PURDUE UNIVERSITY®

CS 240: Programming in C

Lecture 6: More Structures
Declaration vs. Definition
String Functions

Prof. Jeff Turkstra

Lecture Quizzes

- Email Rebekah Sowards (rsowards@purdue.edu) with questions
- It is your job to properly configure your device
- Test URL: https://mandalore.cs.purdue.edu/~cs240quiz/geo.php

Announcements

- Homework 2 is due Wednesday, 2/1, 9:00pm
  - Are you writing your solutions out on paper first?
- Don’t forget about the code standard
  - 20 points!
- MOSS – Policy Reminder

Friday Office Hours

- Slight adjustment
  - Shifted 10 minutes earlier
  - 1:20pm – 2:20pm

What does this do?

```
$ gcc -o hw1.c hw1_test.o
```

Reading

- Read Chapter 6.1-6.3, 6.7-6.9 in K&R
  - ...and/or Chapter 8 in Beej's
  - If you see anything about pointers, COVER YOUR EYES!

Getting the syntax of typedef…

- The syntax of typedef might seem backwards to you. Here's how to always get it right…
  - Pretend that you're defining a variable of the type that you want to see
  - Let the variable name be the name of the new type
  - Add 'typedef' to the beginning of the definition

typedef syntax cont

- For instance, if you want a type called uint5 that can be used to define an array of five unsigned integers,
  - Pretend you're defining a variable:
    ```
    unsigned int uint5[5];
    ```
  - Make it a type:
    ```
    typedef unsigned int uint5[5];
    ```
  - Use it:
    ```
    uint5 arr = { 1, 2, 3, 4, 5};
    ```

What does a structure look like?

- Type declaration:

```
struct my_data {
   int age;
   float height;
};
```

Note the semicolon!

- Storage definition and initialization:

```
struct my_data my_var = { 19, 5.3 };
```

Declaration/definition together

```
struct my_data {
   int age;
   float height;
} my_var = { 19, 5.3 };
```

Accessing elements

- Once a structure variable has been defined, you can access its internal elements with the dot (.) operator:

```
my_var.height = 6.1;
x = my_var.age;

printf("Age: %d, Height: %f\n",
        my_var.age, my_var.height);
```

How about a struct typedef

- It works the same as before
- Pretend you're defining a struct variable:

```
struct my_data md;
```

- Change it to a typedef:

```
typedef struct my_data md;
```

- Use it as a type:

```
md my_var = { 12, 5.1 };
```

Often we declare the typedef when we declare the struct…

```
typedef struct my_new_struct {
  int x;
  float y;
} new_struct_type;
…
new_struct_type var = { 1, 3.2 };
```

Meaningful example

```
struct coord {
   float x;
   float y;
   float z;
};

typedef struct coord coord_type;
```

A function that uses it

```
coord_type add_coord(coord_type a,
                     coord_type b) {
  coord_type sum = { 0.0, 0.0, 0.0 };
  sum.x = a.x + b.x;
  sum.y = a.y + b.y;
  sum.z = a.z + b.z;

  return sum;
}
```

A function that uses it

```
#include <stdio.h>

void print_coord(coord_type coord) {
  printf("(%f, %f, %f)", coord.x,
         coord.y, coord.z);
  return;
}
```

Do something with it

```
int main() {
   coord_type one = { 2, 4, 6 };
   coord_type two = { 1, 2, 3 };

   print_coord(add_coord(one, two));

   return 0;
}
```

Purdue trivia

- Purdue University is known as the "cradle of astronauts" with twenty one alumni having been chosen for space travel. Purdue astronauts include the first and last men on the moon, Neil Armstrong and Gene Cernan as well as one of America's original Project Mercury astronauts, Gus Grissom. Jerry Ross has logged 58 hours and 18 minutes in nine spacewalks - more than any other NASA astronaut.
- Purdue is also home to the oldest university-based airport in the nation.

Definitions vs. declarations

- Definition: allocates storage for a variable (or function)
- Declaration: announces the properties of a variable (or function)
- What's this?

```
struct hey {
   int   zap;
   float zing;
};
```

And this?

```
   struct point {
        int x;
        int y;
   } var;
```

More examples

- A function prototype is a forward declaration:
```
double some_function(int, float);
```
- The creation of that function is a definition:
```
double some_function(int age, float
height) {
   return age * height;
}
```
- What's a typedef? It's a declaration...
  - It turns a variable definition into a type declaration

Where to put things

- Put pure declarations outside of functions:

```
struct point {
   int x;
   int y;
};
```

- Put definitions (e.g. variables) inside functions:

```
struct point inc_point(struct point pt) {
   struct point new_pt;
   new_pt.x = pt.x + 1;
   new_pt.y = pt.y + 1;
   return new_pt;
}
```

Where to put declarations

- Don't put declarations inside functions:

```
int compute_diff(int x, int y) {
  struct point {  /* not here! */
    int x;        /* not here! */
    int y;        /* not here! */
  };              /* not here! */
  struct point my_point; /* OK */
  my_point.x = x;
  my_point.y = y;
  return my_point.x – my_point.y;
}
```

Takehome Quiz 3

1. Is the following a declaration or a definition?

```
struct wat {
   int x;
   char str[5];
} watwat;
```

2. Create a type named “waticus” for the above structure

3. How is the course going?

- Feedback on lectures?
- Feedback on TAs?
- Anything else?

