Structures and Pointers

Comp-206 : Introduction to Software Systems
Lecture 11

Alexandre Denault
Computer Science
McGill University
Fall 2006

Note on Assignment 1

- Please note that *handin* does not allow you to hand in a file whose name starts with a period
  - ex: .bash_profile
- You will have to rename that file before handing it in
  - ex: bash_profile

Pass by reference, pass by value

- Primitives (such as int, short, long, float, etc) are passed by value. This means you can change their value in a function and they will not be affected.
- Arrays are passed by value. This means that if you change the values in an array, it will affect the whole application.
- Pointers, which we will see in a couple of lectures, complicate this even more.

```
void testFunction(int a, int myArray[]) {
   a = 10; // No effect to rest of application
   myArray[0] = 10; // Affects rest of app.
}
```

Structures

- Structures are a data type composed of several other data types.
  - Think of it as a container, a variable that has variables inside it.
- You can define new structures using the `struct` keyword.

```
struct course {
      int number_of_students;
      char[100] name_professor;
      char[100] location_building;
      int location_room;
}
```

Using a structure

- To use a structure, you need to instantiate a copy of it.
- All you need to do is to declare the variable for the instance.

```
struct course cs206;
```

- You can then fill it with data.

```
cs206.number_of_student = 60;
cs206.name_professor = “Alex”;
cs206.location_building = “MacDonald”;
cs206.location_room = 328;
```

- With structures, you can declare the variable and initialize it with data in one command.

```
struct course cs206 = {60, “Alex”, “MacDonald”,
  328);
```

typedef and struct

- You can use typedef to define the structure as a new type.

```
typedef struct course {
      int number_of_students;
      char[100] name_professor;
      char[100] location_building;
      int location_room;
}
```

- When creating a variable of this type, you no longer need to specify the struct keyword.

```
struct course cs206;
```

Coercion or Type-Casting

- Coercion : forcing one variable of one type to be another type.
- Sometimes, type-casting is implicit :

```
int a = 2;
float b = a;            // b = 2.0
```

- Most of the time, it's safer to specify it:

```
float a = 3.1415;
int b = (int)a;         // b = 3
```

- When in doubt, type cast:

```
int a = 2;
float b = 3 / a;        // b = 1.0
float c = 3 / (float)a; // c = 1.5
```

Enumerated Types

- Enumerated types : contain a list of constants that can be addressed in integer values.

  -
    ```
    enum days {monday, tuesday, wednesday,
    thursday, friday, saturday, sunday};
    ```
- As with arrays first enumerated name has index value 0.
  - So monday has value 0, tuesday 1, ...
- We can also override the 0 start value:

  -
    ```
    enum days {monday = 1, tuesday, wednesday,
    thursday, friday, saturday, sunday};
    ```
- Or simply assign different numerical values:

  -
    ```
    enum days {monday = 10, tuesday = 20,
    wednesday = 30, thursday = 40, friday = 50,
    saturday = 60, sunday = 0};
    ```

Using Enumerations

- Creating a variable of an enumeration is similar to a structure:
  - enum days week1;
- If you typedef an enumerated type, you can use it without the enum keyword.

  -
    ```
    typedef enum days {monday = 1, tuesday,
    wednesday, thursday, friday, saturday,
    sunday};
    ```
  - `days week1;`

Static Variables

- Static Variable : variable local to particular function but only initialized once (on the first call to function).

```
function int count() {
        static int counter = 0;
        counter++;
        return counter;
}
```

- The following function will count the number of time it is called.
- The same count have been done with a global variable, but counter doesn't need global visibility.

Pointers

- One of the most difficult feature of C.
- Also one of the most fundamental and important feature.
- Pointers exist of efficiency and flexibility reasons.
- They are used explicitly with
  - Functions
  - Arrays
  - Structures

What are pointers?

- A pointer is a variable which contains the address in memory of another variable.
  - Think of it as an integer variable that points to a block of memory.
- We can have a pointer to any variable type.

