CSO-Recitation 11

CSCI-UA 0201-007

R11: Assessment 09 & Dynamic memory allocation

Today’s Topics

- Assessment 09
- Dynamic memory allocation
  - implement your malloc & free

Assessment 09

Q1 seg fault

Which of the following statements on segmentation faults are true?

A. Performing any out-of-bounds array access will result in an immediate segmentation fault.

B. Segmentation faults only occur when an instruction tries to write to memory.

C. Dereferencing a null pointer will always result in a segmentation fault.

D. Performing pointer arithmetic will always result in a segmentation fault.

E. none of the above

Q2 local variable

Suppose *local variable* a is defined as int a[16]; Which of the following statements are true?

A. a takes up space on the stack.

B. a takes up space on the heap.

C. subq $64, %rsp allocates space for a and addq $64, %rsp de-allocates space for a.

stack goes downwards

D. addq $64, %rsp allocates space for a and subq $64, %rsp de-allocates space for a.

E. none of the above

Q3 str_concat

The following C function str_concat appends the src string to the dst string, overwriting the terminating null byte at the end of dst, and then adds a terminating null byte.

- **Q3.1** line 5
- Please fill in the code at line 6 (must be a one liner). *To facilitate automatic grading, please do not have any spaces in your C statement, and make sure to include the end of the statement semicolon*.
- dst[len+i]=src[i];

```
1: void str_concat(char *dst, char *src)
2: {
3:   int len = strlen(dst);
4:   int i;
5:   for (i = 0; src[i]!='\0'; i++) {
6:      ???
7:   }
8:   ???
9: }
```

Q3.2 line 7

Which of the following C statement can be used at line 8 correctly, without compilation nor runtime error?

A. `dst[len] = '\0';`
B. `dst[len] = NULL;`
C. `dst[len] = 0;`
D. `dst[len+i] = '\0';`
E. `dst[len+i] = NULL;`
F. `dst[len+i] = 0;`

NULL: means a null pointer

Q4. Given the following C program (which invokes str_concat defined in Q3)

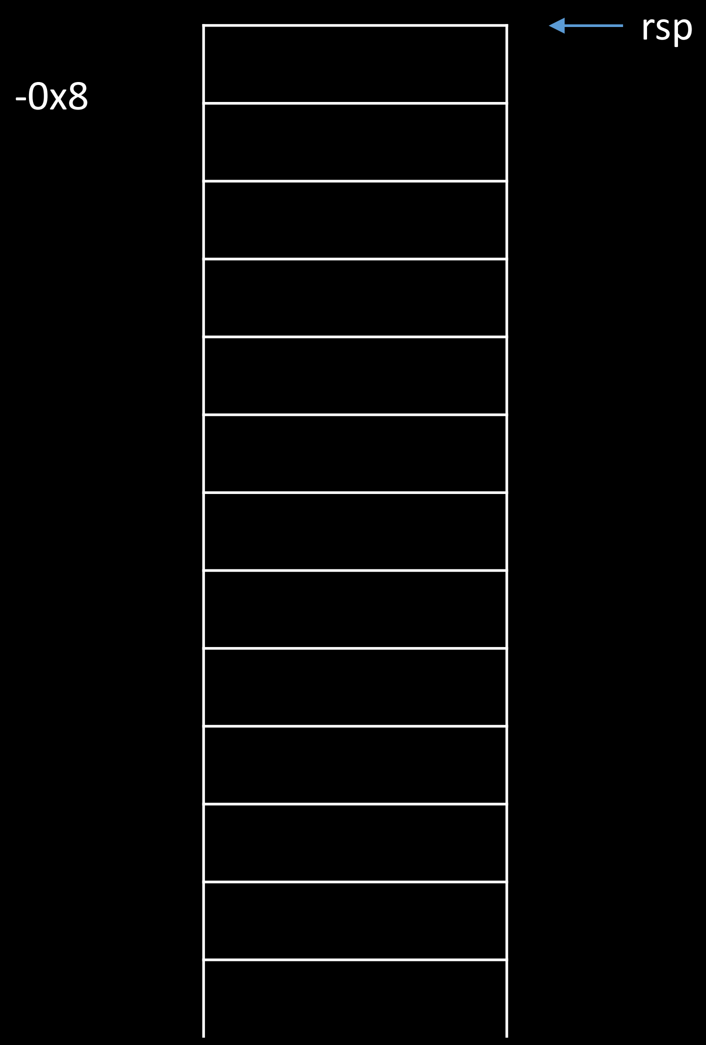
```c
void dangerous()
{
  char buf1[8] = "hello";
  char buf2[8] = "world";
  str_concat(buf1, buf2);
}

int main()
{
  dangerous();
  printf("i wonder if this program is correct\n");
}
```

