

EL2310 – Scientific Programming

Lecture 11: Structures and Memory



Andrzej Pronobis
(pronobis@kth.se)

Royal Institute of Technology – KTH

Overview

Lecture 11: Structures and Memory

Wrap Up

Structures Continued

Memory Allocation

Tasks

Tasks

NULL

- ▶ Bad idea to leave variables uninitialized
- ▶ This is true for pointers as well
- ▶ To mark that a pointer is not assigned and give it a well defined value we use the `NULL` pointer.

- ▶ Ex:

```
int *p = NULL;
```

```
...
```

```
if (p != NULL) *p = 4;
```

- ▶ Testing if not `NULL` before using a pointer is good practice (and setting it to `NULL` when unassigned)

Pointer to functions

- ▶ Just like in MATLAB you can work with pointers to functions
- ▶ In C you need to declare explicitly what the argument the function has as input and output
- ▶ Ex: Pointer (`fcn`) to a function that returns an `int` and takes a `double` as argument
`int (*fcn)(double)`

Arithmetic operations with pointers

► Comparison

- ▷ `Let int *p1, *p2;`
- ▷ What is the difference?
 - ...
 - `if (p1 == p2) ...`
 - ...
 - `if (*p1 == *p2) ...`

► Adding/subtracting pointer

- ▷ `int *p1, *p2;`
- ▷ What does `p1+3` mean?
- ▷ What does `p2-p1` mean?

Pointers and functions

- ▶ Functions can only have a single return type
- ▶ Scope of argument is local to function
- ▶ Use of pointers for multiple “output” from function

Task 0

- ▶ Write a set of void functions that exemplifies,
 1. Returning a value through a pointer argument of a function
 2. Difference in use between arrays and pointers passed to a function
 3. Passing function pointers to functions
- ▶ Use Task1 and Task3 from previous lecture (lecture 10) as examples.

struct

- ▶ So far we looked at basic data types and pointers
- ▶ It is possible to define your own types
- ▶ For this we use a `struct`

- ▶ Ex:

```
struct complex_number {  
    double real;  
    double imag;  
};
```

- ▶ The variables `real` and `imag` are called *members* of the `struct complex_number`.
- ▶ Declaring variables `x, y` of type `complex_number` is done with `struct complex_number x, y;`

Lecture 11: Structures and Memory

Wrap Up

Structures Continued

Memory Allocation

Tasks

Tasks

Assigning struct

- ▶ Can be assign similar to arrays

- ▶ `struct complex_number x = { 1.1, 2.4 };`

- ▶ Will give the complex number $x = 1.1 + 2.4i$.

- ▶ One more example:

```
struct person {  
    int age;  
    int sibling_ages[10];  
    char name[32];  
};  
struct person p1 = {1, {26, 33}, ``Mr T``};
```

- ▶ Will assign value to the ages of two siblings

Accessing members of a `struct`

- ▶ If you want to set/get the value of a member you use the “.” operator
- ▶ Ex:

```
struct complex_number {  
    double real;  
    double imag;  
};  
struct complex_number x;  
x.real = 1.1;  
x.imag = 2.4;
```

typedef

- ▶ `typedef` can be used to give types a new name, like a synonym
- ▶ Can introduce shorter names for things
- ▶ Ex:

```
struct position {  
    double x;  
    double y;  
};  
typedef struct position pos;
```

- ▶ **Alternative:**

```
typedef struct {  
    double x;  
    double y;  
} pos;
```

Pointers and structures

- ▶ You can use pointers to structures

- ▶ Ex:

```
struct complex_number x;  
struct complex_number *xptr = &x;
```

- ▶ To access a member using a pointer we use the “->” operator

- ▶ Ex: `xptr->real = 2;`

- ▶ Same as `(*xptr).real = 2;`

- ▶ and `x.real = 2;`

Structures of structures

- ▶ You can have any number of levels of structures of structures

- ▶ Ex:

```
struct position {
    double x;
    double y;
};
struct line {
    struct position start;
    struct position end;
};
```

Structures of structures

```
▶ Continued. struct line l;  
l.start.x = 4;  l.start.y = 6;  
l.end.x = 2;    l.end.y = -1;  
struct line *lp = &l;  
l->start.y = 42;
```

