EL2310 – Scientific Programming

Lecture 10: Pointers and Structures

Andrzej Pronobis
(pronobis@kth.se)

Royal Institute of Technology – KTH

Overview

Lecture 10: Pointers and Structures

- Wrap Up
- Pointers Continued
- Function Pointers
- Constant variables and structs
- Pointers and Structs

C Tasks

Last time

- Splitting into separate files
- Makefiles
- Scope rules
- Beginning with pointers

Today

- Even more on pointers
- Complex data types (`struct`)

Variable scope: local variables

- The scope of a variable tells where this variable can be used
- Local variables in a function can only be used in that function
- They are automatically created when the function is called and disappear when the function exits
- Local variables are initialized during each function call

Variable scope: `extern`

- If you want to use a variable defined externally to a function in some other file, you need to use the keyword `extern`
- `extern int value;` declares a variable `value` defined externally that will now available to us

Variable scope: `static`

- If you want a variable defined outside a function to be hidden in a file, use the keyword
  `static`
- A variable declared `static` can be used as any other variable in that file but will not be seen from outside

Initialization

- External and static variables are guaranteed to be 0 if not explicitly initialized
- Local variables are NOT initialized (contain whatever is in the memory)

Pointers

- Pointers are special kinds of variables
- They contain the address of another variable
- Pointers are like bookmarks
- Used heavily in C:
  - To pass reference to big things in memory
  - To return multiple values from functions
- Have to be used with care

Declaring a pointer

- A pointer is declared by a * as prefix to the variable
  Can think of it as a suffix to the data type as well
  "`int*` is a pointer to an `int`"
- Ex: Pointer to an interger

```
int *ptr;
```

Assigning a pointer

- You assign a pointer to a value being an address of a memory location
- The address typically correspond to a variable in memory
- You get the address of a variable with the unary & operator
- Ex:

  ```
  int a;
  int *b = &a;
  ```
- We say that b “points” to a

Dereferencing a pointer

- To get the value in the address pointed to by a pointer, use the operator dereferencing operator *
- Ex:

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

- Will set `a` to be 4

Copying pointers

- Copying the data

```
*ptr1 = *ptr2;
```

- Copying the pointer address

```
ptr1 = ptr2;
```

Makefiles

- MAKE tool to automate building, ex. compilation
- Rules from Makefile
- `task1:`
  ```
  gcc -o task1 task1.c task1_includes.c -lm
  ```
- Tutorial in the course materials! Check out tasks!

Lecture 10: Pointers and Structures

- Wrap Up
- Pointers Continued
- Function Pointers
- Constant variables and structs
- Pointers and Structs

C Tasks

Pointers and arrays

- Can use pointer to perform operations on arrays
- Ex:

```
int a[] = {1,2,3,4,5,6,7,8};
int *p = &a[0];
```

- Will create a pointer that points to the first element of `a`
- The following are equivalent

  `p = &a[0]` and `p = a;`

  `a[i]` and `*(a+i)`

  `&a[i]` and `a+i`

  `*(p+i)` and `p[i]`

  `fcn(int *a)` and `fcn(int a[])`

Stepping forward backward with pointers

- A pointer points to the address of a variable of the given data type
- If you say `ptr = ptr + 1;` you step to the next variable in memory assuming that they are all lined up next to each other
- Can also use shorthand `ptr++` and `ptr--` as well as `ptr+=2;` and `ptr-=3;`

More on pointers

- One has to be careful when moving pointers
- Common mistake when using a pointer: you move it outside the memory space you intended and change unexpected things
- The following is allowed but make it hard to read

  ```
  int a[] = {6,5,4,3,2,1};
  int *p = &a[2];
  p[-2] = 2;
  ```
- What value will change?

Constant strings

- The “Hello world” in `printf("Hello world");` is a constant string literal
- It cannot be changed
- Consider the two expressions

```
char amsg[] = "Hello world";
char *pmsg = "Hello world";
```

- `amsg` is a character array initialized to “Hello world”. You can modify the content of the array since it contains a copy of the string literal.
- `pmsg` is a pointer that points to a constant string directly. You cannot change the character in the string but change what pmsg points to.

