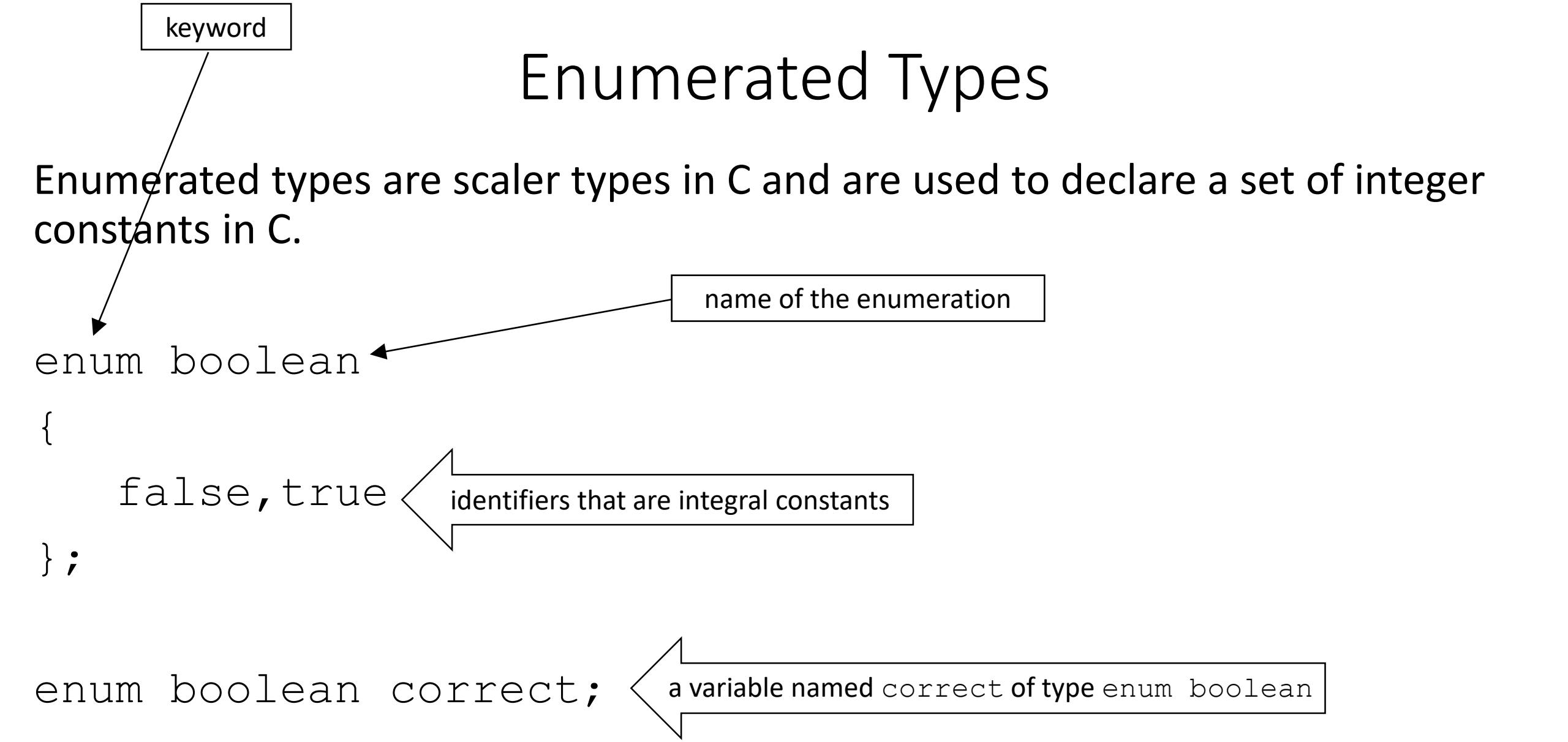
CSE 1320

Week of 3/04/2019

Instructor : Donna French

Enumerated Types

Enumerated types are scaler types in C and are used to declare a set of integer constants in C.



