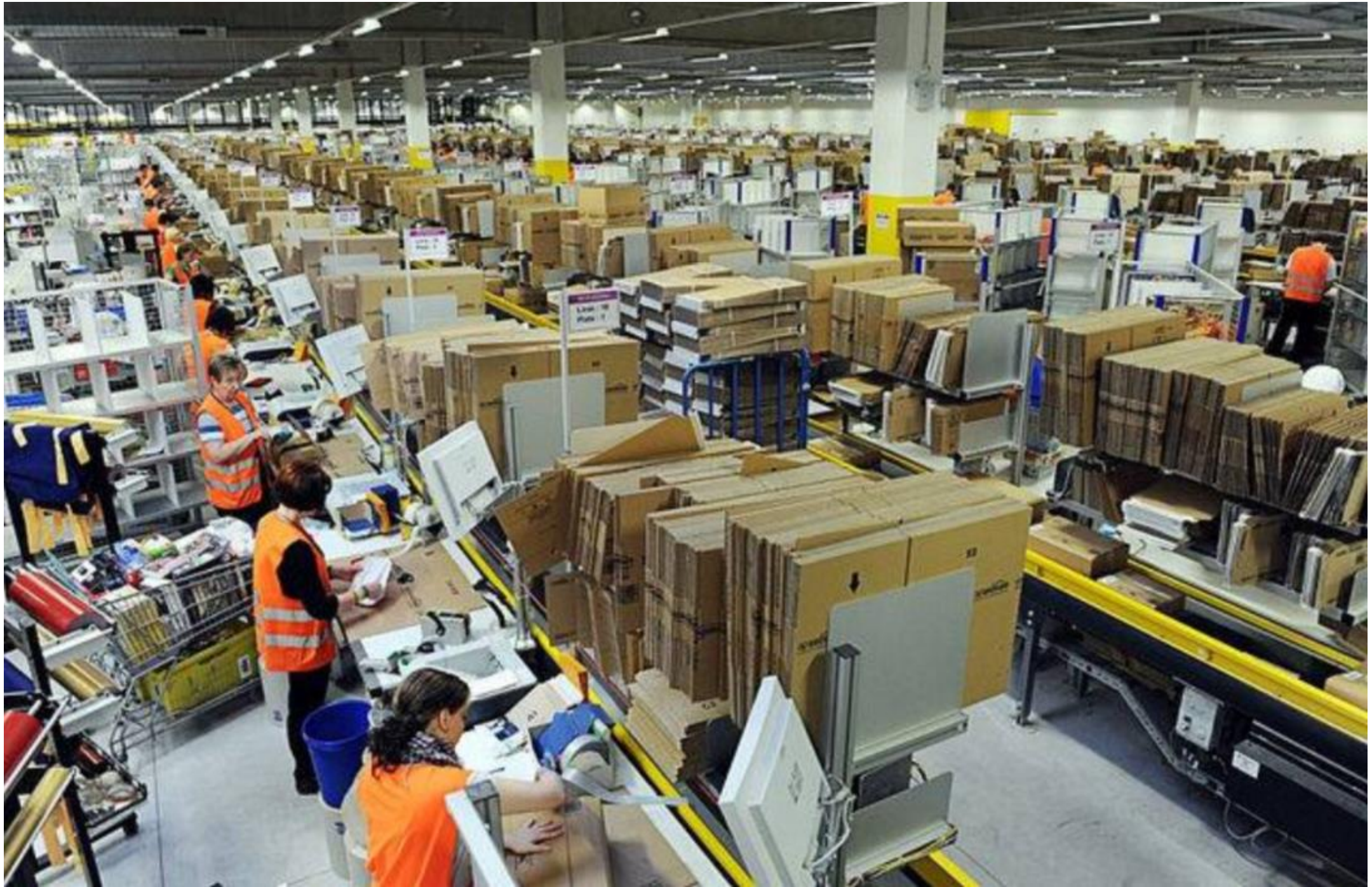
1-A

```
struct tshirt
{
    char  size[5] = "XS";
    char  color[10] = "BLUE";
    char  design[100] "DISNEY";
    char  fittype = 'A';
    float price = 11.10;
    int   inventory_level = 20;
};
```