Suppose the following assembly is generated for the above C program (including str_concat). Please assume that the addresses to the left of each instruction shown below are the actual addresses where the instructions are stored at during runtime.

```
000000000550068a <str_concat>:
            68a:        48 83 ec 28             subq    $0x18,%rsp
            ...omitted....
            711:        c3                      ret

00000000055006ba <dangerous>:
            6ba:        48 83 ec 10             subq    $0x10,%rsp
            6be:        48 b8 68 65 6c 6c 6f    movq    $0x6f6c6c6568,%rax
            6c8:        48 89 44 24 08          movq    %rax,0x8(%rsp)
            6cd:        48 b8 77 6f 72 6c 64    movq    $0x646c726f77,%rax
            6d7:        48 89 04 24             movq    %rax,(%rsp)
            6db:        48 89 e6                movq    %rsp,%rsi
            6de:        48 8d 7c 24 08          leaq    0x8(%rsp),%rdi
            6e3:        e8 82 ff ff ff          callq  66a <str_concat>
            6e8:        48 83 c4 10             addq    $0x10,%rsp
            6ec:        c3                      ret

00000000055006ed <main>:
            6ed:        48 83 ec 08             subq    $0x8,%rsp
            6f1:        b8 00 00 00 00          movq    $0x0,%eax
            6f6:        e8 bf ff ff ff          callq  6ba <dangerous>
            6fb:        48 8d 35 a2 00 00 00    leaq    0xa2(%rip),%rsi
            702:        bf 01 00 00 00          movq    $0x1,%edi
            707:        b8 00 00 00 00          movq    $0x0,%eax
            70c:        e8 2f fe ff ff          callq  540 <__printf_chk@plt>
            711:        b8 00 00 00 00          movq    $0x0,%eax
            716:        48 83 c4 08             addq    $0x8,%rsp
            71a:        c3                      ret
```

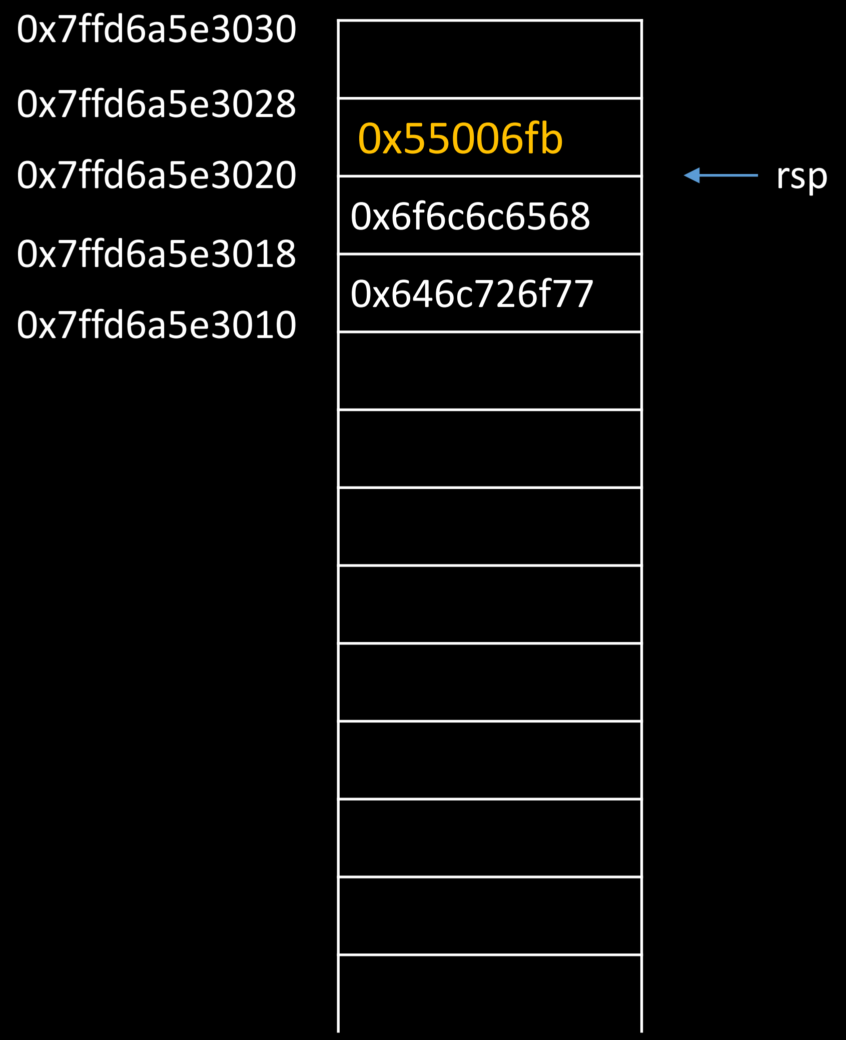
Q4

```
000000000550068a <str_concat>:
            68a:       48 83 ec 28                subq    $0x18,%rsp
            ...omitted....
            711:       c3                         ret

00000000055006ba <dangerous>:
            6ba:       48 83 ec 10                subq    $0x10,%rsp
            6be:       48 b8 68 65 6c 6c 6f       movq    $0x6f6c6c6568,%rax
            6c8:       48 89 44 24 08             movq    %rax,0x8(%rsp)
            6cd:       48 b8 77 6f 72 6c 64       movq    $0x646c726f77,%rax
            6d7:       48 89 04 24                movq    %rax,(%rsp)
            6db:       48 89 e6                   movq    %rsp,%rsi
            6de:       48 8d 7c 24 08             leaq    0x8(%rsp),%rdi
            6e3:       e8 82 ff ff ff             callq   66a <str_concat>
            6e8:       48 83 c4 10                addq    $0x10,%rsp
            6ec:       c3                         ret

00000000055006ed <main>:
            6ed:       48 83 ec 08                subq    $0x8,%rsp
            6f1:       b8 00 00 00 00             movq    $0x0,%eax
            6f6:       e8 bf ff ff ff             callq   6ba <dangerous>
            6fb:       48 8d 35 a2 00 00 00       leaq    0xa2(%rip),%rsi
            702:       bf 01 00 00 00             movq    $0x1,%edi
            707:       b8 00 00 00 00             movq    $0x0,%eax
            70c:       e8 2f fe ff ff             callq   540 <__printf_chk@plt>
            711:       b8 00 00 00 00             movq    $0x0,%eax
            716:       48 83 c4 08                addq    $0x8,%rsp
            71a:       c3                         ret
```