Enumerated Types

```
enum colors
{
     red,orange,yellow,blue,green,indigo,violet
};

enum colors rainbow;

enum colors
{
     red,orange,yellow,blue,green,indigo,violet
}
rainbow;
```

Enumerated Types

```
enum colors
{
      red,orange,yellow,blue,green,indigo,violet
}
rainbow;
```

The default values are assigned starting with 0 and each succeeding identifier is assigned successive integer values.

red	0	green	4
orange	1	indigo	5
yellow	2	violet	6
blue	3		

Enumerated Types

The default values for the identifiers in an `enum` type can be overridden.

```
enum colors
{
      red=3,orange=6,yellow=6,blue=4,green=5,indigo,violet
}
rainbow;
```

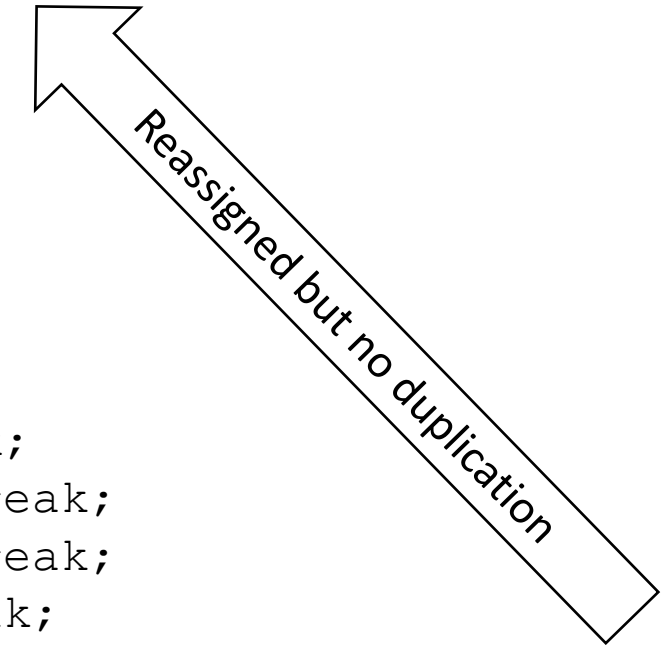
- More than one identifier can be assigned the same value – `orange` and `yellow` are both 6
- `indigo` will be set to 6 since it appears in the list after `green` which was assigned 5.
- `violet` will be set to 7 since it appears in the list after `indigo` which was assigned 6

```
switch (ColorNumber)
{
   case red    : printf("red\n");
   case orange : printf("orange\n");
   case yellow : printf("yellow\n");
   case blue   : printf("blue\n");
   case green  : printf("green\n");
   case indigo : printf("indigo\n");
   case violet : printf("violet\n");
   default : printf("You have fallen off the rainbow\n");
}
```

```
enum colors
{
   red=3,
   orange=6,
   yellow=6,
   blue=4,
   green=5,
   indigo,
   violet
}
rainbow;
```

```
[frenchdm@omega ~]$ gcc enum1Demo.c
enum1Demo.c: In function ‘main’:
enum1Demo.c:21: error: duplicate case value
enum1Demo.c:20: error: previously used here
enum1Demo.c:24: error: duplicate case value
enum1Demo.c:20: error: previously used here
```

```
enum colors
{
	red=3,orange=2,yellow=6,blue=4,green=15,indigo,violet
}
rainbow;

printf("Enter a number between 0 and 7 ");
scanf("%d", &ColorNumber);

switch (ColorNumber)
{
	case red    : printf("red\n"); break;
	case orange : printf("orange\n"); break;
	case yellow : printf("yellow\n"); break;
	case blue   : printf("blue\n"); break;
	case green  : printf("green\n"); break;
	case indigo : printf("indigo\n"); break;
	case violet : printf("violet\n"); break;
	default : printf("You have fallen off the rainbow\n");
}
```



enum2Demo.c

Enumerated Types

An `enum` variable is legally supposed to accept only the values defined in it but compilers are not required to check that assigned value is in the declared list.

Operations with `enum` types are limited. The identifiers are treated as constants of type `int` and can appear anywhere that an `int` constant can.

```
enum colors
{
   red,orange,yellow,blue,green,indigo,violet
};

enum colors color1;
enum colors color2;
enum colors color3;

color1 = orange;
color2 = green;
color3 = red+orange+yellow+blue+green+indigo+violet;

printf("color1 = %d\ncolor2 = %d\ncolor3 = %d\n", color1, color2, color3);
```

Enumerated Types

Other than assignment and equality tests, no other operations must be supported by an ANSI C compiler. Other operations may be supported but their use is not encouraged due to portability issues.

Why use them?