Questions?

- Questions so far?

- Recall the `struct coord` declaration:

```
struct coord {
   float x;
   float y;
   float z;
};
```

Structures inside structures

- You can define structured variables inside other structures. E.g.:

```
struct segment {
   struct coord one;
   struct coord two;
};
```

- When you initialize, you need extra braces:

```
struct segment my_seg = { {0, 0, 0},
                          {0, 0, 0} };
```

Arrays in structures

- You can define array variables in structure declarations:

```
struct person {
   char name[40];
   char title[15];
   int  codes[4];
};
```

- Definition and initialization:

```
struct person jt = { “Jeff Turkstra”,
                     “Troublemaker”,
                     {10, 20, 25, 9} };
```

Assignment

- Take another definition and initialization:

```
struct person jr = { “”, “”, {0, 0, 0, 0} };
```

- Assigning elements of the arrays is usually done individually:

```
#include <string.h>
…
strncpy(jr.name, “Jon Rexeisen”, 40);
strncpy(jr.title, “Assistant Troublemaker”,
          15);
jr.codes[0] = 5;
jr.codes[1] = 10;
jr.codes[2] = 15;
jr.codes[3] = 20;
```

Compound literals

- But with C99, it doesn't have to be…

```
jr = (struct person) { "Who", "Wat",
                       {1, 2, 3, 4} };
```

Example (page 1)

```
#include <stdio.h>
#include <string.h>

struct person {
   char name[40];
   char title[16];
   int  codes[4];
};
typedef struct person person_t;

void print_person(person_t);
```

Example (page 2)

```
int main() {
  person_t jt = { “Jeff Turkstra”,
                  ”Troublemaker”,
                  {10, 20, 25, 9} };
  persont_t jr = { “”, “”, {0, 0, 0, 0} };

  strncpy(jr.name, “Jon Rexeisen”, 40);
  strncpy(jr.title,
          “Assistant Troublemaker”, 16);
  jr.codes[0] = -1;
  jr.codes[1] = 10;
  jr.codes[2] = 15;
  jr.codes[3] = 20;
}
```

Example (page 3)

```
  print_person(jt);
  print_person(jr);
  return 0;
}

void print_person(person_t pt) {
  printf("Name:  %s\n", pt.name);
  printf("Title: %s\n", pt.title);
  printf("Codes: %d, %d, %d, %d, %d\n\n",
         pt.codes[0], pt.codes[1],
         pt.codes[2], pt.codes[3]);
  return;
}
```

Example (output)

```
$ vi ex1.c
$ gcc -std=c99 -g -o ex1 ex1.c
$ ./ex1
Name:  Jeff Turkstra
Title: Troublemaker
Codes: 10, 20, 25, 9

Name:  Jon Rexeisen
Title: Assistant Troubl????     (hey!)
Codes: -1, 10, 15, 20
$
```

Avoid global variables

- A global definition is one that exists outside of any function. Avoid where possible. E.g.:

```
int my_count = 0;

int do_count() {
  int index = 0;
  for (index = 0; index < 100; index++) {
    my_count = my_count + index;
  }
  return my_count;
}
```

String comparison functions

- You can compare two strings with strcmp():

```
int result;

result = strcmp(one, two);
if (result == 0) {
  printf("The strings are equal.\n");
}
else if (result < 0) {
  printf("%s comes before %s\n", one, two);
}
else {
  printf("%s comes before %s\n", two, one);
}
```

General string operations

- Determine the length of a string: strlen()

```
char name[20] = “Jeff”;
printf(“Length of string %s is %d\n”,
        name, strlen(name));
```

  - How many bytes do you need to store it?
- When using string functions, always:

```
#include <string.h>
```

- We can explain strings more when we study pointers

Homework 1 Histogram

```
596 scores total…
100+:  (0)
 100: ====================== (524)
  90: = (9)
  80: = (17)
  70: = (14)
  60: = (5)
  50: = (6)
  40: = (2)
  30:  (0)
  20:  (0)
  10:  (0)
   0: = (19)
```

Homework 1 Remediation

- Grade reports were released last Thursday
- We will allow you to resubmit Homework 1 before Monday 2/10 at 9:00pm for 50% credit
  - We will not, however, regrade the code standard

For next lecture

- If this stuff is confusing, review Chapter 6.1-6.3, 6.7-6.9 in K&R
  - ...and/or Chapter 8 in Beej's
- PRACTICE THE EXAMPLES
- PRACTICE THE EXAMPLES
- PRACTICE THE EXAMPLES
- Keep working on Homework 2

Boiler Up!