CS 214 Recitation(Sec. 6)

Zuohui Fu

Ph.D. Department of Computer Science

Office hour: Mon 2pm-3pm

Email: zf87 AT cs dot rutgers dot edu

10/05/2017

Topics

- Malloc
- Valgrind

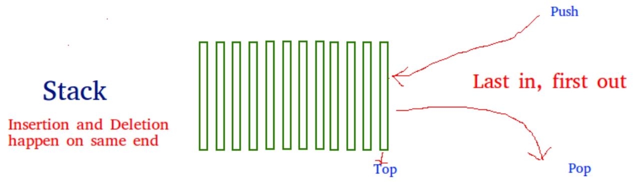
Memory allocation

- malloc()

  Allocates requested size of bytes and returns a pointer first byte of allocated space

- Stack

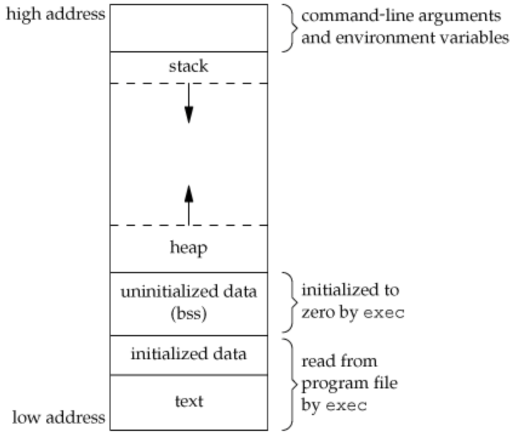
  Stack is a linear data structure which follows a particular order in which the operations are performed. The order may be LIFO(Last In First Out) or FILO(First In Last Out).



Memory allocation

- Heap

  Heap is the segment where dynamic memory allocation usually takes place. Heap area is managed by malloc, realloc, and free. The Heap area is shared by all shared libraries and dynamically loaded modules in a process.


high address
command-line arguments and environment variables
stack
heap
uninitialized data (bss)
initialized to zero by exec
initialized data
text
read from program file by exec
low address

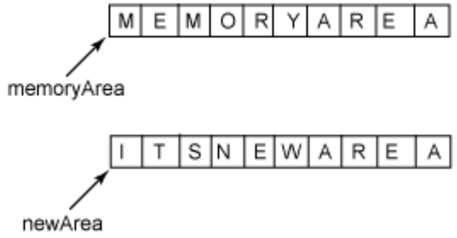
Memory Leek

- After you use malloc to assigh a dynamic variable, be sure to free it.

```
char *name = (char *) malloc(11);
// Assign some value to name
memcpy ( p,name,11); // Problem begins here
```

- Another example

```
char *memoryArea = malloc(10);
char *newArea = malloc(10);
```

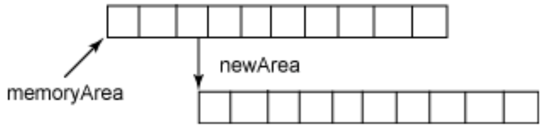


```
memoryArea = newArea;
```

Memory Leek

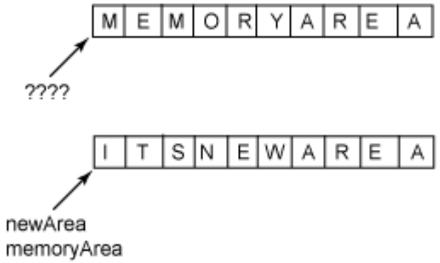
- memoryArea has not been FREE

- Example again:



```
free(memoryArea)
```

```
free( memoryArea->newArea);
free(memoryArea);
```



Memory Leek

- For more:
- https://wr.informatik.uni-hamburg.de/_media/teaching/sommersemester_2014/cgk-14-menck-memory-leaks-report.pdf
- https://cardinalpeak.com/blog/bens-golden-rule-for-preventing-memory-leaks/
- https://www.cprogramming.com/tutorial/memory_debugging_parallel_inspector.html
- http://www.yolinux.com/TUTORIALS/C++MemoryCorruptionAndMemoryLeaks.html

Valgrind

- Valgrind is a programming tool for memory debugging, memory leak detection.
- Official webpage:
  http://valgrind.org/
- Add -g option of gcc to sets up debugging information:
  - *gcc -g xxx.c -o xxx*
- Launch Valgrind by valgrind ./xxx
- How to use it:
  https://www.youtube.com/watch?v=bb1bTJtgXrI&feature=youtu.be

Valgrind-Example

```c
 1 int main(void)
 2 {
 3          char *p1;
 4          char *p2;
 5
 6          p1 = (char *) malloc(512);
 7          p2 = (char *) malloc(512);
 8
 9          p1=p2;
10
11          free(p1);
12          free(p2);
13 }
```