- Improve readability
- Improve maintainability
- Automatic assignment and accounting of values

```
enum DayofWeek
{
      Sunday, Monday, Tuesday, Wednesday, Thursday, Friday, Saturday
};

int Today;

printf("Enter the day of the week ");
scanf("%d", &Today);

switch (Today)
{
      case Sunday    : printf("Today is Sunday\n"); break;
      case Monday    : printf("Today is Monday\n"); break;
      case Tuesday   : printf("Today is Tuesday\n"); break;
      case Wednesday : printf("Today is Wednesday\n"); break;
      case Thursday  : printf("Today is Thursday\n"); break;
      case Friday    : printf("Today is Friday\n"); break;
      case Saturday  : printf("Today is Saturday\n"); break;
      default        : printf("That isn't a day of the week\n");
}
```

```c
#include <stdio.h>

enum year{Jan, Feb, Mar, Apr, May, Jun, Jul,Aug, Sep, Oct, Nov, Dec};

int main(void)
{
	int i;
	for (i = Jan; i <= Dec; i++)
	{
		printf("%d ", i);
	}

	return 0;
}
```

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;
};
```

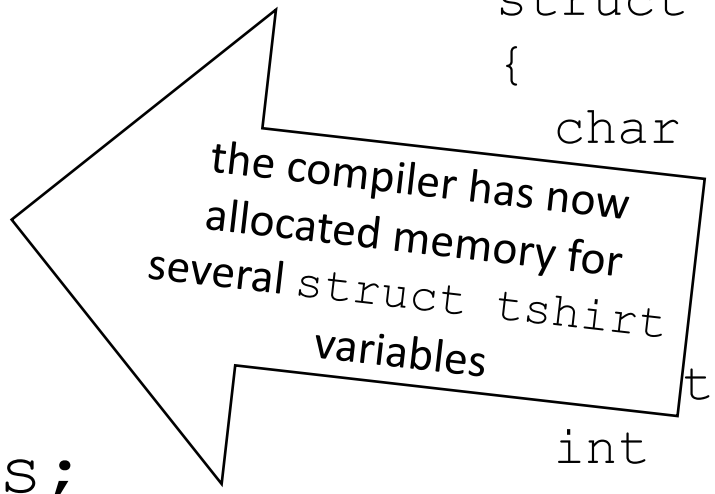
the compiler knows what a `struct tshirt` looks like but has not created one

`struct tshirt` has 6 members

Introduction to Structures

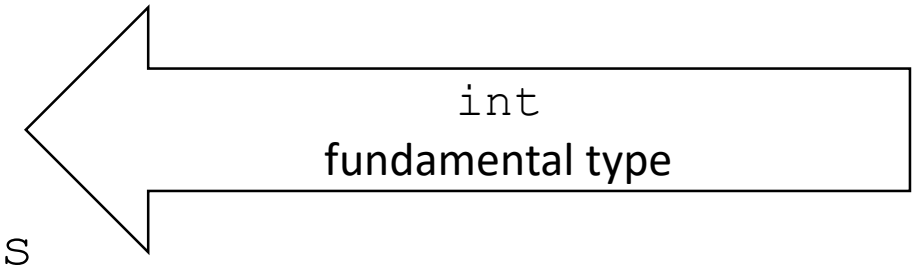
`struct tshirt` is now a user-define type that can be used to declare variables of that type

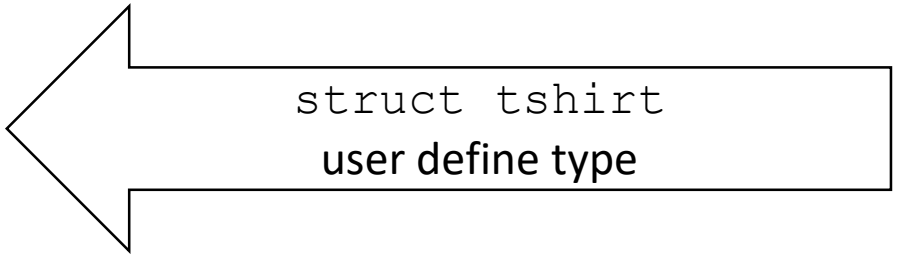
```
struct tshirt MyTShirts;
struct tshirt YourTShirts;
struct tshirt TheirTShirts;
struct tshirt OurTShirts;
struct tshirt NobodysTShirts;
```




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





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\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\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\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

reusable

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

struct tshirt NobodysTShirts;
struct tshirt OurTShirts;
```

