Bits, Bytes, and Integers (part IV)

CSCI 237: Computer Organization
6th Lecture, Wednesday, Sept. 18

Kelly Shaw

Administrative Details

- Lab #2 assigned today
  - Start on it in lab. Yay!
- Snack and Gab (4:10-4:30pm today in CS Commons)
- Practice problems posted
- Read CSAPP 2.3, 2.1.3-2.1.4
- Read K&R Ch. 4-5 (as reference)
- CS Colloquium on Friday, 2:35pm in Wege
  - Concurrent Communication Contracts
  - Hannah Gommerstadt, Vassar College
  - A concurrent system is a system where multiple processes collaborate on a computation by exchanging messages. A communication contract represents a property of the computation that should remain true throughout the computation. Monitors can be used to check at runtime that a computation adheres to its contract. My work uses session types to monitor concurrent contracts. This talk will introduce session types, and present a variety of contracts that can be monitored.

Last Time: Bits, Bytes, and Integers

- Integers (Ch 2.2)
  - Representation: unsigned and signed
  - Conversion, casting
  - Expanding, truncating
  - Addition, negation, multiplication, shifting (Ch 2.3)

Today: Bits, Bytes, and Integers

- C memory addressing
  - Arrays
  - References
  - Pointers
- Integers (Ch 2.2-2.3)
  - Representation: unsigned and signed
  - Conversion, casting
  - Expanding, truncating
  - Addition, negation, multiplication, shifting (Ch 2.3)

Practice On Your Own

- What would be be printed?

```
char c = 0xe7;
short s = (short) c;
printf("c: %d s: %d \n", c, s);

s = 0xff07;
c = (char)s;
printf("c: %d s: %d \n", c, s);
```

5

5

Example Data Representations

C Data Type	Typical 32-bit	Typical 64-bit	x86-64
char	1	1	1
short	2	2	2
int	4	4	4
long	4	8	8
float	4	4	4
double	8	8	8
long double	–	–	10
pointer	4	8	8

6

6

Pointers

- Pointers store a memory address
- A static variable's memory address can be obtained with `&`
  - `int val = 3;`
  - `int *ptr = &val;` `// Same as int *ptr; ptr = &val;`
- The pointer variable stores a value that is the address
  - `ptr == NULL`
- To access the value pointed to, need to use the * operator
  - `int num = *ptr;`
  - `*ptr = 7;`
- Modifying the pointer variable without the * operator changes what is pointed to
  - `ptr = &num;`
  - `*ptr = 10;`

7

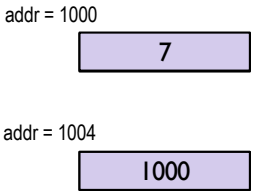
7

```
int val = 4;

int *ptr = NULL;

ptr = &val;

*ptr = 7;
```



8

8

Practice

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

int main(int argc, char *argv[])
{
   int x = 3, y = 10;
   int *ptr = NULL;

   ptr = &y;
   *ptr = 7;
   y++;
   printf("x %d y %d ptr %d\n", x, y, *ptr);

   ptr = &x;
   *ptr = *ptr + 1;
   printf("x %d y %d ptr %d\n", x, y, *ptr);

   ptr = NULL;
   // Why doesn't the next line of code work?
   printf("x %d y %d ptr %d\n", x, y, *ptr);
   return 0;
}
```

x 3 y 8 ptr 8

x 4 y 8 ptr 4

ptr == NULL

9

9

structs

- Statically declared structs created just like primitive variables
  - `struct set set1;`
- To access data fields, use . Operator
  - `set1.num_elements = 3;`
- Pointers can store the address of structs
  - `struct set *ptr = &set1;`
- To use pointer to access fields in the struct, use either combination of * and . operators or -> operator
  - `(*ptr).num_elements = 10;`
  - `ptr->num_elements = 7;`

10

10

```
#include <stdio.h>

struct set {                  // declaration of struct set
  int elements[5];
  int num_elements;
};

int main(int argc, char *argv[])
{
  struct set setA;            // instance of struct set

  setA.num_elements = 3;      // initialize instance fields
  setA.elements[0] = 1;
  setA.elements[1] = 2;
  setA.elements[2] = 3;

  for(int i = 0; i < setA.num_elements; i++){
    printf("%d\n", setA.elements[i]);
  }

  struct set *ptr;            // create alias through pointer
  ptr = &setA;
  printf("%d\n", ptr->num_elements);

  return 0;
}
```

11

11

Representing Arrays In Memory

- Each array element the size of the specified type
- Array elements laid out in memory in contiguous memory locations
- Array name also refers to the address of the first element in the array
  - `arr == &arr[0]`
- Any operation done with an array can be done with pointers (pointer arithmetic)
  - `int *ptr = arr;`
  - `ptr == &arr[0]`
  - `(ptr+1) == &arr[1]`

```
int arr[6];
```

arr	
31	arr[0]
38	arr[1]
32	arr[2]
31	arr[3]
33	arr[4]
00	arr[5]

12

12

```
int main(int argc, char *argv[])
{
  int arr[4];

  arr[0] = 0;
  arr[1] = 1;
  arr[2] = 2;
  arr[3] = 3;

  int *ptr = arr;
  // Above equivalent to
  // int *ptr;
  // ptr = arr;

  printf("Addresses: \n");
  printf("\t arr     \t %llx \n", (unsigned long long)arr);
  printf("\t &arr[0] \t %llx \n", (unsigned long long)(&arr[0]));
  printf("\t ptr     \t %llx \n", (unsigned long long) ptr);
  printf("\t &ptr    \t %llx \n", (unsigned long long)&ptr);

  return 0;
}
```

Addresses:
arr 7ffee62258a0
&arr[0] 7ffee62258a0
ptr 7ffee62258a0
&ptr 7ffee6225888

13

13

Why use pointers? (for statically declared variables)

- Modular programming
  - Allows you to pass the memory address of a local variable to another function
    - Allows other function to read and update variable so that modifications are reflected in function where variable created
  - Avoids need to create a copy of the variable contents (so less memory is used)
- Enables single name for collection of elements/memory addresses (i.e., array)
  - Only need one pointer/array name to access all of the elements of an array

14

14

Representing Strings

```
char S[6] = "18213";
```

- Strings in C
  - Represented by array of characters
  - Each character encoded in ASCII format
    - Standard 7-bit encoding of character set
    - Character "0" has code 0x30
      - Digit *i* has code 0x30+*i*
  - String should be null-terminated
    - Final character = 0

IA32	Sun
31	31
38	38
32	32
31	31
33	33
00	00

- `char *s` **vs.** `char s[]`

15

15

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

int main(int argc, char *argv[])
{
  char arr[10];
  char *ptr = NULL;
  strcpy(arr, "Donuts!");

  printf("arr %s\n", arr);
  ptr = arr;
  printf("ptr %s\n", ptr);

  arr[2] = 'N';
  ptr[3] = 'U';
  printf("\nAfter uppercase\n");
  printf("arr %s\n", arr);
  printf("ptr %s\n", ptr);

  *(ptr+2) = '\0';
  printf("\nAfter null terminator\n");
  printf("arr %s\n", arr);
  printf("ptr %s\n", ptr);
}
```

arr Donuts!
ptr Donuts!

After uppercase
arr DoNUts!
ptr DoNUts!

After null terminator
arr Do
ptr Do

16

16