Pointer operators

- The unary or monadic operator & gives the “address of a variable”.
- The indirection or dereference operator * gives the “contents of an object pointed to by a pointer”.
- Pointers are declared using the indirection operator:
  - int* a;

Simple Pointer Example

```
int a, b;
int* p;

a = 5;
b = 10;
p = &a;    // p is pointing on a
*p = 6;    // Value of a is now 6;
p = &b;    // p is pointing on b
*p = 11;   // Value of b is now 11;
```

Pointers and Functions

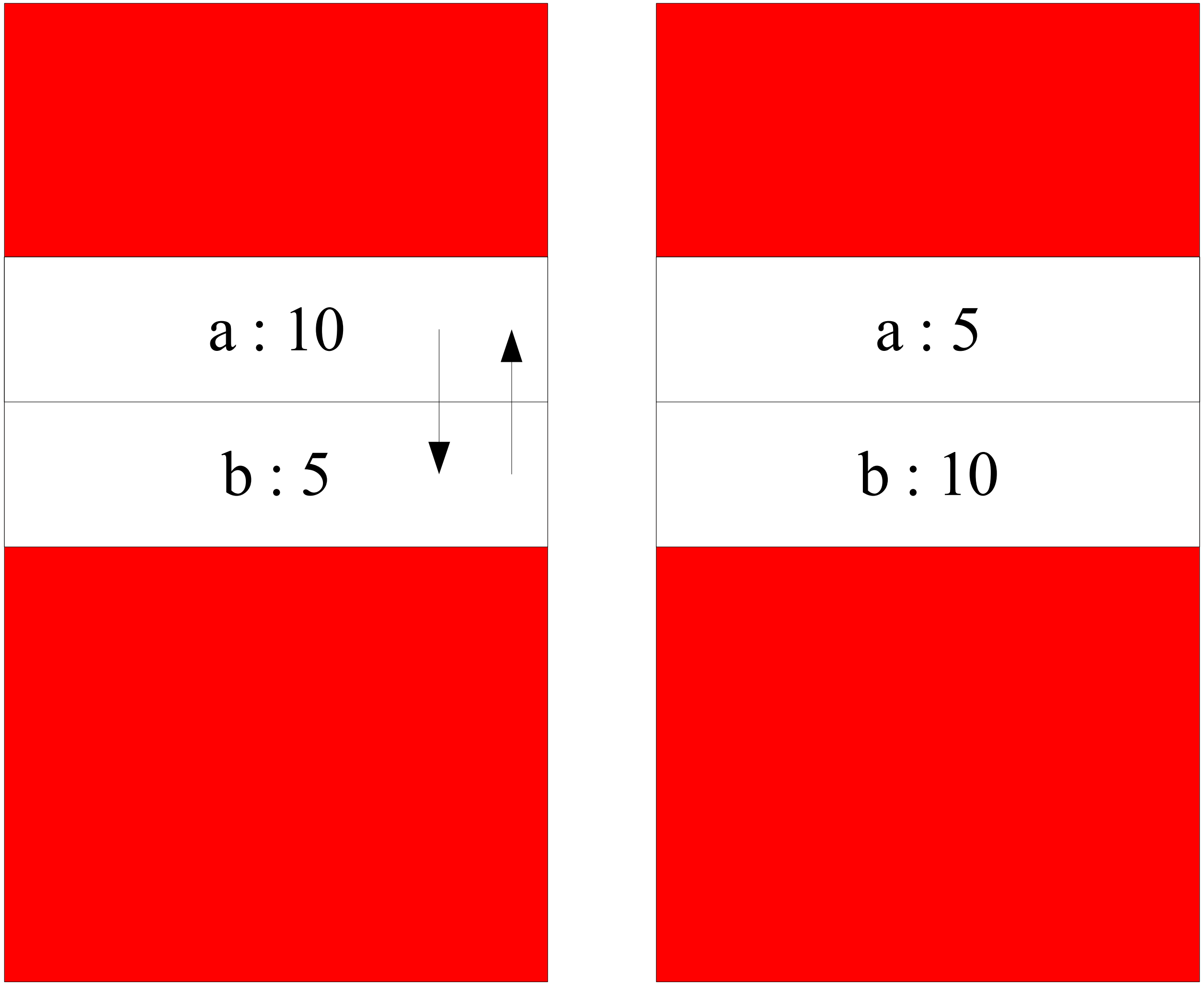
- The following functions cannot be implemented without pointers:

```
void swap(int a, int b) {
   int temp = a;
   a = b;
   b = temp;
}
```

