**ASSIGNING STRING**

There is no difference between these 2. c and s are pointers.

There is a difference in semantic. First one is array. Unless you assign sth to second one, it is not array.

In first one, an array is created in local memory, referred to that array with s and copy “abcd” to that array. Array and s are in the stack.

In second one, “abcd” creates an array, “abcd” is not copied, address of that array is copied to c. Array and c are in the stack.

char s[] = “abcd”;

char \*c = “abcd”;

**HOW PRINTF - SCANF KNOW HOW MANY ENTRIES THEY WILL GET?**

Prototype of printf: int printf(const char \*format, ...);

printf looks what is the format in the runtime, and it will have list of arguments.

**ONE REASON FOR A ZOMBIE**

struct movie \* func(){

struct movie \*temp = ...malloc()...;

return temp;

}

void fcall(){

struct movie \*l;

func();

}

I didn’t assign the return value which is allocated in func so I cannot free it from now on. So it became a zombie.

**MEMORY**

Programs cannot exceed their memory limits that given to them in the beginning. There is an upper limit. Actual limit (lower limit) that is imposed to program depends on compiler and compiler settings. Default setting on your program will have a limited stack and upper limit heap. You need to read your compiler for your system. That is OS specific which is onto hardware so also hardware specific.

If there is a limitation emposed by your OS and still it is not as much as you want, then there are some other tricks that you can do. One of them is creating multiple processes and you communicate with these processes. You will need more memory, you will need more disc space… so you can work with your OS and you can exploit your OS’s helps and advantages and disadvantages. In modern systems, you don’t have any limitations. These all depend on the OS and compiler settings.

**CHAR POINTER**

You can't copy a string to a pointer, but you can store the pointer of the string in a char pointer. The pointer doesn't have enough memory to store the whole string.

**STRING IN STRUCT**

node \*t = malloc(sizeof(node));

char \*fname;

char sname[15];

...

t->name = calloc(1, strlen(fname) + 1);

strcpy(t->name, fname);

...

strcpy(t->name, sname);