Collin Gros
08-30-2020
CS-471
Program #1

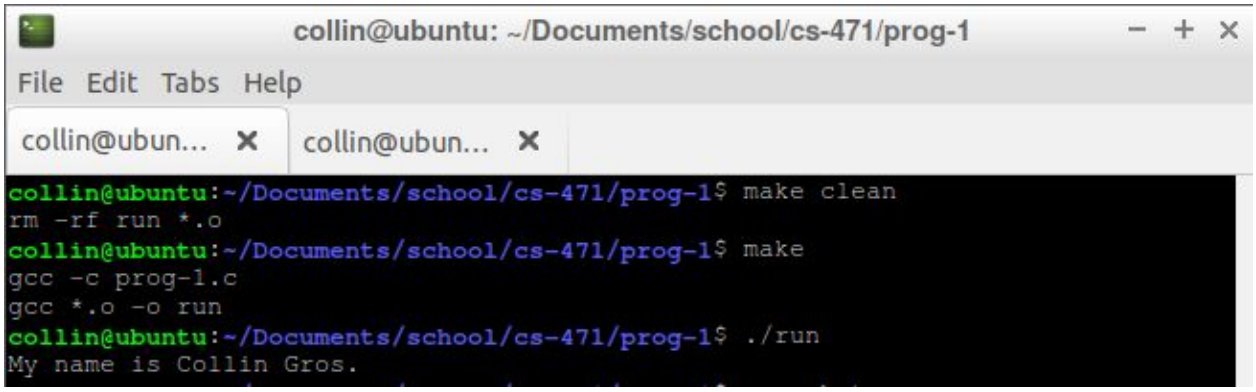
PROGRAM 1

1) Your code shall be properly commented include name , date, input, output, preconditions and postconditions

Code (prog-1.c):

```c
/*
    collin gros
    08-29-2020
    cs-471
    prog-1
*/


#include <stdio.h>


/*  INPUT: void
    OUTUT: return status of main
    PRECONDITION: program is running
    POSTCONDITION: prints out my full name  */
int main()
{
    /*  3 bytes are needed for my name, plus the one
        for a null */
    int nameArr[4];
    /*  for printing the array of ascii-coded bytes with printf */
    char *S = (char *) nameArr;

    /*  must multiply by powers of 256 (i think this is the same
        thing as bitshifting by 8 because 2^8 = 256?)   */
    nameArr[0] = 'C' +
                 'o' * (256) +
                 'l' * (256 * 256) +
                 'i' * (256 * 256 * 256);
    nameArr[1] = 'i' +
                 'n' * (256) +
                 ' ' * (256 * 256) +
                 'G' * (256 * 256 * 256);
    nameArr[2] = 'r' +
                 'o' * (256) +
                 's' * (256 * 256) +
                 '.' * (256 * 256 * 256);
    /*  null character to stop printf   */
    nameArr[3] = 0;

    /*  print my name!  */
    printf("My name is %s\n", S);
}

```

2) A screen shot of you program running



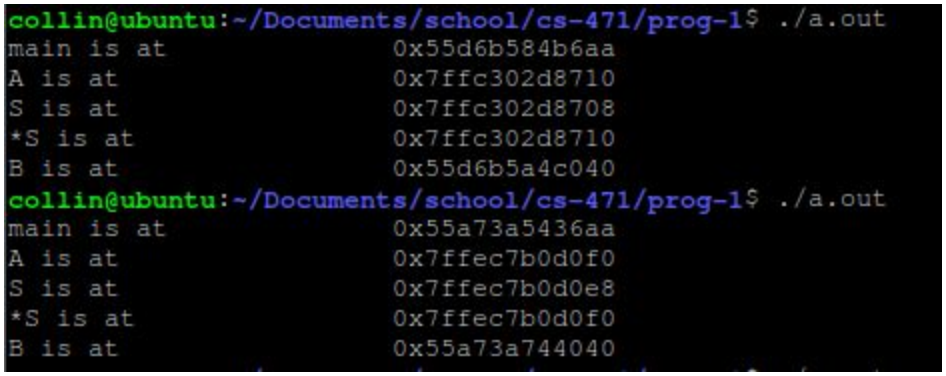
3) Answers to the following questions

a) in what memory segment is the array allocated? Give proof that your answer is correct

The array is allocated in the stack. I experimented with a little program using the same method as shown during lecture (program is below).

```
18 int main()
19 {
20     /*  in main's activation record (stack) */
21     int A[100];
22     /*  same as A   */
23     char *S = (char *)A;
24
25     /*  in the data segment */
26     static int B[100];
27
28     printf("main is at \t\t%p\n"
29            "A is at \t\t%p\n"
30            "S is at \t\t%p\n"
31            "*S is at \t\t%p\n"
32            "B is at \t\t%p\n", main, A, &S, S, B);
33
34     return 1;
35 }
36
```

output from running twice:



From the output, main and B are really close to each other, while A and S are also close to each other (located in the stack) and change every time the program is re-run.

b) in what memory segment is the pointer to the array allocated? Give proof that your answer is correct

The pointer to the array (in my example, S) is located near A, in the stack. From my output, you can see that A and S are very close to each other, meaning they are likely in the same part of memory. I know they must be in the stack because they change every time the program is ran, and are very far away from main and B.

c) how can you make your array be in another segment? Show how you did this and show proof

I made B appear in the data segment instead of the stack segment by applying the keyword *static*. From my output, you can see that B and main are close to each other, meaning they are both located in the data segment.

d) What endianness was the computer you ran your problem on?

