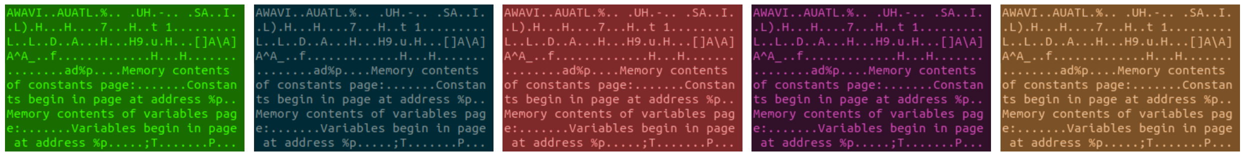
CMPSC 400
Operating Systems
Spring 2020

Participation 1: Interactive Demonstration of Virtual Memory



```
AWAVI..AUATL.%.. .UH.-.. .SA..I.
.L).H...H....7...H..t 1.........
L..L..D..A...H...H9.u.H...[]A\A]
A^A_..f.............H...H.......
........ad%p....Memory contents
of constants page:.......Constan
ts begin in page at address %p..
Memory contents of variables pag
e:.......Variables begin in page
 at address %p.....;T.......P...
```

Figure 1: The computer's physical memory and disk space may be mapped to produce virtual memory. This virtual memory is then able to be applied to running (and sleeping) processes by the OS in more adaptive ways.

Summary

In this participation, you are given an opportunity to spend some time at a Robert Elder's blog titled, *Virtual Memory With 256 Bytes of RAM - Interactive Demo* where you can gain a deeper understanding of how virtual memory is mapped from physical memory. In this demonstration, you are able to interact with the page settings to experiment and test outcomes of memory mapping. You will have questions to respond to, as well as some code from the blog that you are invited to run and test to enrich your understanding.

GitHub Starter Link

General participation Repository
`https://classroom.github.com/a/S8lbI9Z5`

To use this link, please follow the steps below.

- Click on the link and accept the assignment.
- Once the importing task has completed, click on the created assignment link which will take you to your newly created GitHub repository for this lab.
- Clone this repository (bearing your name) and work on the practical locally.
- As you are working on your practical, you are to commit and push regularly. You can use the following commands to add a single file, you must be in the directory where the file is located (or add the path to the file in the command):
  - `git commit <nameOfFile> -m ‘‘Your notes about commit here’’`
  - `git push`

  Alternatively, you can use the following commands to add multiple files from your repository:

  - `git add -A`
  - `git commit -m ‘‘Your notes about commit here’’`
  - `git push`

What TODO

Your tasks are outlined below.

Please visit Robert Elder's Blog
(`https://blog.robertelder.org/virtual-memory-with-256-bytes-of-ram/` and spend a moment to play with the author's interactive demo of virtual memory. Once you have tinkered with all the settings and believe that you have a working idea what the demonstration is showing, you are to address the Questions-In-Blue below.

Questions-In-Blue

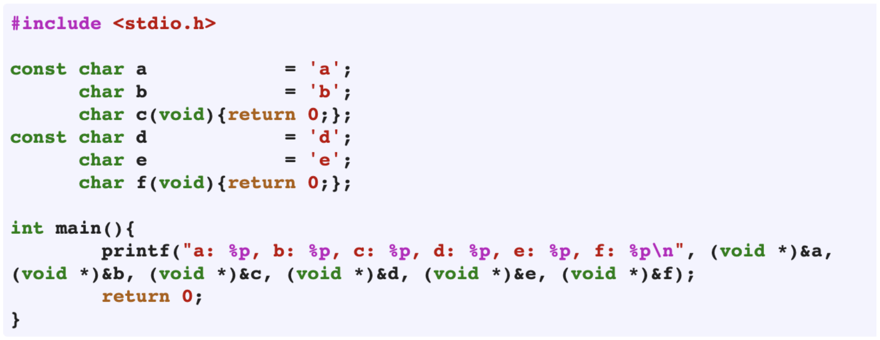
Below are the same questions that you will find in your `reflection.md` file along with this assignment sheet in your `classDocs/` repository.

Please make the directory `01part/` in your participation repository using the command, `mkdir 01part/` and be sure to place your completed `reflection.md` file inside this directory.