1-B

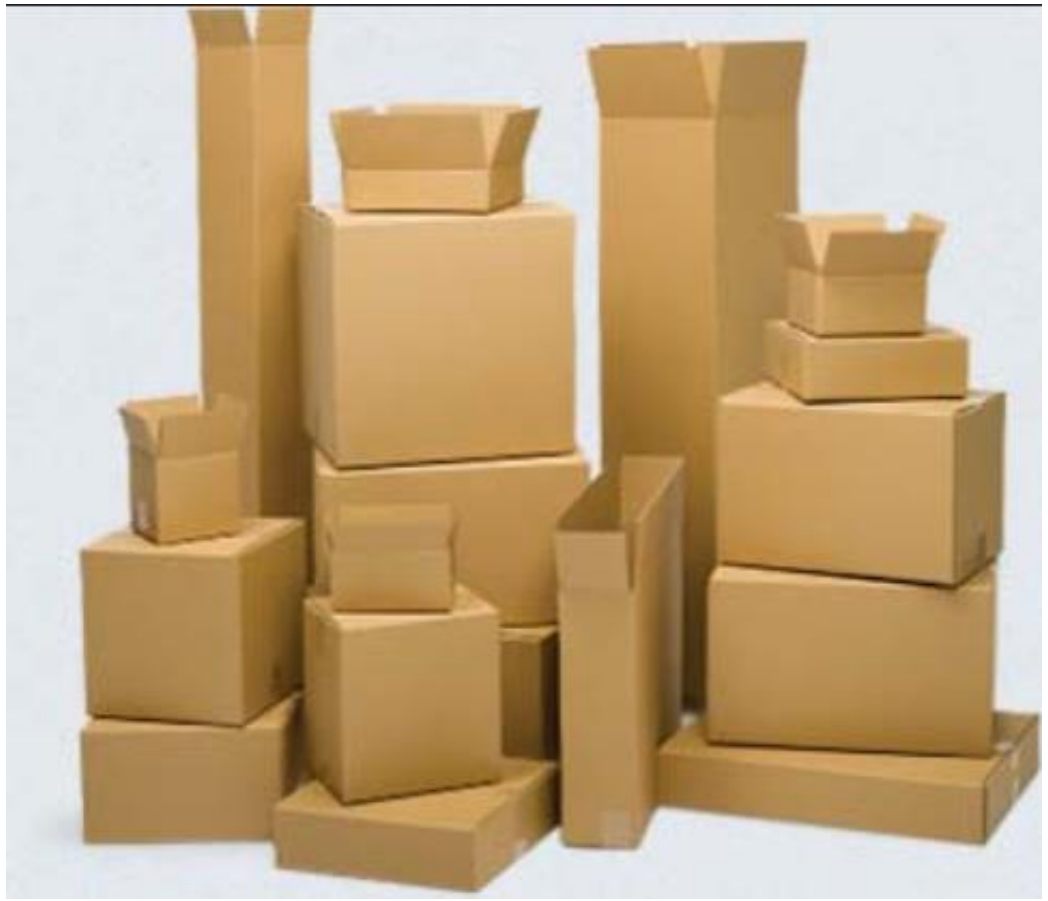
```
struct tshirt
{
    char  size[5];
    char  color[10];
    char  design[100];
    char  fittype;
    float price;
    int   inventory_level;
};
```

```
struct tshirt MyTShirts = {"XS", "BLUE", "DISNEY", "A", 11.10, 20};

struct tshirt MyTShirts = {"XS", "BLUE", "DISNEY", 'A', 11.10, 20};

struct tshirt MyTShirts = {'XS', 'BLUE', 'DISNEY', 'A', 11.10, 20};

struct tshirt MyTShirts = {"XS", "BLUE", "DISNEY", 'A', "11.10", "20"};
```



Introduction to Structures

Build a structure for a box



```
struct box
{
  int height;
  int length;
  int depth;
  float weight;
  char size[3]; // XS,S,M,L,XL
  char strength[10]; // how heavy duty
  int code; // USPS assigns codes
  int inventory_level;
};
```

