Programming Skills: C++ **Homework 1: C Review**

Due Tues 1/22 (start of clas)

Name: _____

1. **(Makefiles)** Assume you have the following C source files:
   - `derp.c`: The main program which includes `parser.h`.
   - `parser.c`: A utility file that includes `parser.h` and `symtbl.h`.
   - `symtbl.c`: Another utility file that include `symtbl.h`.

   Using `gcc`, write a simple makefile that captures all the required dependencies. The user should be able to compile a source file, producing debugging information, into an object file by issuing the command (where `src` is either `derp`, `parser` or `symtbl`):

   ```
   make src.o
   ```

   The user should be able to recompile any source code that changes and link all object files into an executable named `derp` when issuing the command:

   ```
   make derp
   ```

   In addition, the user should be able to remove all generated object files with the command:

   ```
   make clean
   ```

   Finally, the user should be able to remove all object files, as well as the executable, with the command:

   ```
   make realclean
   ```

2. **(Debugging)** When the program from the previous question is run, it produces the following error:

```
$ ./derp
Segmentation fault (core dumped)
```

(a) Assume the program is run on the command line as:

```
$ gdb
```

What is the `gdb` command that will load the executable file?

(b) What is the `gdb` command you would use to start the debugged program?

(c) The program runs and you now see the following:

```
Program received signal SIGSEGV, Segmentation fault.
0x00000000004006c9 in createTable() at derp.c:26
26     printf("%d\n", *p);
```

What is the `gdb` command you would use to print a backtrace of all stack frames?

(d) Now that you see the backtrace, what `gdb` command will select the top stack frame and print the offending statement that is causing the segmentation fault?

(e) What is the `gdb` commandt that will print out the source lines around and including the offending statement?

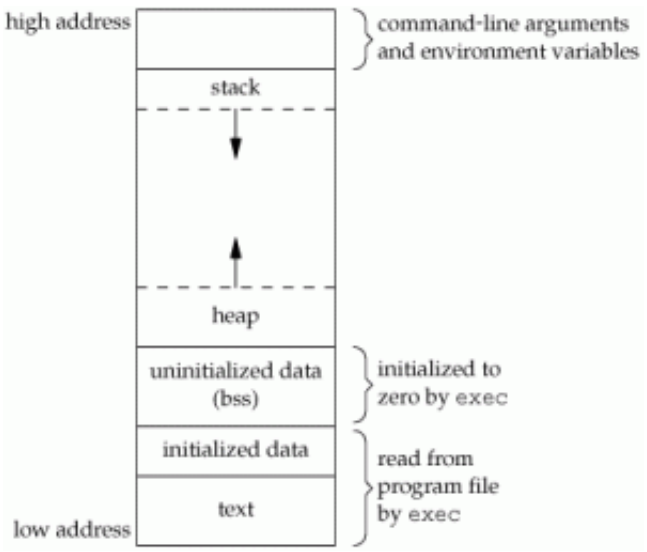
(f) What is the `gdb` command that will print the value stored in the pointer `p`?

(g) What is the `gdb` command that will set a breakpoint at the start of the `createTable` function?

(h) What is the `gdb` command that will have the debugger execute the next 3 source lines?

(i) What is the `gdb` command that will continue executing the program until the next breakpoint (or end of program)?

3. **(Memory Layout)** Consider a C program whose executable image is loaded into the memory area of the processes address space as follows:



For each of the following segments, give an example of a compiled C statement that would be located in that segment when run. Be sure to state, if necessary, any assumptions you make about the scope the statement appears in, e.g. global or local.

(a) **stack**

(b) **heap**

(c) **uninitialized data (bss)**

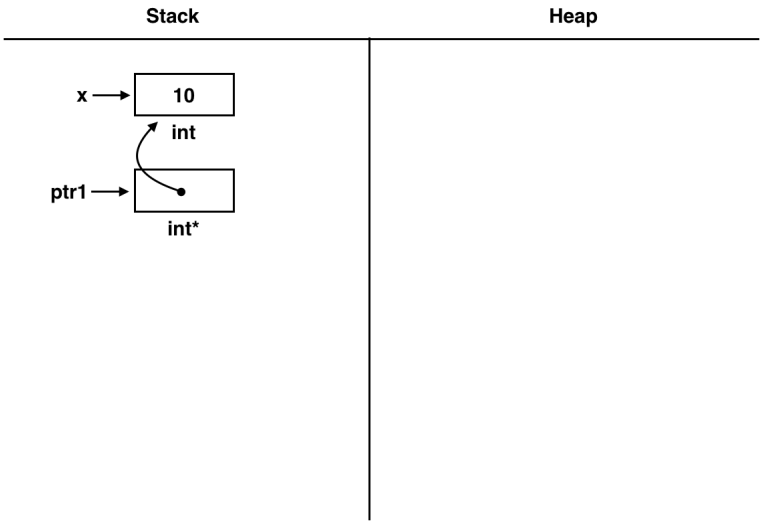
(d) **initialized data**

(e) **text**

4. **(Memory Diagrams)** Given the following C declarations:

```
int x = 10;
int y[3];
int* ptr1 = &x;
int* ptr2;
int* ptr3 = (int*) malloc(sizeof(int));
int* ptr4 = (int*) malloc(sizeof(int) * 3);
int** ptr5 = &ptr2;
```

(a) Complete this initial memory diagram. If a variable's value is undefined use `???` as the value.



(b) Assuming the following assignments are made, re-draw the complete final memory diagram.

```
*ptr1 = 20;
ptr2 = &y[1];
ptr1 = ptr3;
ptr3 = 0;
*ptr1 = 30;
**ptr5 = 40;
*(ptr4+2) = 50;
```

(c) If the program was analyzed by `valgrind`, how many bytes of heap allocated space would be reported as leaked (assume an integer is 4 bytes)?

(d) What statement/s need to be added before the program completes to prevent the memory leak?

5. **(Alignment)** Assume you are working with a C program on a 64 bit machine with the following data sizes in bytes:

```
sizeof(char): 1
sizeof(short): 2
sizeof(int): 4
sizeof(long): 8
sizeof(int*): 8
```

Here are the local variable declarations, in order:

```
short a;
char b;
long c;
char d;
int e;
int* f;
short g;
char h;
```

As the local variables are encountered by the compiler they are pushed onto a stack. Next, they are popped off to be layed out in memory (from low to high address) as efficiently as possible. However, the starting address of a variable cannot be arbitrarily aligned, e.g. a `short` must start at an even address (divisible by 2), an `int` must start at an address divisble by 4, and so on.

Assume the following:

- the starting address of the last variable, `h`, is `0x7fff5033ca4d`.
- if padding occurs, the lower memory address/es are padded.

List the starting address (last two bytes) for each of the remaining variables:

- `h`: `0x...4d`
- `g`: _____
- `f`: _____
- `e`: _____
- `d`: _____
- `c`: _____
- `b`: _____
- `a`: _____

6. **(Bitwise Operations)** Write a function, `bits`, that takes an unsigned integral value and extracts `n` bits from the value starting at position `pos` (assume the rightmost bit is position 0). For example:

```
val = 54 = 00110110 (base 2)

unsigned bits(unsigned val, int pos, int n):
bits(val, 5, 3) = 00000110 (base 2) = 6
bits(val, 3, 2) = 00000001 (base 2) = 1
bits(val, 6, 4) = 00000110 (base 2) = 6
```

You should use bit shifting and bit masking. Hint: it is possible to do this in one line of code.