EL2310 – Scientific Programming

Lecture 10: Memory, External Input, File and Bitwise Operations

Hakan Karaoguz (hkarao@kth.se)

Royal Institute of Technology – KTH

Overview

Lecture 10: Memory, External Input, File and Bitwise Operations

- Announcements
- Wrap Up
- External Input
- File Operations
- Bitwise Operations

Annonuncements

- C assignment $\rightarrow$ deadline 30^{th} September
- Help Session $\rightarrow$ 27^{th} September @Teknikringen 14 Room 304 09:30-12:00
- Course Webpage $\rightarrow$ `https://people.kth.se/~hkarao/el2310/el2310.html`

Lecture 10: Memory, External Input, File and Bitwise Operations

Announcements
Wrap Up
External Input
File Operations
Bitwise Operations

Last time

- Complex data structures (`struct`)
- Memory

Today

- More on Memory
- External Input
- Reading/writing files
- Bitwise operations

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;
};
```

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;` (cast from int to double)
- In other cases you need to tell the compiler to do this
- Ex: `int a = (int)4.2;` (will truncate to 4)
- Often used together with pointers
- Ex:

```
int a;
unsigned char *byte = (unsigned char*)&a;
```

Dynamic allocation of memory

- Sometimes you do not know the size of arrays etc.
- Idea: Allocate memory dynamically
- This way you can allocate memory at runtime

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));`
- To avoid warnings, add explicit cast
  Ex: `int *p = (int *)malloc(100*sizeof(int));`
- Will allocate memory for 100 `int`s

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)

Lecture 10: Memory, External Input, File and Bitwise Operations

Announcements
Wrap Up
External Input
File Operations
Bitwise Operations

Command line arguments

- `int main(int argc, char **argv)`
- Arguments are seperated by spaces: `./test 12 john`
- **argc**: argument count **argv**:argument vector
- If nothing is inputted, then argc is 1 and argv[0] is the name of the executable
- `atoi` and `atof` are useful to get number from char arrays
- Ex:

```
int value;
...
if (argc > 1) value = atoi(argv[1]);
else value = 42;
```

Reading from the keyboard

- Can use `char getchar();` to get a single character
- For more more complex input try `scanf(...)` which is the "dual" of `printf(...)`
- The arguments for `scanf` the same as for printf except that it wants pointers to where to put the data
- Ex:

```
int i;
double num[3];
printf("Enter 3 numbers:  ");
for (i = 0; i < 3; i++) {
  scanf("%lf", &num[i]);
}
```

Task 1

- Write a code that accepts number of people as a command line argument
- If the argument is not given the program should quit with a warning
- Create a structure named person with members *firstname*, *lastname* and *age*
- Based on the inputted number of people the program should get the following input from keyboard *firstname, lastname, age* and store those as a vector of person

Lecture 10: Memory, External Input, File and Bitwise Operations

Announcements
Wrap Up
External Input
File Operations
Bitwise Operations

Opening/closing a file

- `FILE *fopen(char *path, char *mode);`
- mode is “r”: read, “w”: write, “a”:append, . . .
- On success returns pointer to `file descriptor`, else NULL
- `fclose(FILE*);`

Writing to a file

- Write to the file with for example
- `fprintf(FILE*, ...);`
- Ex:

```
double x=1, y=2, theta=0.5;
FILE *fd = NULL;
fd = fopen("test.txt", ``w'');
if(fd){
  fprintf(fd, "Robot pose is %f %f %f\n",
x,y,theta);
  fclose(fd);
}
```

Task 2

- Add a function to the previous code which opens a file called *person.txt* and writes the data of the each inputted person as a row.
- Each attribute of a person should be separated with a space or tab.

Reading from a file

- Read from the file with for example
- `fscanf(FILE*, ...);`
- Ex:
```
double x,y,theta;
FILE *fd = NULL;
fd = fopen("test.txt", "r");
fscanf(fd, "Robot pose is %lf %lf %lf\n",
&x,&y,&theta);
fclose(fd);
```
- Notice that you need `%lf` when you read a double, `%f` for a float
- Function `sscanf()` is similar but operates on a char array instead of a file

Task 3

- Write a code that accepts age as an input (command line or keyboard depends on your choice)
- It reads the previously created *person.txt* file and checks if there are any people that are at the same age as the input
- If it finds then it outputs the information about these people otherwise it outputs *No record found!* and quits.

Lecture 10: Memory, External Input, File and Bitwise Operations

Announcements
Wrap Up
External Input
File Operations
Bitwise Operations

Bitwise operations

- When programming at low level, bitwise operations are common (Embedded programming using Microcontrollers, Arduinos etc.)
- Memory efficiency. In C all the variables are multiples of bytes (1 byte = 8 bits = [0-255]) but for flags you just need 1 bit.
- Typical construction, use *bitmask*
- The operators do not change the actual value of the variable, they create a temporary variable.

Bitwise operators

- `&` bitwise AND
- `|` bitwise inclusive OR
- `^` bitwise exclusive OR
- `<<` left shift
- `>>` right shift
- `~` bitwise NOT

Example of bit operations

- `mask = mask & 0xF` Set all but the lower 4 bits to zero ($0xF = 1111$) $\rightarrow$ Each hex(0x) digit is 4 bits
- `mask = mask | 0x3` Set lower 2 bits $0x3 = 11$
- ```
  short value;
  ...
  unsigned char lower = (value & 0xFF);
  ```
  ($0xFF = 11111111$)
  ```
  unsigned char upper = (value >> 8);
  ```
- What is printed?
  ```
  int x = 1, y = 2;
  if (x && y) printf("Case 1\n");
  if (x & y) printf("Case 2\n");
  ```
  ```

Shift operators >> and <<

- Should primarily be used on `unsigned` data types
- Shifting results in division (right) and multiplication (left) of integers by 2 times the number of shifts
- Exm: unsigned int b = a<<n where n is the number of shifts to the left

Task 4

- Assume you have a car with auto-AC control. You have an 8 bit array that controls 5 switches. What should be the bitmask and bit operation for running the AC correctly?

Master Switch	Airbag Activate	AC Motor	Wiper Motor	Indoor Temperature Sensor

- Investigate the difference between logical and bitwise operators
  - Assume x = 0 and y = 1
  - Evaluate `if(x+y == 1 || y/x == 1)` and `if(x+y == 1 | y/x == 1)`
- Evaluate the binary operators on integers such as 32 and 16.
  - What is the result of 32^16, 32&16, 32|16 and~16?
  - What is the result of $32 >> 1 + 16$ and $16 << 1 + 32$?