Task 1

- Write the function
  `void strcpy2(char *dest, char *src);`
- Should copy the string `src` into `dest`

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`

Arrays of pointers

- Can also make arrays of pointers like any other data type
- Ex: `char *sa[100];` array of 100 C strings
- Ex: `int *ia[100];` array of 100 `int` pointers

`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

`void` pointer cont'd

- NOT ALLOWED

  ```
  int a = 4;
  void *b = &a;
  *b = 2;
  ```

- ALLOWED

  ```
  int a = 4;
  void *b = &a;
  int *c = b; *c = 2;
  ```

NULL

- Bad idea to leave variables unitialized
- 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)

Selective computations

- Using the `NULL` pointer we can tell a function parameters need not be calculated
- Ex: `void calc(double x, double *v1, double *v2);`
- If we call this method with v1 or v2 `NULL` the function can choose not to perform certain calculations

Lecture 10: Pointers and Structures

- Wrap Up
- Pointers Continued
- Function Pointers
- Constant variables and structs
- Pointers and Structs

C Tasks

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)
  ```

Arrays of pointers to functions

- Can store arrays of function pointers
- To declare an array `pf` of 4 pointers to functions we do
  `double (*pf[4])(double);`
- You assign values by
  `pf[0] = &fcn1;`
- and you use them as
  `pf[0](4.2);`

Lecture 10: Pointers and Structures

Wrap Up

Pointers Continued

Function Pointers

Constant variables and structs

Pointers and Structs

C Tasks

const

- If you want to make sure that a variable is not changed you can use the `const` keyword
- Ex: `const double pi = 3.1415;`

`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;`

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 {
  char *name;
  int age;
};
struct person p1 = {"Jan Kowalski", 38};
```

- Order must be same as in structure, unless:

```
struct person p1 = {.age=38, .name="Jan
Kowalski"};
```

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;
```

- Now you can use pos instead of struct position

Lecture 10: Pointers and Structures

Wrap Up

Pointers Continued

Function Pointers

Constant variables and structs

Pointers and Structs

C Tasks

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;`
- This is the same as `x.real = 2;`

Lecture 10: Pointers and Structures

Wrap Up

Pointers Continued

Function Pointers

Constant variables and structs

Pointers and Structs

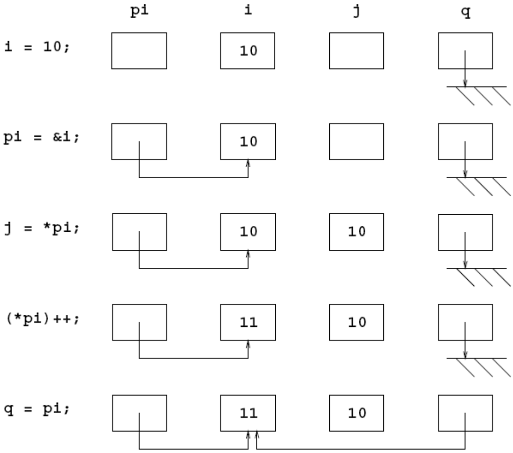
C Tasks

Task 2

- Illustrate what happens in the following case

```
int *pi, i, j, *q = NULL;
i = 10;
pi = &i;
j = *pi;
(*pi)++;
q = pi;
```

Task 2 cont'd



```
i = 10;
pi = &i;
j = *pi;
(*pi)++;
q = pi;
```

Task 3

Write a program which accesses the functions,

- `int add(int x,int y){return x+y}`
- `int mul(int x,int y){return x*y}`

using function pointers

Task 4

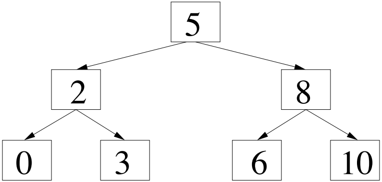
- Rewrite the Newton function so that it can take a function pointer instead
- This makes it easier to switch function

Task 5

- Write a program with several functions, all with the same interface
- Create an array of pointers to these functions
- Loop through the pointers and call the functions

Task 6

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

Next Time

- Lecture
  - Continue with structs
  - Memory Handling
- Homework: Do C lab!!!