Introduction to Structures

The individual fields of a structure can be accessed with this syntax

```
variable_name.member_name
```

```
struct tshirt
{
  char  size[5];                    MyTShirts.size
  char  color[10];                  MyTShirts.color
  char  design[100];                MyTShirts.design
  char  fittype;                    MyTShirts.fittype
  float price;                      MyTShirts.price
  int   inventory_level;            MyTShirts.inventory_level
};

struct tshirt MyTShirts;
```

```c
printf("What size is your tshirt? ");
scanf("%s", MyTShirts.size);

printf("What color is your tshirt? ");
scanf("%s", MyTShirts.color);

printf("What design is your tshirt? ");
scanf("%s", MyTShirts.design);

printf("What fit type is your tshirt? ");
scanf(" %c", &MyTShirts.fittype);

printf("What is the price of your tshirt? ");
scanf("%f", &MyTShirts.price);

printf("How many do you have in stock? ");
scanf("%d", &MyTShirts.inventory_level);

printf("Tshirt size      : %s\n", MyTShirts.size);
printf("Tshirt color     : %s\n", MyTShirts.color);
printf("Tshirt design    : %s\n", MyTShirts.design);
printf("Tshirt fit type  : %c\n", MyTShirts.fittype);
printf("Tshirt price     : %.2f\n", MyTShirts.price);
printf("Tshirt inventory : %d\n", MyTShirts.inventory_level);
```

```
Tshirt size      : M
Tshirt color     : RED
Tshirt design    : MARVEL
Tshirt fit type  : Y
Tshirt price     : 12.99
Tshirt inventory : 100
```

struct3Demo.c

Operations on Structures

Very few operations may operate on a structure as a whole.

The following operations are allowed.

1. The selection operators access a single member from the structure
2. The assignment operator assigns the contents of one structure variable to another.
3. The address operator, &, can be used with a structure variable in most interfaces
4. The `sizeof()` operator is usually defined for structures

Operations on Structures

The selection operators access a single member from the structure

```
scanf("%d", &MyTShirts.inventory);
printf("Tshirt inventory : %d\n", YourTShirts.inventory);
```

The assignment operator assigns the contents of one structure variable to another.

```
YourTShirts = MyTShirts;
```

The address operator, `&`, can be used with a structure variable in most interfaces

```
printf("The address of MyTShirts   is %p\n", &MyTShirts);
printf("The address of YourTShirts is %p\n", &YourTShirts);
```

The `sizeof()` operator is usually defined for structures

```
printf("\n\nThe sizeof() MyTShirts is %d", sizeof(MyTShirts));
printf("The sizeof() MyTShirts.size is %d\n", sizeof(MyTShirts.size));
```

Using Structures with Arrays and Pointers

Arrays of Structures

```
struct tshirt MarvelTShirts[10];
struct tshirt DisneyTShirts[15];
struct tshirt DCComicsTShirts[5];
```

Each cell of the array is a structure

```
struct tshirt
{
        char  size[5];
        char  color[10];
        char  design[100];
        char  fittype;
        float price;
        int   inventory;
};
```

Using Structures with Arrays and Pointers

Arrays of Structures

```
struct tshirt MarvelTShirts[10];
```

	size	color	design	fittype	price	inventory
MarvelTShirts[0]						
MarvelTShirts[1]						
MarvelTShirts[2]						
MarvelTShirts[3]						
MarvelTShirts[4]						
MarvelTShirts[5]						
MarvelTShirts[6]						
MarvelTShirts[7]						
MarvelTShirts[8]						
MarvelTShirts[9]						

Arrays of Structures

	size	color	design	fittype	price	inventory
MarvelTShirts[0]						
MarvelTShirts[1]						
MarvelTShirts[2]						
MarvelTShirts[3]						
MarvelTShirts[4]						
MarvelTShirts[5]						
MarvelTShirts[6]						
MarvelTShirts[7]						
MarvelTShirts[8]						
MarvelTShirts[9]						

```
MarvelTShirts[0].color
MarvelTShirts[5].fittype
MarvelTShirts[6].size
MarvelTShirts[9].inventory
```

Arrays of Structures

Arrays of structures can be initialize by nesting the initial values for each structure as list elements in the braces enclosing the initial values for the array.

```
struct tshirt MarvelTShirts[10] = {};
struct tshirt DisneyTShirts[15] = {{"XS"},
                                   {"S"},
                                   {"M"},
                                   {"L"},
                                   {"XL"}
                                  };
struct tshirt DCComicsTShirts[5] = {{"XS","BLACK","BATMAN",'Y',12.99,198},
                                    {"S","BLUE","SUPERMAN",'M',24.99,34},
                                    {"M","RED","WONDER WOMAN",'W',27.99,87},
                                    {"L","YELLOW","AQUAMAN",'M',26.99,65},
                                    {"XL","GREEN","GREEN LANTERN",'Y',15.99,81}
                                   };
```

structarray1Demo.c

Arrays of Structures

Individual elements in an array inside the structure can be accessed the same way as regular arrays.

```
MarvelTShirts[5].color[0] = 'R';
MarvelTShirts[5].color[1] = 'E';
MarvelTShirts[5].color[2] = 'D';
MarvelTShirts[5].fittype = 'Y';
MarvelTShirts[5].inventory = 123;

printf("%s\n", MarvelTShirts[5].color);
printf("%c\n", MarvelTShirts[5].fittype);
printf("%d\n", MarvelTShirts[5].inventory);
```

```
RED
Y
123
```

