## EL2310 – Scientific Programming

Lecture 11: Structures and Memory



Yasemin Bekiroglu (yaseminb@kth.se)

Royal Institute of Technology - KTH

### Overview

### Lecture 11: Structures and Memory

Wrap Up Structures Continued Memory Allocation

#### **Tasks**

Tasks

# Lecture 11: Structures and Memory Wrap Up

Memory Allocation

#### **Tasks**

Tasks

### Last time

- Pointers
- Started with (struct)

# **Today**

- Repetition pointers
- More on (struct)
- Memory

### **Pointer**

- A pointer is a variable that holds the address of another variable
- **E**x:

```
int a;
int* b = &a;
*b = 4;
```

Will set a to be 4

# Pointers to pointers

- Can have pointers to pointer
- "Address of the address to the value"
- Notation similar

```
int a;
int *p = &a;
int **pp = &p;
```

- Example use: Change address of pointer in function
- Dereferencing:
  - \*pp to get pointer to a
  - \*\*pp to get value of a

## void pointer

- Normal pointers point to a certain type like int
- The void pointer (void\*) represents a general pointer that can point to anything
- You can assign to and from a void \* without a problem
- You can not dereference a void∗
- The void pointer allows you to write code that can work with addresses to any data type

#### 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

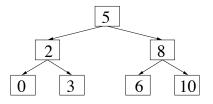
- int \*p1, \*p2;
- ▶ What does p1+3 mean?
- What does p2-p1 mean?

### 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 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

### Task 0

Wrap Up

- 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
- **E**x:

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

### Structures of structures

### 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
- $\triangleright$  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 convertions):

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

Will generate a warning, use an explicit cast:

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

Be even more careful when casting pointers!

### 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

### Common mistakes

- Forgetting to free memory (memory leak!!!)
- Using memory that you have not initialized
- Using memory that you do not own
- Using more memory than you allocated
- Returning pointer to local variable (thus no longer existing)

# Tip when using dynamic memory allocation

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

# How to Debug C Program using gdb

- Compile your C program with -g option to allow the compiler to collect the debugging information.
- Launch gdb with your program e.g.: gdb main
- Set a breakpoint: break "line number"
- Run your program: run
- Print variable values: print "variable name"
- Run step by step:
  - c or continue: continue executing until the next break point.
  - ▷ n or next: execute the next line as single instruction.
  - s or step: same as next, but does not treats function as a single instruction, instead goes into the function and executes it line by line.

Tasks

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: 
$$(a + bi) + (c + di) = (a + c) + (b + d)i$$

$$\qquad \qquad \text{Subtraction: } (a+bi)-(c+di)=(a-c)+(b-d)i$$

$$\qquad \qquad \text{Multiplication: } (a+bi)(c+di) = ac+bci+adi+bdi^2 = (ac-bd)+(bc+ad)i$$

$$\qquad \text{Division:} \ (a+bi)/(c+di) = \left(\frac{ac+bd}{c^2+d^2}\right) + \left(\frac{bc-ad}{c^2+d^2}\right)i$$

Tasks

### 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