EL2310 – Scientific Programming

Lecture 11: Memory, Files and Bitoperations

Yasemin Bekiroglu
(yaseminb@kth.se)

Royal Institute of Technology – KTH

Overview

Lecture 11: Memory, Files and Bit operations

- Wrap Up
- Main function; reading and writing
- Bitwise Operations

Lecture 11: Memory, Files and Bit operations

Wrap Up

Main function; reading and writing

Bitwise Operations

Last time

- Complex data structures (`struct`)
- Memory

Today

- More on Memory
- 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)

Tip when using dynamic memory allocation

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

Lecture 11: Memory, Files and Bit operations

- Wrap Up
- Main function; reading and writing
- Bitwise Operations

Command line arguments

- You add parameters to the `main` function
- `int main(int argc, char **argv)`
- First argument is in `argv[1]`, `argv[0]` contains program name
- `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 number:  ");
for (i = 0; i < 3; i++) {
  scanf("%lf", &num[i]);
}
```

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'');
fprintf(fd, "Robot pose is %f %f %f\n",
x,y,theta);
fclose(fd);
```

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

Lecture 11: Memory, Files and Bit operations

Wrap Up

Main function; reading and writing

Bitwise Operations

Bitwise operations

- When programming at low level, bitwise operations are common
- Also, if you want to store flags it is very wasteful to use 1 byte for every flag that can only be 0 or 1.
- Typical construction, use *bitmask*
- Let each bit in the variable be one flag

Bitwise operator

&	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 $(0xF = 1111)$
- `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

- 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