Pointers to structures in structures

- ▶ Normally you need to declare a type before you use it.
- ▶ You can have a pointer to the structure you define
- ▶ Ex:

```
struct person {  
    char name[32];  
    struct person *parent;  
};
```

cast

- ▶ Some conversions between types are implicit
- ▶ Ex: `double x = 4;`
- ▶ In other cases you need to tell the compiler to do this
- ▶ Ex: `double fraction = 3 / 4;` will give 0
- ▶ Ex: `double fraction = (double)3 / 4;`
- ▶ We *casted* 3 from an `int` to a `double`
- ▶ **Be careful when casting!**

Casting pointers

- ▶ We can convert also between pointer types
- ▶ These are typically allowed with gcc (implicit conversions):

```
int a;  
char *pa = &a;  
int *b;  
char *pb = b;
```

- ▶ Will generate a warning, use an explicit cast:

```
int a;  
char *pa = (int*) &a;  
int *b;  
char *pb = (int*) b;
```

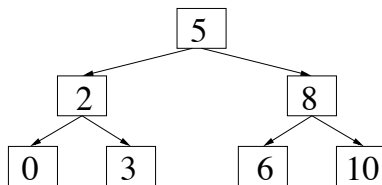
- ▶ What will happen?

```
printf("%c %c %c %c", a[0], a[1], a[2], a[3]);
```

- ▶ **Be even more careful when casting pointers!**

Task 1

Assign any integer to the closest in the set: $\{0, 3, 6, 10\}$



- ▶ Use the above decision tree structure.
- ▶ If greater or equal than the node value, follow right, otherwise, follow left

Lecture 11: Structures and Memory

Wrap Up

Structures Continued

Memory Allocation

Tasks

Tasks

Dynamic allocation of memory

- ▶ Sometimes you do not know the size of arrays when you write code
- ▶ Idea: Allocate memory dynamically
- ▶ This way you can allocate memory at runtime
- ▶ You can calculate how much memory you need and allocate (e.g. array) only then

malloc

- ▶ Allocate memory with `malloc`
- ▶ Need to `#include <stdlib.h>`
- ▶ This function returns a pointer of type `void*`
- ▶ Ex: `int *p = malloc(100*sizeof(int));`
- ▶ Will allocate memory for 100 ints
- ▶ You can use an explicit cast:
`int *p = (int*)malloc(100*sizeof(int));`

free

- ▶ You should free the memory that you no longer need!!!
- ▶ Ex:

```
int *p = (int *)malloc(100*sizeof(int));  
  
...  
  
free(p);
```
- ▶ If you do not free allocated memory you will get memory leaks
- ▶ Your program will crash eventually
- ▶ A big problem if you program should run a very long time

Memory

- ▶ When you run your program the memory is divided between the *heap* and the *stack*
- ▶ The stack:
 - ▷ Memory allocated for all parameters and local variables of a function
 - ▷ Fast-allocation memory
 - ▷ Current function at the top of the stack
 - ▷ When a function returns its memory is removed from the stack
- ▶ The heap:
 - ▷ Used for *persistent* data
 - ▷ Dynamically allocated memory

From [http://www.csl.mtu.edu/cs3090/www/lecture-notes/Memory Allocation.ppt](http://www.csl.mtu.edu/cs3090/www/lecture-notes/Memory%20Allocation.ppt)

Tip when using dynamic memory allocation

- ▶ If you have a `malloc` think about where the corresponding `free` is

Lecture 11: Structures and Memory

Wrap Up

Structures Continued

Memory Allocation

Tasks

Tasks

Task 2

- ▶ Define a structure for a complex number
 - ▶ Define functions to perform operations on the complex numbers
 - ▶ Write a program that uses these functions to compute
 - ▷ addition, multiplication, subtraction and division of two complex numbers
 - ▷ the magnitude
 - ▷ the angle
- Addition: $(a + bi) + (c + di) = (a + c) + (b + d)i$
 - Subtraction: $(a + bi) - (c + di) = (a - c) + (b - d)i$
 - Multiplication: $(a + bi)(c + di) = ac + bci + adi + bdi^2 = (ac - bd) + (bc + ad)i$
 - Division: $(a + bi)/(c + di) = \left(\frac{ac + bd}{c^2 + d^2}\right) + \left(\frac{bc - ad}{c^2 + d^2}\right)i$

Task 3

- ▶ Write a program that investigates how different numbers are represented
- ▶ Define a set of variables `short`, `int`, `unsigned int`, ... and print the value of each byte in turn
- ▶ Does your computer use *little-endian* (least significant byte (LSB) at the lowest address) or *big-endian* (MSB at the lowest address) representation of numbers

Next Time

- ▶ Continue with Memory
- ▶ File I/O
- ▶ C Project online!

`http://www.pronobis.pro/teaching/el2310/
coursework/project2`