one time use

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

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

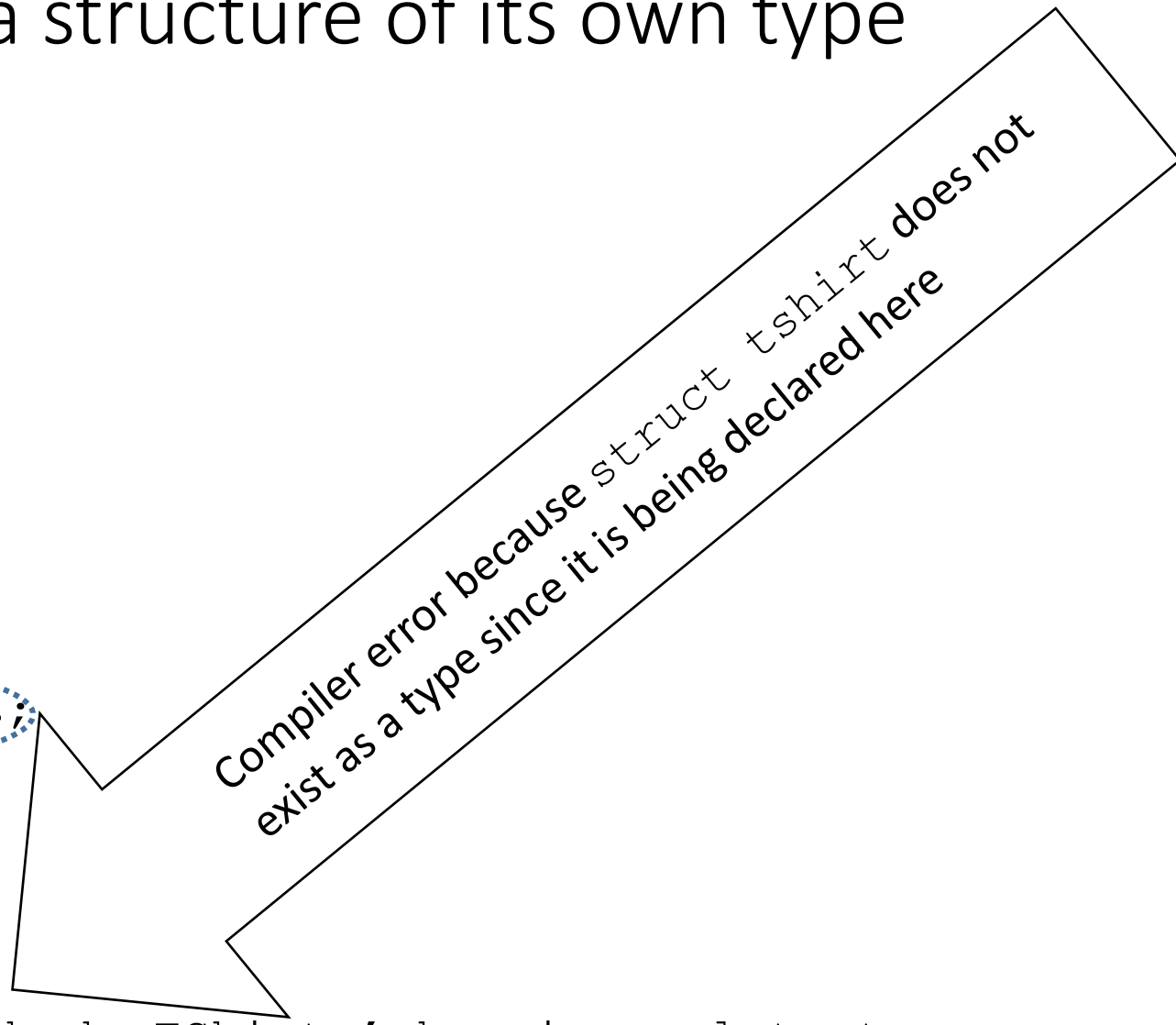
- a member of a structure cannot be a function
- a structure may not nest a structure of its own type

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
```

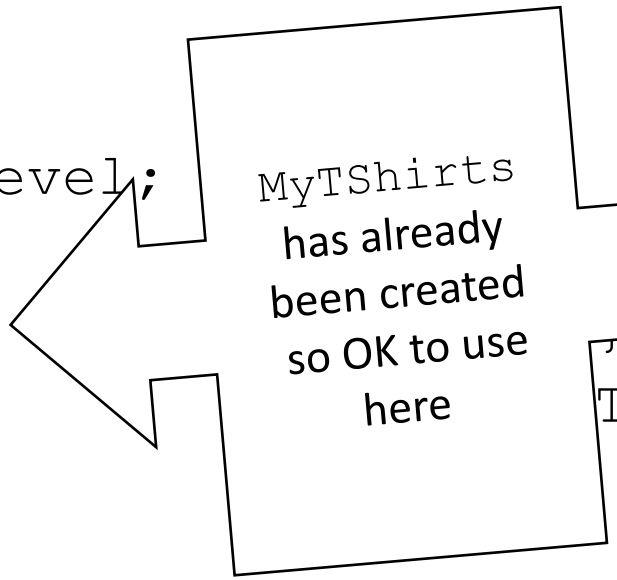
a structure may not nest a structure of its own type