We lose p2, memory leak happens

```
# gcc –g –o test2 test2.c
# valgrind ./test2
.
.
==31468== Invalid free() / delete / delete[]
==31468==    at 0xFFB9FF0: free (vg_replace_malloc.c:152)
==31468==    by 0x100004B0: main (test2.c:12)
==31468== Address 0x11899258 is 0 bytes inside a block of size 512
free'd
==31468==    at 0xFFB9FF0: free (vg_replace_malloc.c:152)
==31468==    by 0x100004A4: main (test2.c:11)
==31468==
==31468== ERROR SUMMARY: 1 errors from 1 contexts (suppressed: 7 from
1)
==31468== malloc/free: in use at exit: 512 bytes in 1 blocks.
==31468== malloc/free: 2 allocs, 2 frees, 1024 bytes allocated.
==31468== For counts of detected errors, rerun with: -v
==31468== searching for pointers to 1 not-freed blocks.
==31468== checked 167936 bytes.
==31468==
==31468== LEAK SUMMARY:
==31468==    definitely lost: 512 bytes in 1 blocks.
==31468==      possibly lost: 0 bytes in 0 blocks.
==31468==    still reachable: 0 bytes in 0 blocks.
==31468==         suppressed: 0 bytes in 0 blocks.
==31468== Use --leak-check=full to see details of leaked memory.
```

Solution to HW1

- What's wrong with this #define line?

#define N 10;

The semicolon at the end of the line will become part of N's definition, which is hardly ever what you want.

- Suppose you had the definition #define SIX 2*3 . What value would the declaration int x = 12 / SIX; initialize x to?

18

Solution to HW1

- Write your own version of atoi

Please refer to Leetcode 8, just pass the character like ‘a,b,c….’

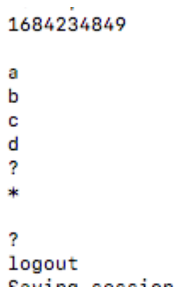
Solution to HW2

- ip is a variable which can point to an int (that is, its value will be a pointer to an int; or informally, we say that ip *is* "a pointer to an int”). Its value is a pointer which points to the variable i.

- ip is a pointer that stores the address of i, not what the i points to.

Solution to HW2

- Write some code that declares two arrays of size 10 that are string literals. Make a pointer to one of the arrays, cast it to be an int pointer, and print out its value. Make a new integer, set it equal to the value of your int pointer, then make a pointer to that integer, cast it to be a char pointer, and print out 8 chars. What happened? Why?



```
1684234849

a
b
c
d
?
*

?
logout
Saving session
```

Solution to HW2

p* = char $\boxed{\text{'a'}}\boxed{\text{'b'}}\boxed{\text{'c'}}\boxed{\text{'d'}}\boxed{\text{'e'}}\boxed{\text{'f'}}\boxed{\dots}$

By ASCII: Decimal: 97 98 99 100 101 102 ....

Hexadecimal: 61 62 63 64 65 ----

Binary: 11000001, 11000010, 11000011, 11000100

Int C only takes 4 bits $\Rightarrow$ $\boxed{\text{'d'}}$ $\boxed{\text{'c'}}$ $\boxed{\text{'b'}}$ $\boxed{\text{'a'}}$

11000100 11000011 11000010 11000001

Higher $\longleftarrow$ Lower

1684234849

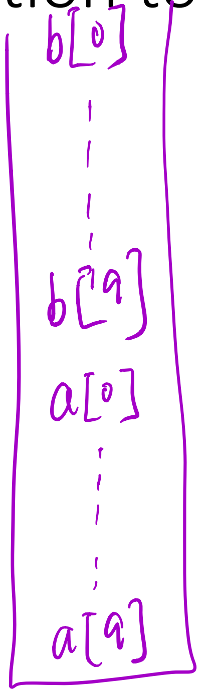
only print out a, b, c, d.

Solution to HW2

- Write some code that declares two arrays of size 10 that are string literals. Create a pointer that points to the beginning of the first array, then in a loop, increment the pointer and print out the char it points to, out to index 20. What happened? Why?

Solution to HW2

char a[10]
char b[10]



stack.

First pushed, to the bottom

HW3

- 0. What are the differences between strlen and sizeof a string in C? Why?
- 1. Write the function: replace(char string[], char from[], char to[])

  which finds the string from in the string string and replaces it with the string to. You may assume that from and to are the same length. For example, the code:

  ```
  char string[] = "recieve";
  replace(string, "ie", "ei");
  ```

should change string to "receive".

HW3-Cont.

- 2. Write a short program to read two lines of text, and concatenate them using strcat. Since strcat concatenates in-place, you'll have to make sure you have enough memory to hold the concatenated copy. For now, use a char array which is twice as big as either of the arrays you use for reading the two lines. Use strcpy to copy the first string to the destination array, and strcat to append the second one.