- This function only alters the value of the local variables a and b. The change is invisible to the calling function.

```
int a = 5, b = 10;
swap (a,b);
```

A look into memory



a : 10

b : 5

a : 5

b : 10

memory of swap function

memory of function calling swap

Swap using pointers

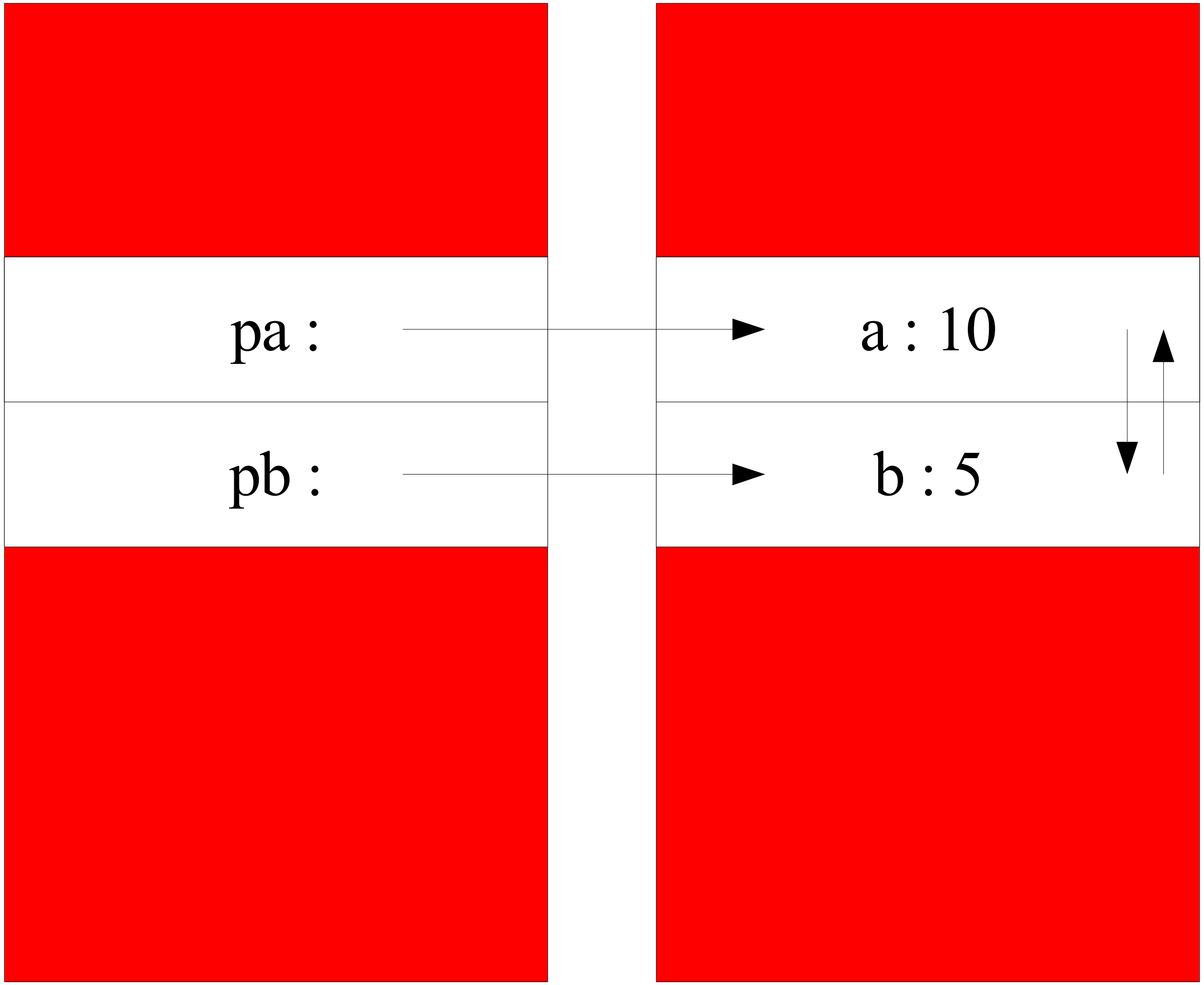
The following function does work, because it uses pointers to the integers.

```
void swap (int * pa, int * pb) {
   int temp = *pa;
   *pa = *pb;
   *pb = temp;
}
```