```
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
```



Introduction to Structures

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




```
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

Introduction to Structures

Build a structure for a box

```
struct box
{
  int height;
  int length;
  int depth;
  float weight;
  char size[2]; // 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;
```

```
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));
```

Operations on Structures

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

```
The sizeof() MyTShirts     is 124
The sizeof() YourTShirts   is 124
The address of MyTShirts   is 0x7fff861e8ac0
The address of YourTShirts is 0x7fff861e8a40
```

128

The `sizeof()` these structures was 124 bytes yet these pointers are 128 bytes apart.

Why?

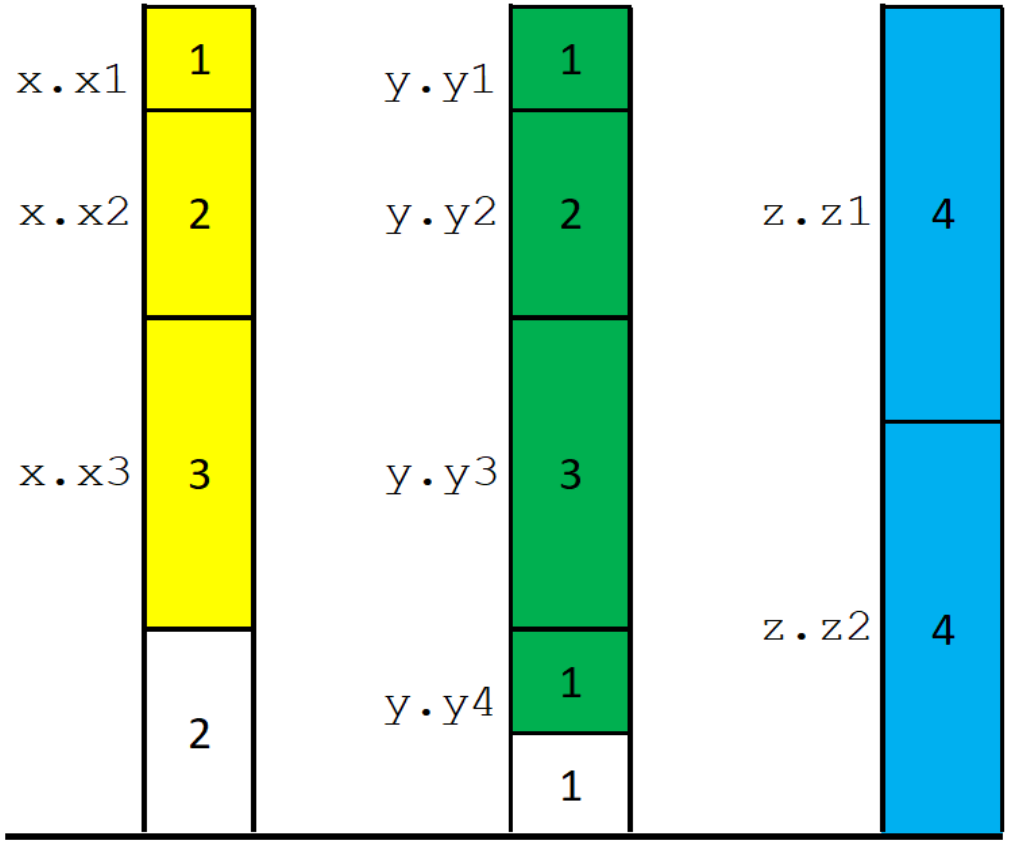
Operations on Structures

```
struct x
{
    char x1;
    char x2[2];
    char x3[3];
};

struct y
{
    char y1;
    char y2[2];
    char y3[3];
    char y4;
};

struct z
{
    int z1;
    int z2;
};
```

```
struct x  sizeof() 6      address 0x7fff9b6bb550
struct y  sizeof() 7      address 0x7fff9b6bb540
struct z  sizeof() 8      address 0x7fff9b6bb530
```

8 bytes apart

Two identical structures could still have empty/undefined space; therefore, comparison between structures is not defined.



Omega used word boundry – other systems may not.