Little-endian (intel processor).

e) Why is there a difference between little and big endian? Which one is better? Provide a source

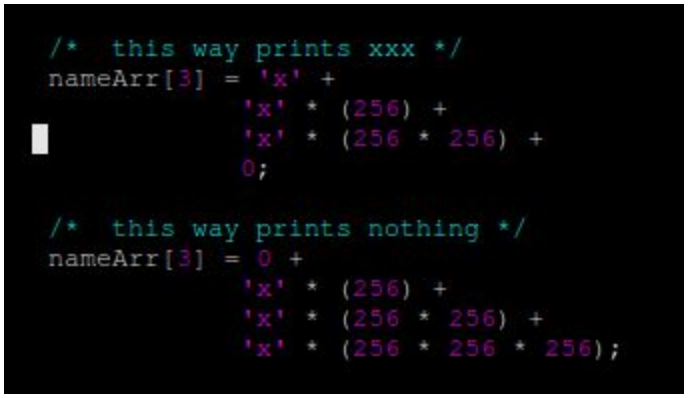
Big-endian and little-endian determine from which direction to read bytes. In order to begin processing data, the data must be read from one direction to the other. According to Wikipedia [1], network protocols still use big-endian, while processor types use little-endian or middle-endian. In a stack overflow response by I. J. Kennedy [2], addition makes it important for processors to use little-endian instead of big-endian. Since carries move towards significant digits, it's faster to have a little-endian read the first byte and begin addition than it is to wait for the big-endian processor to read the entire string of bytes.

Sources

1. https://en.wikipedia.org/wiki/Endianness#History
2. https://stackoverflow.com/questions/5185551/why-is-x86-little-endian

4) Do you we need to fill the entire last interger with '0', or can we just fill in the last byte with '0'. Show an experiment that shows this (make sure you pay attention to endianness and ensure that your other bytes are NOT 0 when doing the experiment.

From an added experiment in my code:



The first method, placing the 0 byte at the end, prints all the other bytes before it. This means that the entire integer does NOT need to be filled with 0, but only a single byte does.
Method 1:

```c
int main()
{
    /*  3 bytes are needed for my name, plus the one
        for a null */
    int nameArr[4];
    /*  for printing the array of ascii-coded bytes with printf */
    char *S = (char *) nameArr;

    /*  must multiply by powers of 256 (i think this is the same
        thing as bitshifting by 8 because 2^8 = 256?)    */
    nameArr[0] = 'C' +
                 'o' * (256) +
                 'l' * (256 * 256) +
                 'l' * (256 * 256 * 256);
    nameArr[1] = 'i' +
                 'n' * (256) +
                 ' ' * (256 * 256) +
                 'G' * (256 * 256 * 256);
    nameArr[2] = 'r' +
                 'o' * (256) +
                 's' * (256 * 256) +
                 '.' * (256 * 256 * 256);

    /*  this way prints xxx */
    nameArr[3] = 'x' +
                 'x' * (256) +
                 'x' * (256 * 256) +
                 0;

    /*  this way prints nothing */
/*
    nameArr[3] = 0 +
                 'x' * (256) +
                 'x' * (256 * 256) +
                 'x' * (256 * 256 * 256);
*/

    /*  print my name!  */
    printf("My name is %s\n", S);
}
```

Output:

```
collin@ubuntu:~/Documents/school/cs-471/prog-1$ !gcc
gcc prog-1-alt.c
collin@ubuntu:~/Documents/school/cs-471/prog-1$ ./a.out
My name is Collin Gros.xxx
```

Method 2:

```c
int main()
{
    /*  3 bytes are needed for my name, plus the one
        for a null */
    int nameArr[4];
    /*  for printing the array of ascii-coded bytes with printf */
    char *S = (char *) nameArr;

    /*  must multiply by powers of 256 (i think this is the same
        thing as bitshifting by 8 because 2^8 = 256?)   */
    nameArr[0] = 'C' +
                 'o' * (256) +
                 'l' * (256 * 256) +
                 'l' * (256 * 256 * 256);
    nameArr[1] = 'i' +
                 'n' * (256) +
                 ' ' * (256 * 256) +
                 'G' * (256 * 256 * 256);
    nameArr[2] = 'r' +
                 'o' * (256) +
                 's' * (256 * 256) +
                 '.' * (256 * 256 * 256);

    /*  this way prints xxx */
/*
    nameArr[3] = 'x' +
                 'x' * (256) +
                 'x' * (256 * 256) +
                 0;
*/

    /*  this way prints nothing */
    nameArr[3] = 0 +
                 'x' * (256) +
                 'x' * (256 * 256) +
                 'x' * (256 * 256 * 256);

    /*  print my name!  */
    printf("My name is %s\n", S);
}
```

Output:

```
collin@ubuntu:~/Documents/school/cs-471/prog-1$ !gcc
gcc prog-1-alt.c
collin@ubuntu:~/Documents/school/cs-471/prog-1$ ./a.out
My name is Collin Gros.
```