structarray1Demo.c

Pointers to Structures

In C, it is possible to declare a pointer to any type

This includes pointers to structures.

```
struct tshirt MyTShirts = {"M","BLUE","DISNEY','W',29.99,1};
struct tshirt *tshirtptr;
tshirtptr = &MyTShirts;

printf("MyTShirts.design\t%s\n",MyTShirts.design);
printf("(*tshirtptr).design\t%s\n\n", (*tshirtptr).design);

MyTShirts.design          DISNEY
(*tshirtptr).design       DISNEY
```

Pointers to Structures

In C, it is possible to declare a pointer to any type

This includes pointers to structures in arrays.

```
struct tshirt DCComicsTShirts[5] = {{"XS","BLACK","BATMAN",'Y',12.99,198},
                                    {"S","BLUE","SUPERMAN",'M',24.99,34},
                                    {"M","RED","WONDER WOMAN",'W',27.99,87},
                                    {"L","YELLOW","AQUAMAN",'M',26.99,65},
                                    {"XL","GREEN","GREEN LANTERN",'Y',15.99,81}
                                    };

struct tshirt *tshirtarrayptr;
tshirtarrayptr = &DCComicsTShirts[3];


printf("DCComicsTShirts[3].design\t%s\n", DCComicsTShirts[3].design);
printf("*(tshirtarrayptr).design\t%s\n", (*tshirtarrayptr).design);

DCComicsTShirts[3].design        AQUAMAN
(*tshirtarrayptr).design         AQUAMAN
```

Pointers to Structures

The () are necessary because the dot selector has precedence over the dereferencing operator *

```
printf("tshirtptr design\t%s\n\n", (*tshirtptr).design);
printf("tshirtarrayptr design\t%s\n", (*tshirtarrayptr).design);
```

Without the (), the compiler complains

```
printf("tshirtptr design\t%s\n\n", *tshirtptr.design);
```

```
error: request for member ‘design’ in something not a structure or union
```

Pointers to Structures

The concept of a pointer to structure is used so often in C that a special syntax was developed to reference the members of the target structure.

`(*struct_pointer).member` can be written as `struct_pointer->member`

```
printf("tshirtptr design\t%s\n\n", (*tshirtptr).design);
printf("tshirtptr design\t\t%s\n", tshirtptr->design);

printf("tshirtarrayptr design\t%s\n", (*tshirtarrayptr).design);
printf("tshirtarrayptr design\t%s\n", tshirtarrayptr->design);
```

Passing Structures to and from Functions

Pointers to structures are also used to make structures available to functions.

When a pointer to a structure is passed to a function, the function can access the information in the structure and can modify the information.

```
struct tshirt *tshirtptr = &MyTShirts;
struct tshirt *tshirtarrayptr = &DCComicsTShirts[3];

UpdateInventory(tshirtarrayptr);
```

structpointer2Demo.c

```
struct tshirt MyTShirts = {};

struct tshirt MarvelTShirts[10] = {};

struct tshirt DisneyTShirts[15] = { {"XS"},
                  {"S"},
                  {"M"},
                  {"L"},
                  {"XL"}
                  };

struct tshirt DCComicsTShirts[5] = {{"XS","BLACK","BATMAN",'Y',12.99,198},
                  {"S","BLUE","SUPERMAN",'M',24.99,34},
                  {"M","RED","WONDER WOMAN",'W',27.99,87},
                  {"L","YELLOW","AQUAMAN",'M',26.99,65},
                  {"XL","GREEN","GREEN LANTERN",'Y',15.99,81}
                   };

struct tshirt *tshirtptr = &MyTShirts;

struct tshirt *tshirtarrayptr = &DCComicsTShirts[3];
```

tshirtptr

tshirtarrayptr

structpointer2Demo.c

Function call passing the pointer to the structure

```
struct tshirt *tshirtarrayptr = &DCComicsTShirts[3];

UpdateInventory(tshirtarrayptr);
```

Function receiving the pointer to the structure

```
void UpdateInventory(struct tshirt *TShirtPointer)
{
   printf("Enter the new inventory level for the %s design TShirt ",
           TShirtPointer->design);
   scanf("%d", &TShirtPointer->inventory);

   return;
}
```

structpointer2Demo.c