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;
};
```

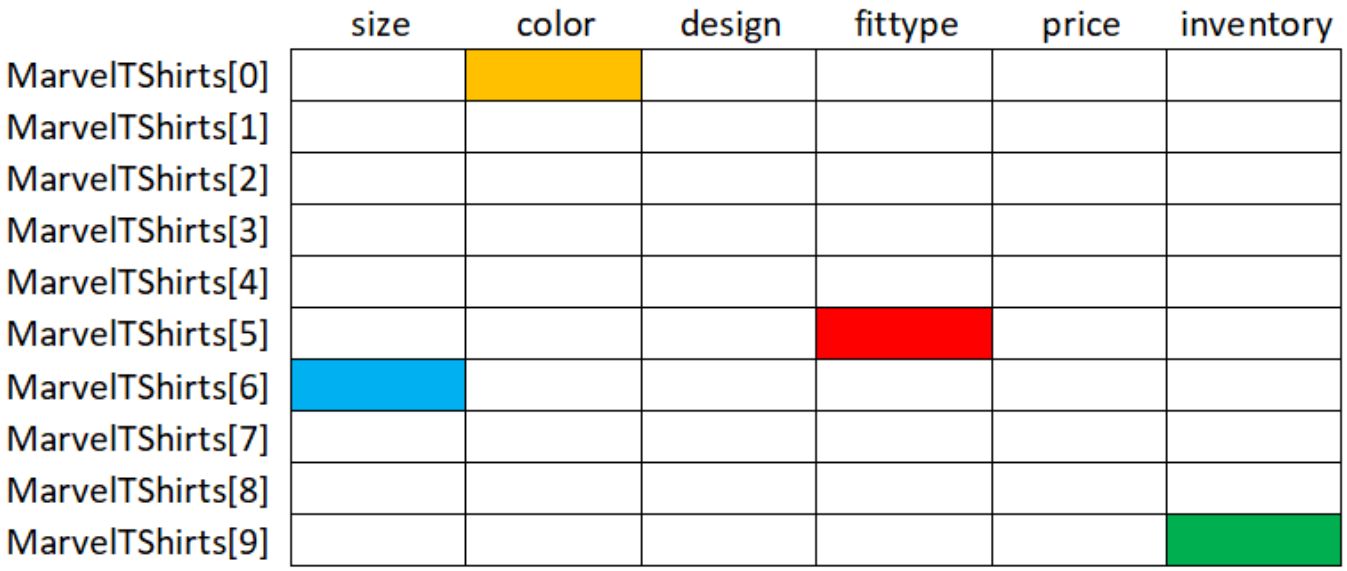
Using Structures with Arrays and Pointers

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




```
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];
```

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

Unions

A union is an aggregate data type

A union is like a structure

- it can hold members of different types
- the rules for declaring unions are the same as structures
  - a list of members is declared
  - members of a union cannot be a function and it may not contain a union of its own type
- arrays of unions and pointer to unions are allowable
- the -> member selection method is used with unions

Unions

A union is different from a structure

- will only contain one of its members at any instant (instead of all members)
- storage is allocated for the largest member (instead of all members)
- when a value is assigned to any member, it will overwrite any previously stored information (values are not retained for all members)

Unions

```
union tag
{
    member list
};
```

`union` is a keyword

`tag` is optional

Unions

```
union MyNumberUnion
{
  short ashort;
  int bint;
  long clong;
};

union MyCharUnion
{
  char a5[5];
  char b10[10];
  char c100[100];
};

union MyNumberUnion MyNU = {};
union MyCharUnion MyCU = {};
```

```
(gdb) ptype MyNU
type = union MyNumberUnion {
    short int ashort;
    int bint;
    long int clong;
}

(gdb) ptype MyCU
type = union MyCharUnion {
    char a5[5];
    char b10[10];
    char c100[100];
}
```

union1Demo.c

Unions