- When calling the swap function, the address the integers must be provided:

```
int a = 5, b = 10;
swap (&a,&b);
```

Using pointers instead

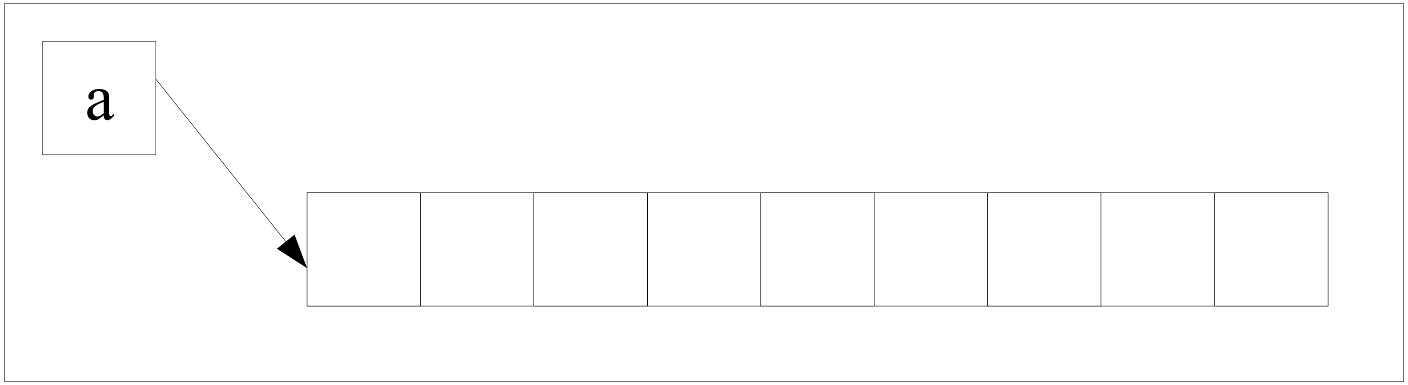


memory of swap function

memory of function calling swap

Pointers and Arrays

- Arrays and pointers are very related in C.
- In fact, when you create an array in C, you allocate a block of memory and create a pointer to the first element of that block of memory.
  - int a[10];



Dynamic Memory Allocation

- The malloc() function allocates a block of memory and returns a pointer to that allocated memory.
  - void *malloc(size_t size);
- The size of the block must be specified.
- That block memory is not initialized.
  - It will contain whatever is currently in memory.
- Be careful not to access memory outside what you allocated.
  - Nothing will prevents you from accessing outside that block of memory.

Using the blocks of memory

- Both malloc and calloc return a void pointer (void *).
- In C, you use a void* when return a generic pointer.
- This generic block of memory must be cast before it can be used.

  ```
  int *a = (int *) malloc( sizeof(int) * 40 );
  ```
- The sizeof() function simplifies the allocation of memory by calculating the size of the provided data type.

Deallocating Memory

- The free() function releases the specified memory space.
  - void free(void *ptr);
- The specified memory must have been returned by a previous call to malloc(), calloc() or realloc().
  - Otherwise, undefined behavior occurs.
- Not releasing memory after finishing with it can create memory leaks.
  - This can be an especially serious problem if you continually allocate memory.