CS 222: Arrays, Strings, Structs

Chris Kauffman

Week 3-1
Logistics

Reading

- Ch 6 (arrays)
- Ch 7 (structs)

HW 2 & 3

- HW 2 due tonight
- Conditionals, loops, arrays, natural log
- HW 3 up tomorrow, due next week

Exam 1

- This Thursday
- Zyante Ch 1-6
- This week’s Material Included

Exam Practice

Post problems tomorrow morning
Goals

- HW 2 Questions
- Arrays
Any to discuss?
Data Types

Define  Now there’s a type, it looks like blah
Declare Here is a variable, it’s type is bleh
Access Element foo of variable bar has value ...
Assign Element foo of variable bar gets value blip
Scalar Types

Only one element/value per variable

**Define**  Done for you for int, double, char etc

**Declare**  ??

**Access**  ??

**Assign**  ??
Aggregate Data: Two Kinds

Arrays  collection of the same thing (*homogeneous*)
  ▶  Like vectors/matrices
  ▶  Indexed by number
  ▶  Elements accessed via `array[index]`

structs  collection of different things (*heterogeneous*)
  ▶  A record
  ▶  Named elements (field)
  ▶  Elements accessed via `mystruct.fieldname`
Arrays

A block of memory

Define  Built in
Declare  type name[size];
Access  x = name[index];
Alter  name[index] = x;

See arraytypes.c
Initialize

Initial values are undefined - gabbledegook
Must initialize values, typically
  ▶ By hand
  ▶ By loop
    ▶ Immediate notation: \{e1, e2, e3\}
  ▶ By library call (later)

See array_init.c
Exercise: Price is Right

```c
int guesses[] = {45, 22, 86, 37, 12, 13, 47};
int find_closest_guess(int actual_price)
{
    ...
}
```

- Use a set of loops and conditionals to determine the closest value in guesses to actual_price that does not go over actual_price.
- Return the closest value from the function
- If all values in guesses are larger than actual_price, return -1

```
lila [w02-2-code] % gcc price_right.c
lila [w02-2-code] % a.out
Guesses: 45 22 86 37 12 13 47
Actual 42 closest_guess 37
Actual 46 closest_guess 45
Actual 22 closest_guess 22
Actual 10 closest_guess -1
```
Arrays as Function Arguments

Definitely can pass arrays as arguments

```c
void print_doubs(double d[], int len){
    int i;
    for(i=0; i<len; i++){
        printf("%2d: %lf\n",i,d[i]);
    }
    printf("\n");
}

int main(){
    double my_doubs[] = {1.23, 4.56, 37.89, 3.21};
    print_doubs(my_doubs, 4);

    /* VERY COMMON ERROR: don’t use [] when passing */
    print_doubs(my_doubs[], 4);
    return 0;
}
```
Exercise

Define

```c
int occurrences(int a[], int length, int key)
{
    a an array of ints
    length number of elements in a
    key what to look for in a
    returns how many times key occurs in a
}
```

```c
int stuff[8] = {-2, 1, 1, 0, -1, 1, 0, 2};
int zeros = occurrences(stuff, 8, 0); /* 2 */
int ones = occurrences(stuff, 8, 1); /* 3 */
int tens = occurrences(stuff, 8, 10); /* 0 */
```
Arrays as Multiple Returns

Definitely can set array values; changes in the caller

```c
void change_doubs(double d[], int len){
    int i;
    for(i=0; i<len; i++){
        d[i] = len-i;
    }
    printf("\n");
}

int main(){
    double my_doubs[] = {1.23, 4.56, 37.89, 3.21};
    change_doubs(my_doubs, 4);
    printf("%lf\n",my_doubs[1]);
}
```

Does this work for scalar int, double, char arguments?
Arrays as Function Returns

Definitely *cannot* return arrays from functions

/* Error! */
int [] someints(){
    int x[3] = {1,2,3};
    return x;
}

Try compiling arrayreturn.c
  ▶ Overcome this limitation next week
Arrays and the Stack: Be Careful

Uninitialized stack memory could be anything
See random_initialize.c
Common Misconceptions

- Arrays have a length but it is NOT stored anywhere implicitly
- Must use a local variable or constant to track length
- No way to ask what the length of an array is
  - `sizeof()` DOES NOT do this
BREAKTIME

Back in 15 minutes
Goals

- Exam 1 Rules
- struct
Open Resource Exam Rules

Exam 1 time: 1 Hours and 15 Minutes

Can Use, physical or electronic

- Notes
- Textbook(s) (online ok)
- Editor
- Compiler
- IDE
- Authorized Docs
- Locally stored webpages
- Dictionary