```
union MyNumberUnion
{
  short ashort;
  int bint;
  long clong;
};

union MyCharUnion
{
  char a5[5];
  char b10[10];
  char c100[100];
};

union MyNumberUnion MyNU = {};
union MyCharUnion MyCU = {};
```

```
(gdb) p sizeof(MyNU)
$3 = 8

(gdb) p sizeof(MyCU)
$4 = 100
```

union1Demo.c

Unions

```
union MyNumberUnion
{
  short ashort;
  int bint;
  long clong;
};

union MyCharUnion
{
  char a5[5];
  char b10[10];
  char c100[100];
};

union MyNumberUnion MyNU = {};
union MyCharUnion MyCU = {};
```

```
30                  MyNU.ashort = 1;
$8 = {ashort = 1, bint = 1, clong = 1}

31                  MyNU.bint = 10;
$9 = {ashort = 10, bint = 10, clong = 10}

32                  MyNU.clong = 100;
$10 = {ashort = 100, bint = 100, clong = 100}
```

union1Demo.c

Unions

```
union MyNumberUnion
{
  short ashort;
  int bint;
  long clong;
};

union MyCharUnion
{
  char a5[5];
  char b10[10];
  char c100[100];
};

union MyNumberUnion MyNU = {};
union MyCharUnion MyCU = {};
```

```
38  strcpy(MyCU.a5, "Friday");
$11 = {a5 = "Frida", b10 = "Friday\000\000\000",
  c100 = "Friday", '\000' <repeats 93 times>}

39  strcpy(MyCU.b10, "HappyFriday");
$12 = {a5 = "Happy", b10 = "HappyFrida",
  c100 = "HappyFriday", '\000' <repeats 88 times>}

40  strcpy(MyCU.c100,"Hello there world.  How are you?");
$13 = {a5 = "Hello", b10 = "Hello ther",
  c100 = "Hello there world.  How are you?", '\000'
<repeats 67 times>}
```

union1Demo.c

Unions

```
union MyNumberUnion
{
  short ashort;
  int bint;
  long clong;
};

union MyCharUnion
{
  char a5[5];
  char b10[10];
  char c100[100];
};

union MyNumberUnion MyNU = {};
union MyCharUnion MyCU = {};
```

```
MyNU.ashort address       0x7ffffffe7a0
MyNU.bint   address       0x7ffffffe7a0
MyNU.clong  address       0x7ffffffe7a0

MyCU.a5   address         0x7ffffffe730
MyCU.b10  address         0x7ffffffe730
MyCU.c100 address         0x7ffffffe730
```

union1Demo.c

Unions

```
struct Request
{
      char program_name;
      int service_code;
      char message[100];
} A;

struct Reply
{
      char message[100];
      int error;
      long result;
} Z;
```

```
union RequestReply
{
      struct Request A;
      struct Reply Z;
};

union RequestReply Service1 = {};
union RequestReply Service2 = {};
```

union2Demo.c

Unions

```
(gdb) ptype Z
type = struct Reply {
    char message[100];
    int error;
    long int result;
}
(gdb) ptype A
type = struct Request {
    char program_name;
    int service_code;
    char message[100];
}
```

```
(gdb) p Service1
$7 = {
  A = {
    program_name = 0
'\000',
    service_code = 0,
    message = '\000'
<repeats 99 times>
  },
  Z = {
    message = '\000'
<repeats 99 times>,
    error = 0,
    result = 0
  }
}
```

```
(gdb) p Service2
$6 = {
  A = {
    program_name = 0
'\000',
    service_code = 0,
    message = '\000'
<repeats 99 times>
  },
  Z = {
    message = '\000'
<repeats 99 times>,
    error = 0,
    result = 0
  }
}
```

Unions

CAUTION

There is no automatic mechanism to determine which member of a union is in use at any given time. It is up to the programmer to keep track.

Any member of a union can be accessed at any time; however, the contents of a member may not be meaningful if another member was most recently assigned.

Typedefs

The `typedef` storage class is used to associate an identifier with a type.

```
typedef old_type new_type;
```

- a `typedef` declaration does not cause any storage to be allocated
- `typedef` is a keyword
- `typedef` appears in declarations

Typedefs

A `typedef` is similar to a `#define` - they can both define data types

Differences

`typedef`

- processed by the compiler
- only used to define data types

`#define`

- processed by the preprocessor
- can be used to define constants, macros, and other entities as well as data types

Typedefs

```
typedef short MyShort;
typedef int   MyInt;
typedef long  MyLong;

MyShort x = 0;
MyInt y = 1;
MyLong z = 2;
```

```
(gdb) ptype x
type = short int

(gdb) ptype y
type = int

(gdb) ptype z
type = long int
```

typedef1Demo.c

Typedefs

```
typedef short Velma;
typedef int   Daphne;
typedef long  Fred;
typedef char  Shaggy[100];
typedef char  Scooby;

Velma Dinkley;
Daphne Blake;
Fred Jones;
Shaggy Rogers;
Scooby Doo;

