Mechanics of Programming:
Exam 1 Review
September 18, 2021

1. The C programming language is a (circle one) (functional/imperative/object-oriented) programming language.

   Imperative: a programming paradigm that describes computation in terms of statements that change a program state.

2. What are the gcc commands for compiling a single C source file, hello.c, into an executable named hello?

   To compile but do not link - gcc -c hello.c
   To compile into an executable named **hello** - gcc -o hello hello.c

3. How do you fix the code fragment?

```
1     for (int i = 0; i < 10; ++i) {
2         printf(i);
3         //printf("%d\n",i);
4     }
```

4. What is the purpose of header files? How do we prevent header files from being included multiple times?

   Header files break implementation into multiple parts, as a contract between client and supplier. It allows you to prototype the function before it is used by the compiler.

   Use the preprocessor directive:
   #ifndef _SOMETHING
   #define _SOMETHING

...
#endif

It also allows you to re-use the code between multiple files easily.

5. Number the following in order of occurrence:

   _____ Link

   _____ Preprocessor

   _____ Assemble

   _____ Compile

   Preprocessor -> Compile -> Assemble -> Link

6. Describe the following with regards to variable storage:

   static

   const

   stack

   heap

   static: In the C programming language, static is used with global variables and functions to set their scope to the containing file. In local variables, static is used to store the variable in the statically allocated memory instead of the automatically allocated memory. While the language does not dictate the implementation of either type of memory, statically allocated memory is typically reserved in data segment of the program at compile time, while the automatically allocated memory is normally implemented as a transient call stack.

   const: used to declare a constant in C

   stack: uninitialized local variables

   heap: uninitalized dynamic variables

7. What is the difference between passing by reference and passing by value?

   Pass By Value

   (a) The local parameters are copies of the original arguments passed in
   (b) Changes made in the function to these variables do not affect originals

   Pass By Reference

   (a) The local parameters are references to the storage locations of the original arguments passed in.
   (b) Changes to these variables in the function will affect the originals
   (c) No copy is made, so overhead of copying (time, storage) is saved

8. Explain `argc` and `argv` in `int main(int argc, char* argv[])`?

   argc (argument counter) is the number of command line arguments while argv (argument values) is the values stored in them.

9. Is there difference between `i++` and `++i`? If yes, what is it?

   Yes, i++ is post increment (increments after assignment) and ++i is pre increment (increments before assignment)
   Pre-increment operators increments the value of the object and returns a reference to the result.
   Post-increment creates a copy of the object, increments the value of the object and returns the copy from before the increment.

10. True/False

    (a) ___ The first value in an array `arr` is `arr[0]`
    (b) ___ An uninitialized `int` is guaranteed to have a value of 0.
    (c) ___ A program sees the computer's physical address space.

    TFF

11. What is the output of the following program, `main.c` when executed?

```
1     #include <stdio.h>
2
3     int main(int argc, char* argv[]) {
4         int max = 10;
5         for (int i = 1; i < max; ++i) {
6             printf("i= %d\n", i);
7             break;
8         }
9         return 0;
10    }
```

i= 1

12. What is the output of the following program, `main.c` when compiled into a binary of the same name?

```
1     #include <stdio.h>
2
3     int main(int argc, char* argv[]) {
4         printf("argc: %d\n", argc);
5         for (int i = 1; i < argc; ++i) {
6             printf("argv[%d]: %s\n", i, argv[i]);
7         }
8         return 0;
9     }
```

(a) `$ ./main a b c d e`

argc: 6
argv[1]: a
argv[2]: b
argv[3]: c
argv[4]: d
argv[5]: e

(b) `$ ./main a "b c d" e`

argc: 4
argv[1]: a
argv[2]: b c d
argv[3]: e

13. Will the following code compile? If yes, what is the output, if not why?

```
1     #include <stdio.h>
2
3     int main(int argc, char* argv[]) {
4         char* s = "Hello World!"
5         printf("%s\n", s);
6         return 0;
7     }
```

No, Line 4 - no semicolon

14. What is the name of the configuration file used for automating the compilation process? What is the command to run the configuration file?

    Makefile, make

15. Given the following code, draw a picture of the variables immediately before the return statement. If the value of a variable is unknown, put a question mark.

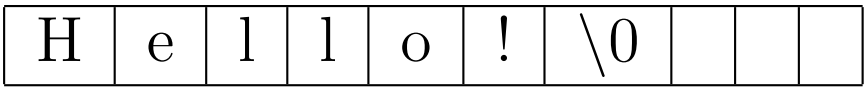
```
1     #include <stdlib.h>
2     #include <stdio.h>
3
4     int main(int argc, char* argv[]) {
5         int x = 5;
6         int y;
7         int arr1[] = {1, 2, 3};
8         getchar();
9         return 0;
10    }
```

    STACK
    x - 5
    y - ?
    arr1 - [1—2—3]

16. Given the following code fragment:

```
1     char str[10];
2     strcpy(str, "Hello!");
```

    (a) Fill in the values for `str`, using '?' if the value is unknown.



(b) In general, which would we rather use: `strcpy` or `strncpy`? Why?

strncpy requires a length as input instead of stopping at a null character. This protects against buffer overflow.

(c) In general, what does `strtol` do?

long int strtol (const char* str, char** endptr, int base);
Parses the C-string str interpreting its content as an integral number of the specified base, which is returned as a long int value. If endptr is not a null pointer, the function also sets the value of endptr to point to the first character after the number.