Cannot Use

- General Internet Search
- Piazza/Discussion Board
- Chat/Texting
- Communication with anyone but Instructor/Proctor

Notes

- Silence your devices
- Keep device screens visible to instructor
- If you aren’t sure of something, ask
An Instructive Example

Zyante: *Iterating through an array example: Program that finds the max item.*

In `w03-1-code/read_arrays_zyante.c`

This example is interesting for several reasons

- User input into an array
  
  ```c
  scanf("%d", &a[i]);
  ```

- Iteration converts 1-indexed loop to 0-indexed arrays

- Finds max element (*best element, useful for HW*)

- Stack allocated array based on local variable N
  
  - Contrast this with
struct: Heterogeneous Data

A block of memory with named fields

Define typedef struct {...} name_t;

Declare name_t var;

Access x = var.fieldname;

Alter var.fieldname = x;

See modernstruct.c
Declare struct

There now exists a data type that looks like ....

**Important:** Several syntax variants

- **Modern** `typedef struct {...} name;`
  - `newstruct.c`

- **Zyante** `typedef struct name_struct {...} name;`
  - `newstruct.c`

- **Old-school** `struct name {...};`
  - `oldstruct.c`

- **One-off** `struct {...} var;`
  - `weirdstruct.c`

- **OMG** `struct name {...}; typedef struct name name_t;`
Define: We’ll favor modern

typedef struct {
    double x, y;
    char color;
    int shape;
} point_t;

Warning: standard libraries use Textbook and Old-school styles
Assigning Aggregate Data

Cannot assign whole arrays to one another
Can assign whole structs to one another

See aggregate_assign.c

Related

Cannot return an array from a function
Can return a struct from a function

---

1You can return a pointer to an array, we’ll do this later; you can return a pointer to a fixed size array but the syntax is an abomination
Both are readily done: colors.c

/* A simple struct for an RGB color */
typedef struct {
    double red;
    double green;
    double blue;
} color_t;
### Returning an int vs struct

typedef struct { int a; double b;} mystruct;

#### Return an int

```c
// return an int like this
int get_int(){
    int a = 22;
    return a;
}

// NOT like this
int get_int(){
    int a = 22;
    return int;
}

// and NOT like this
int get_int(){
    int a = 22;
    return int a;
}
```

#### Return a struct

```c
// return a struct like this
mystruct get_struct(){
    mystruct s = {.a=1, b=2.3};
    return s;
}

// NOT like this
mystruct get_struct(){
    mystruct s = {.a=1, .b=2.3};
    return mystruct;
}

// and NOT like this
mystruct get_struct(){
    mystruct s = {.a=1, b=2.3};
    return mystruct s;
}
```
Exercise

- Write a function bluer
- Takes two color_structs
- Determines which struct has a higher blue field
- Returns that struct

/* A simple struct for an RGB color */
typedef struct {
  double red;
  double green;
  double blue;
} color_t;

int main(){
  color_t c1 = {.red=0.5, .green=0.7, .blue=0.1};
  color_t c2 = {.red=0.6, .green=0.2, .blue=0.5};

  color_t r = bluer(c1, c2);
  /* should be same as c2 now */
}
Aggregate Data Gotchyas

Arrays

Cannot assign whole arrays to one another
Cannot return an array from a function

structs

Can assign whole structs to one another
Can return a struct from a function

See aggregate_assign.c

---

2You can return a pointer to an array, we’ll do this later; you can return a pointer to a fixed size array but the syntax is an abomination
Sharing Types

Can copy/paste struct definitions in every .c file that needs it
  ▶ Lots of code...
  ▶ 1 change breaks everything

Instead, put `planet_t` in "planet.h"

```c
#include "planet.h"
```

Includes definitions in the right places
Composing Elements

See solarsys.c

A struct with an array

typedef struct{
    char name[128];
    double dist;
} planet_t;

...

{
    planet_t earth =
    {"earth", 1.0};
}

An array of structs

planet_t solarsys[9];

double d5 = solarsys[5].dist;

Later, structs with structs as elements
In `solarsys.c` we have a nice way to express the layout of some data in code.

- This doesn’t happen very often in C, C++, Java, etc.
- It happens *a lot* in Lisp, ML, Haskell, Python, etc.
So far

- Comments
- Statements/Expressions
- Variable Types
- Assignment
- Basic Input/Output
- Function Declarations
- Conditionals (if-else)
- Iteration (loops)
- Aggregate data (arrays, structs, objects, etc)
- Library System

Are we done?