Dinkley = 10;
Blake = 1;
Jones = 3;
Doo = (Dinkley*Jones+Jones);
strcpy(Rogers, "Scooby Snacks");

printf("%s %c%c%c%c\n", Rogers, (Dinkley*9-1), (Blake+'T'), (Jones*Dinkley+'/'), Doo);
```

```
(gdb) ptype Dinkley
type = short int
(gdb) ptype Blake
type = int
(gdb) ptype Jones
type = long int
(gdb) ptype Rogers
type = char [100]
(gdb) ptype Doo
type = char

Scooby Snacks YUM!
```

typedef2Demo.c

```
Find the area of a shape

1. Circle
2. Square
3. Rectangle
4. Triangle

Enter choice
```

```
Enter the radius of the circle 1
The area of your circle is 3.141593
```

```
Enter the length of one side 2
The area of your square is 4.000000
```

```
Enter the length of side 1 4
Enter the length of side 2 5
The area of your rectangle is 20.000000
```

```
Enter the length of the base 2
Enter the length of the height 5
The area of your triangle is 5.000000
```

typedef3Demo.c

Typedefs

Structures are a good use of typedefs

```
typedef struct
{
   float radius;
}
CIRCLE;


typedef struct
{
   float side;
}
SQUARE;
```

```
typedef struct
{
   float side1;
   float side2;
}
RECTANGLE;

typedef struct
{
   float base;
   float height;
}
TRIANGLE;
```

```
union shape
{
   CIRCLE circle;
   SQUARE square;
   RECTANGLE rectangle;
   TRIANGLE triangle;
};

enum shapes
{
   circle=1, square, rectangle, triangle
};
```

```
printf("The area of your circle is %f\n",
       M_PI * pow(EnteredShape.circle.radius, 2));

[frenchdm@omega ~]$ gcc typedef3Demo.c
[frenchdm@omega ~]$
```

The compiler optimized the call to `radius * radius` and did not use `pow()`

```
printf("The area of your circle is %f\n",
       M_PI * pow(EnteredShape.circle.radius, 3));

[frenchdm@omega ~]$ gcc typedef3Demo.c
/tmp/ccAIzQQ5.o: In function `main':
typedef3Demo.c:(.text+0xc6): undefined reference to `pow'
typedef3Demo.c:(.text+0x13b): undefined reference to `pow'
collect2: ld returned 1 exit status

[frenchdm@omega ~]$ gcc typedef3Demo.c -lm
[frenchdm@omega ~]$
```

```
int MyShape;
union shape EnteredShape;

printf("Find the area of a shape\n\n");

printf("1. Circle\n"
       "2. Square\n"
       "3. Rectangle\n"
       "4. Triangle\n\n"
       "Enter choice ");

scanf("%d", &MyShape);
```

```
Find the area of a shape

1. Circle
2. Square
3. Rectangle
4. Triangle

Enter choice
```

```
switch(MyShape)
{
  case circle :
    printf("Enter the radius of the circle ");
    scanf("%f", &EnteredShape.circle.radius);
    printf("The area of your circle is %f\n",
            M_PI * pow(EnteredShape.circle.radius, 2));
    break;
```

M_PI is defined in math.h

```
# define M_PI           3.14159265358979323846  /* pi */
```

```
  case square :
    printf("Enter the length of one side ");
    scanf("%f", &EnteredShape.square.side);
    printf("The area of your square is %f\n",
            pow(EnteredShape.square.side, 2));
    break;
```

```
   case rectangle :
      printf("Enter the length of side 1 ");
      scanf("%f", &EnteredShape.rectangle.side1);
      printf("Enter the length of side 2 ");
      scanf("%f", &EnteredShape.rectangle.side2);
      printf("The area of your rectangle is %f\n",
            EnteredShape.rectangle.side1 * EnteredShape.rectangle.side2);
      break;

   case triangle :
      printf("Enter the length of the base ");
      scanf("%f", &EnteredShape.triangle.base);
      printf("Enter the length of the height ");
      scanf("%f", &EnteredShape.triangle.height);
      printf("The area of your triangle is %f\n",
            (EnteredShape.triangle.base * EnteredShape.triangle.height) / 2);

      break;

   default :
      printf("You are out of shape\n");
}
```

```
Enter the radius of the circle 1
The area of your circle is 3.141593


Enter the length of one side 2
The area of your square is 4.000000


Enter the length of side 1 4
Enter the length of side 2 5
The area of your rectangle is 20.000000


Enter the length of the base 2
Enter the length of the height 5
The area of your triangle is 5.000000
```

typedef3Demo.c