The below questions have been designed to be thought-provoking and may not necessarily have one correct answer. Please do your best to answer them. Each question can be answered in a few sentences.

1. The following questions concern the interactive demo in at the top of the blog's page.
   (a) In your own words, using clear and meaningful language, please describe the demonstration in its entirety. How does this demonstration show how physical memory is mapped to virtual memory?
   (b) In clear and meaningful language, describe the function of the demonstration's main actors such as;
      i. The *Physical pages*,
      ii. The *Table pointers*,
      iii. The *Complete Virtual View*,
      iv. The *Complete Physical (Linear) view*,
      v. The lines connecting these actors together.
      vi. The *Directory Entry* table?
      vii. The *Page Entry* table?
   (c) What is the table labelled *Virtual Addr* that is found at the bottom of the page. What kinds of information is stored there?

2. The following questions concern the `gcc` code from the blog titled, *Scanning Pages From Inside A Process*, first sample of `gcc` code and shown in Figure 2.



```
#include <stdio.h>

const char a            = 'a';
      char b            = 'b';
      char c(void){return 0;};
const char d            = 'd';
      char e            = 'e';
      char f(void){return 0;};

int main(){
        printf("a: %p, b: %p, c: %p, d: %p, e: %p, f: %p\n", (void *)&a,
(void *)&b, (void *)&c, (void *)&d, (void *)&e, (void *)&f);
        return 0;
}
```

Figure 2: Source code for Part 1. This code is also available on Robert Elder's blog page.

(a) How is this code related to virtual memory (or any type of memory, for that matter)?

(b) In a few sentences, explain how the code works. Can you make the `printf()` statements more meaningful for a new user?

(c) Give a copied-pasted view of your output. Please explain what the output signifies.

3. The following questions concern the `gcc` code from the blog shown in Figure 3.

   (a) How is this code related to virtual memory (or any type of memory, for that matter)?

   (b) In a few sentences, explain how the code works. Can you make the `printf()` statements more meaningful for a new user?

   (c) Give a copied-pasted view of your output. Please explain what the output signifies.

4. The following questions concern the `gcc` code from the blog shown in Figure 4.

   (a) How is this code related to virtual memory (or any type of memory, for that matter)?

   (b) In a few sentences, explain how the code works. Can you make the `printf()` statements more meaningful for a new user?

   (c) Give a copied-pasted view of your output. Please explain what the output signifies.

5. The following questions concern the `gcc` code from the blog shown in Figure 5.

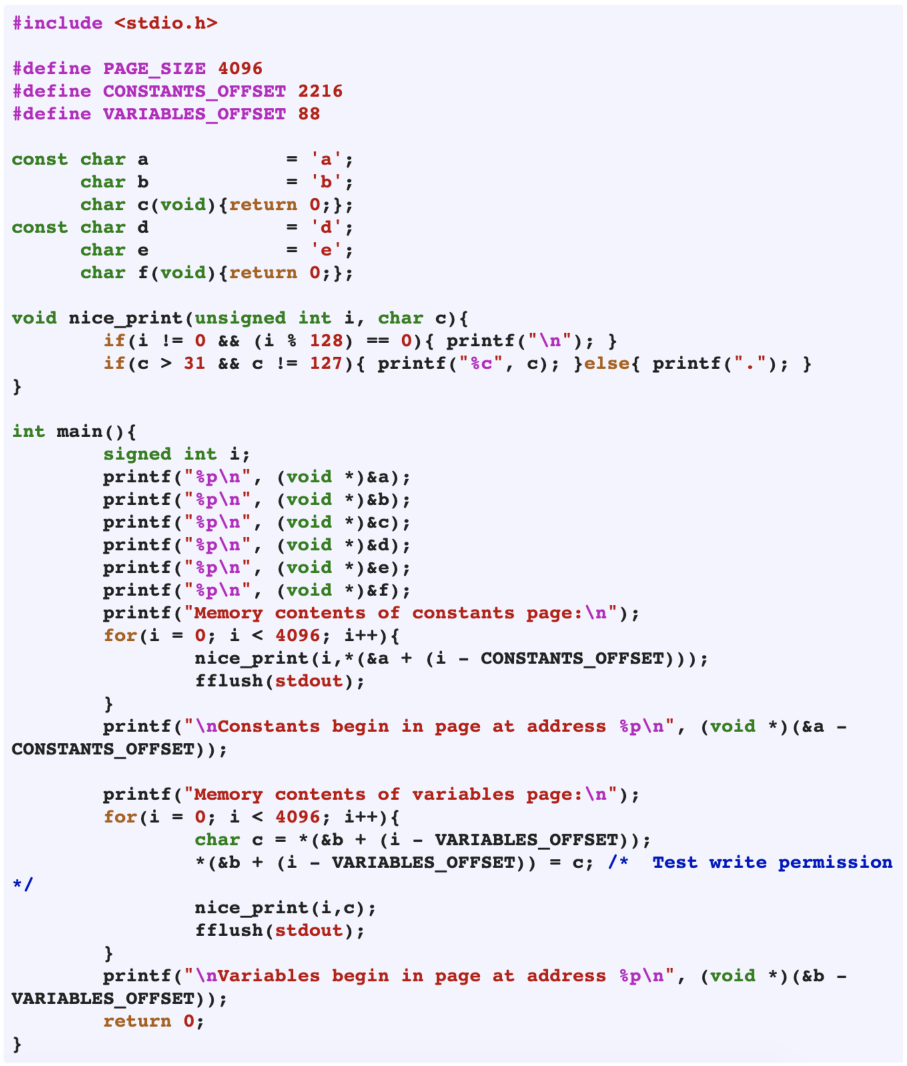
   (a) How is this code related to virtual memory (or any type of memory, for that matter)?

