**Week 4 Notes**

string s = GetString();

* If we feed Serge to GetString(), what GetString returns is the address of ‘S’ because that’s all we need. The way that the program knows what the full String is is it starts at the address of ‘S’ and pulls every address of each subsequent +sizeof(char) until ‘\0’.

strcmp (a, b)

* Returns positive or negative value (greater than or less than).

char\* s

* \* is the address

**FILE I/O**

FILE\* fp = fopen(argv[1], “w”)

* First argument is the name of the file (command line)
* w – write (overwrite other data stored)
* r – read
* a – appending to the file.

fputs, fputc, fwrite -> write a string to the file.

* (input, fp)
  + What needs to be written “input”, and “fp”, where to write it.

fclose(fp)

* Closes file

fgets(output, sizeof(output), fp)

* output – pointer to where string stored
* sizeof(output) - maximum characters to be read
* fp – file for which we are working.
  + Ex. fgets (output, sizeof(output), fp) != NULL 🡨 Not end of file or not enough memory

while(!feof(fp))

* Incorrect.

**GDB**

GNU Project Debugger

* Type ‘gdb’ in console.

Can’t run it on .c files, need to run it on the executable file. Thus:

* gdb ./factorial

breakpoint – pause the program at a certain point to examine state

* ex. break main. (or b main)

next – moves to next line

list (l) – prints source code centred around line we’re currently on

* l 12 – moves back to line number 12

print – prints the value of a variable at the line we’re currently paused on

* print ‘num’ where num is some variable storing an int

info

* info locals – prints value of all in scope local variables

disable

* disables all breakpoints

**MERGE SORT**

Compare elements of each, which is less, put into new array.

* O (n)

1st step:

* Split them until there is a single cup on each list.

2nd step:

* Repeatedly merge the individual (less on left, higher on right)

3rd step

* Merge those lists again.

How fast?

* O (n log n)

**POINTERS**

* &n “address of” operator.
* int\* pointer\_to\_n -> pointer type.
* \*pointer\_to\_n 🡪 dereference operator.
* int \*\* double\_pointer = &pointer;
  + since double\_pointer is an address to a pointer, its type int\*\*.
* \*\*double\_pointer -> dereference twice.

NULL pointer – do not dereference

**RECURSION**

Calling same function again. Need to have a base case (to terminate the function calls).

**STRING**

* \0 null termination character at the end of each string.
  + Signals the end of a string in memory.
* String we’ve been using is actually “char\* s”
* Each char is stored consecutively at an address in memory divided by the size of the char in the system you are using. Therefore a char variable points to the memory of the first char and the program knows the end of that string when it is called by the first terminating character \0.

**STRUCT**

Ex. Struct Student

{

int age;

char\* name;

} S1;

* Names of the variables in the struct are “members”
* Struct student s2 = {3, “S2”};

Typedef Struct

{

int age;

string name;

} student;

* Can now create variables by just saying student bob.
* **Recursive** structs need to have the node in the typedef because resursive doesn’t know yet that node is a typedef struct.

Dereference to members

* \*s.age = 4; 🡨 wrong because it considers it as \*(s.age) which is nowhere in memory.
  + Fix is (\*s).age **OR** s->age = 4;

If structs depend on each other, can forward declare one struct so program compiles.

Struct b; 🡨 **FORWARD DECLARATION**

Struct a

{

Struct b\*x;

};

Struct b

{

Struct a \* y;

}