simQ parameters01.txt 1000 results01.txt

**Must include -ansi argument within the compile script called compileSimQ**

**maxQueueLength can be -1 in which case the queue has no maximum length**

**numServicePoints must be greater than zero and can only serve one customer at a time**

**The source code must be written in well-structured, ANSI standard C.**

**There must be a Linux shell script for compiling and linking your code, called compileSim and nothing else, and must compile your code to an executable called simQ and nothing else**

**You must include one example input parameter file, called testInput.txt and nothing else.**

**You must include a max four-page report, as detailed in the previous section.**

Slide Number 14 L02-3\_functionsIntro

Generally it is **VERY** bad practice to use global variables.

**int globalInteger = 0;**

void printInteger (); **/\* function prototype \*/**

int main ()

{

printf("globalInteger = %d\n",**globalInteger**);

printInteger();

return 0;

}

void printInteger ()

{ printf("globalInteger = %d\n",**globalInteger**); }

Slide Number 39 of L03-1\_pointers

You must **ALWAYS** trap the fact that memory was not allocated:

**if ( !(** p = (int \*)malloc(100000000000000\*sizeof(int)) **) )**

{

printf("Out of memory\n");

exit(1);

}

Slide Number 42 of L03-1\_pointers

Whenever you use malloc, you must **ALWAYS** explicitly free the memory before the variable goes out of scope:

void myFunction ()

{

int \*p = (int \*)malloc(1000000);

**free(p);**

}

Freeing memory that has already been freed, or memory not allocated by

malloc (e.g. “char str[20];”): “behaviour is undefined".

Slide Number 47 of L03-2\_functionParameters

If we want to pass parameter values in from the command line, then we must define it like this:

int main ( **int argc**, **char \*\*argv** )

Technically the variable names can be anything, but they are **ALWAYS** (by **VERY** strong convention), argc and argv.

Slide Number 13 of L04-1\_fileStreams

You must **ALWAYS** test that opening the file was successful:

fp = fopen("myTextFile.lis", "r");

**if ( fp == NULL )**

{

**/\* do some error processing \*/**

}

usually compressed into:

if ( (fp = fopen("myTextFile.lis", "r")) == NULL )

{

**/\* do some error processing \*/**

}

Slide Number 23 of L04-1\_fileStreams

You must **ALWAYS** send error messages to stderr and not stdout

When your program is run from the Linux command line it is likely, in a real system, to be run like this (**$** is the Linux prompt):

**$** myTestProgram **>** out.lis **2>** err.lis

The first redirection (**>**) sends the **stdout** output to a file called out.lis.

The second redirection (**2>**) sends the **stderr** output to a file called err.lis.

**$** cat out.lis

**Unable to open file for read access.**

**$** cat err.lis

**error 13: Permission denied**

To send both streams to the same output file:

**$** myTestProgram **>** out.lis **2>&**

Therefore we can divert the two different types of outputs to two different files.

Slide Number 10 of L04-3\_programStructure

The header (\*.h) file is **ALWAYS** named after the code file (\*.c) it belongs to.

Slide Number 19 of L04-3\_programStructure

The executable is **ALWAYS** named after the program module containing the main function.