   (b) In a few sentences, explain how the code works. Can you make the `printf()` statements more meaningful for a new user?

   (c) Give a copied-pasted view of your output. Please explain what the output signifies.

6. The following questions concern the `gcc` code from the blog shown in Figure 6.

   (a) How is this code related to virtual memory (or any type of memory, for that matter)?



```
#include <stdio.h>

#define PAGE_SIZE 4096
#define CONSTANTS_OFFSET 2216
#define VARIABLES_OFFSET 88

const char a            = 'a';
      char b            = 'b';
      char c(void){return 0;};
const char d            = 'd';
      char e            = 'e';
      char f(void){return 0;};

void nice_print(unsigned int i, char c){
        if(i != 0 && (i % 128) == 0){ printf("\n"); }
        if(c > 31 && c != 127){ printf("%c", c); }else{ printf("."); }
}

int main(){
        signed int i;
        printf("%p\n", (void *)&a);
        printf("%p\n", (void *)&b);
        printf("%p\n", (void *)&c);
        printf("%p\n", (void *)&d);
        printf("%p\n", (void *)&e);
        printf("%p\n", (void *)&f);
        printf("Memory contents of constants page:\n");
        for(i = 0; i < 4096; i++){
                nice_print(i,*(&a + (i - CONSTANTS_OFFSET)));
                fflush(stdout);
        }
        printf("\nConstants begin in page at address %p\n", (void *)(&a -
CONSTANTS_OFFSET));

        printf("Memory contents of variables page:\n");
        for(i = 0; i < 4096; i++){
                char c = *(&b + (i - VARIABLES_OFFSET));
                *(&b + (i - VARIABLES_OFFSET)) = c; /*  Test write permission
*/
                nice_print(i,c);
                fflush(stdout);
        }
        printf("\nVariables begin in page at address %p\n", (void *)(&b -
VARIABLES_OFFSET));
        return 0;
}
```

Figure 3: Source code for Part 2. This code is also available on Robert Elder's blog page.

(b) In a few sentences, explain how the code works. Can you make the `printf()` statements more meaningful for a new user?

(c) Give a copied-pasted view of your output. Please explain what the output signifies.

Summery of Deliverables

1. File `writing/report.md`: In this file, all your questions-in-blue from above would have been answered.

2. Files `src/part1.c`, `src/part2.c`, `src/part3.c`, `src/part4.c` and `src/part5.c`: In which you have added some more meaningful language to the print statements.

```c
#include <stdio.h>

const unsigned int a = 0x12345678; /*  Just placeholders for now */
const unsigned int b = 0x90123456;
const unsigned int c = 0x78901234;
const unsigned int d = 0x56789012;

unsigned int func1(unsigned int i){
        return i + 8;
}

int main(void){
        unsigned int * i;
        unsigned int num;
        /*  Print out the bytecode for 'func1' */
        for(i = (unsigned int*)func1; i < (unsigned int*)main; i++){
                printf("%p: 0x%08X\n", (void *)i, *i);
        }
        num = func1(29);
        printf("%u\n", num); /*  Prints 37*/
}
```

Figure 4: Source code for Part 3. This code is also available on Robert Elder's blog page.

