Final Exam Review

Review Questions

[canvas] | *darkGreenBackground.jpg*

1. What is function prototype for the entry point in a C program?
    - ANSWER: `int main()` or `int main(int, char **)`

2. What function is used to allocate memory on the heap given a specific number of bytes?
    - ANSWER: `malloc`

Review Questions

1. What part of the compiler copies include files into the source file?
    - ANSWER: Preprocessor

2. Which loop will always execute at least once?
    - ANSWER: A `do {...} while` loop

Compiler and syntax:

- C is case-sensitive
- Usually your compiler will complain (`-Wall`)
- Don't ignore or supress compiler warnings
- Always validate the return value
- Entry point is generally one of two prototypes:
    - `int main()`
    - `int main(int argc, char *argv[])`
- Successful return code is `0` while errors return non-zero

Preprocessor

- #include <library.h>
- #ifdef
- Always wrap macro code in parenthesis `()`
- Macros:

```
#define maxNumber(a, b) ( (a > b) ? a : b  )
maxVal = maxNumber(userValue, storedValue);
```

Preprocessor

```
#define findMax(list, size, max) \
( max = -9999; \
  for (int i = 0; i < size; i++) \
  { \
    if (list[i] > max) \
    { \
      max = list[i]; \
    } \
  } \
)
int mylist[] = {12, 45, 1, 23, 78, 145, 61};
int listMax;
findMax(myList, sizeof(mylist)/sizeof(int), listMax);
```

Data types and values:

- `char`, `short`, `int`, `long` and those with `unsigned` will not hold decimals
- `float` and `double` can hold decimals
- Specify hexadecimal notation using 0x123ABC.

Data types and values

- Bit-setting operations
  - To set a bit in a variable, use the bitwise OR `|` operator
    `unsigned int defOdd = someNumber | 0x1;`
  - To clear a bit in a variable, use the bitwise AND `&` operator
    `unsigned int itsEven = someNumber & 0xFFFFFFFE;`

Data types and values

- Bit-shifting operations
  - To move bits to the left, use `<<` operator
    `unsigned int newNum = someNumber << 0x4;`
  - To move bits to the right, use `>>` operator
    `unsigned int anonNum = someNumber >> 2;`

File IO

- `FILE` type is used to store open stream pointers
- `fopen` is used to open a file

- `fclose` is used to close a file
- `fread` or `fgets` can be used to read data from the open stream
- Three standard streams: `stdin`, `stdout`, `stderr` are treated as files
- `fputs` to write to a file

File IO Example

```
#include <stdio.h>
int main()
{ FILE *inFile = fopen("test.dat", "r");
  if (inFile != NULL)
  { char buff[128];
    while (fgets(buff, 128, inFile) != NULL)
      printf("%s", buff);
    fclose(inFile);
  }
  return 0;
}
```

Strings and Memory

- Copy a string:
  `strcpy(char *dest, char *src)`
- Add to the end of a string:
  `strcat(char *dest, char *src)`
- Copy a string but only up to N-bytes:
  `strncpy(char *dest, char *src, size_t len)`
- Length of a string:
  `strlen(char *str)`

Strings and Memory

- Linefeed escape sequence: `"\n"`
- Copy an area of memory to another area:
  `memcpy(void *dest, void *src, size_t len)`
- Fill an area of memory to a value:
  `memset(void *dest, int value, size_t len)`

Loops

- `do { … } while(condition);`
  Executes at least one time

- `while (condition) { … }`
  Executes 0 or more times
- `for (int i = 0; i < N; i++) { … }`
  Executes N times
- Nested loops

Functions:

- Prototype/signature:
  Return type, name, and parameter type/order are important
- Return type or void
- Always validate your input pointers
- Check for NULL pointer:
  `if (ptr == NULL)`
- Functions perform one task over a set of instructions

Arrays:

- Define arrays with `[]` or `*`
- Access array elements `[]`
- Accessing multidimensional arrays
  `array[row][col]`

Dynamic memory allocation:

- Functions used: `malloc`, `calloc`, `realloc`, and `free`
- Check the return pointer for `NULL`
- Be sure to `free` the memory for which you are responsible

Pointers

- A normal variable has 0 levels of indirection/access the value directly:
  `int value = 7;`
- A pointer is one level of indirection:<br> <code>int *ptr = &value;</code>
- To access the value of a pointer, use one level of indirection:
  `*ptr = 3;`

Pointers

- A pointer to a pointer is two levels of indirection:<br> <code>int **dblptr = &ptr;</code>

- To access the value of a double pointer, use two levels of indirection: `**ptr = 1;`

Sorting Arrays

- qsort is part of the standard library `stdlib.h` and implements the quick sort algorithm

```
void qsort(void *base, size_t nmemb, size_t size,
           int (*compar)(const void *, const void *));
base   - start of the array
nmemb  - number of elements in the array
size   - size of an array element
compar - function pointer for comparing the elements
         returns 0 if p1==p2, >0 if p1>p2, <0 if p1<p2
```

Sorting Arrays Example

```
#include <stdio.h>
#include <stdlib.h>
int pSort(const void *p1, const void *p2)
{
  int ip1 = *(int*)p1, ip2 = *(int*)p2;
  return ip1 - ip2;
}
int main()
{ int arr[] = {25, 12, 14, 99, 26, 42, 71};
  qsort(arr, sizeof(arr)/sizeof(int), sizeof(int), pSort);
  for (int i = 0; i < sizeof(arr)/sizeof(arr[0]); i++)
  {
    printf("%d\n", arr[i]);
  }
  return 0;
}
```

Structures:

- How do you define them?
- Order is important/determines memory layout.

```
struct vector
{
  int a;
  int b;
};
```

Command Line Arguments

- `int main(int argc, char *argv[])`
- For `getopt` include unistd.h
- int getopt(int argc, char *const argv[], const char *optstring);
- char *optarg - The option for an argument
- int optind - The next index to be read.
- int opterr - Set to 1 to print error message/0 no message
- int optopt - Option with an error

Command Line Arguments

- Format string:
  - "s" - flag only
  - "s:" - flag and required argument
  - "s::" - flag and optional argument
- Returns:
  - Current flag on success
  - '?' - Error processing argument
  - -1 - No more arguments to process (positional arguments possible)

Command Line Arguments

```
#include <stdio.h>
#include <unistd.h>
int main(int argc, char *argv[])
{ int c, logOutput = 0, verbose = 0;
  char *logFile = "default.log", *output = NULL;
  while ((c = getopt(argc, argv, "vl::f:")) != -1)
  { switch(c)
    { case 'l': logOutput = 1; if (optarg) logFile = optarg; break;
      case 'v': verbose = 1; break;
      case 'f': output = optarg; break;
      case '?': break;
    }
  }
  if (verbose) printf("Verbose\n");
  if (logOutput) printf("Log output to %s\n", logFile);
  if (output) printf("Output to %s\n", output);
  for (int i = optind; i < argc; i++)
    printf("Input File: %s\n", argv[i]);
  return 0;
}
```

Manual Pages

- Use man for function prototypes
- `man string`
- `man stdio`
- `man qsort`
- `man getopt.3`

General:

- Follow instructions
- Automated grading is being used
- No complicated algorithms are required
- Don’t change any function prototypes
- Add your code where instructed
- Free memory where instructed
- Output what is requested
- 2 hours given