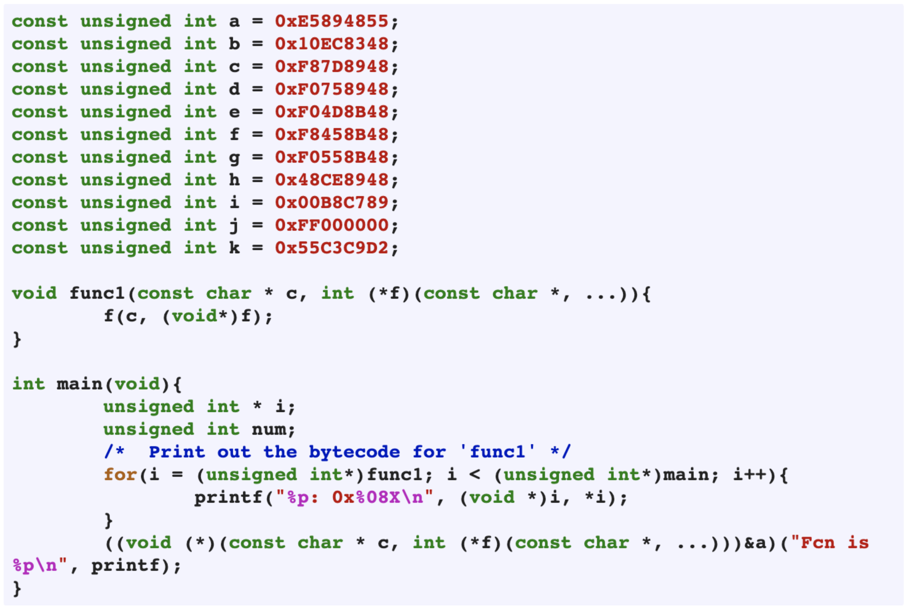
```c
#include <stdio.h>

const unsigned int a = 0xE5894855;  /*  Update these based on */
const unsigned int b = 0x8BFC7D89;  /*  what was output in the */
const unsigned int c = 0xC083FC45;  /*  last run of this program. */
const unsigned int d = 0x55C35D08;

unsigned int func1(unsigned int i){
        return i + 8;
}

int main(void){
        unsigned int * i;
        unsigned int num;
        /*  Print out the bytecode for 'func1' */
        for(i = (unsigned int*)func1; i < (unsigned int*)main; i++){
                printf("%p: 0x%08X\n", (void *)i, *i);
        }
        /*  Cast the address of 'a' to a function pointer and call it */
        num = ((unsigned int (*)(unsigned int))&a)(29);
        printf("%u\n", num); /*  Prints 37*/
}
```

Figure 5: Source code for Part 4. This code is also available on Robert Elder's blog page.



```c
const unsigned int a = 0xE5894855;
const unsigned int b = 0x10EC8348;
const unsigned int c = 0xF87D8948;
const unsigned int d = 0xF0758948;
const unsigned int e = 0xF04D8B48;
const unsigned int f = 0xF8458B48;
const unsigned int g = 0xF0558B48;
const unsigned int h = 0x48CE8948;
const unsigned int i = 0x00B8C789;
const unsigned int j = 0xFF000000;
const unsigned int k = 0x55C3C9D2;

void func1(const char * c, int (*f)(const char *, ...)){
        f(c, (void*)f);
}

int main(void){
        unsigned int * i;
        unsigned int num;
        /*  Print out the bytecode for 'func1' */
        for(i = (unsigned int*)func1; i < (unsigned int*)main; i++){
                printf("%p: 0x%08X\n", (void *)i, *i);
        }
        ((void (*)(const char * c, int (*f)(const char *, ...)))&a)("Fcn is
%p\n", printf);
}
```

Figure 6: Source code for Part 5. This code is also available on Robert Elder's blog page.