Q4

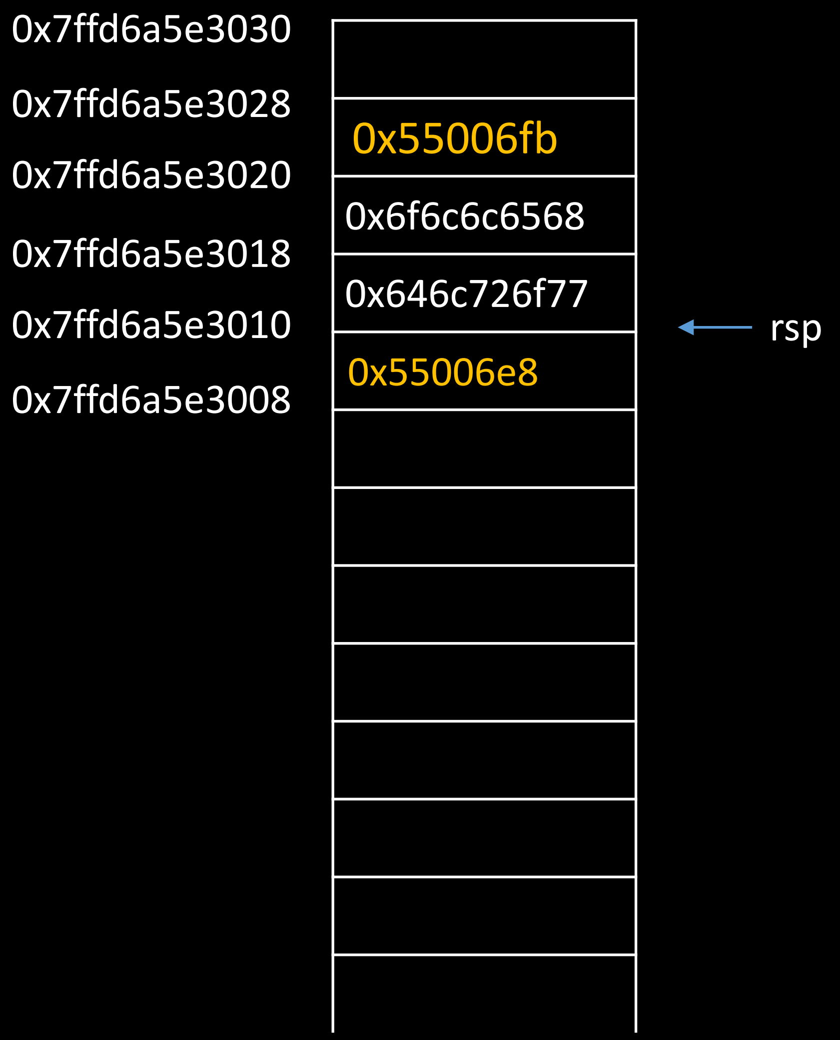
```
000000000550068a <str_concat>:
        68a:    48 83 ec 28             subq   $0x18,%rsp
        ...omitted....
        711:    c3                      ret

00000000055006ba <dangerous>:
        6ba:    48 83 ec 10             subq   $0x10,%rsp
        6be:    48 b8 68 65 6c 6c 6f    movq   $0x6f6c6c6568,%rax
        6c8:    48 89 44 24 08          movq   %rax,0x8(%rsp)
        6cd:    48 b8 77 6f 72 6c 64    movq   $0x646c726f77,%rax
        6d7:    48 89 04 24             movq   %rax,(%rsp)
        6db:    48 89 e6                movq   %rsp,%rsi
        6de:    48 8d 7c 24 08          leaq   0x8(%rsp),%rdi
        6e3:    e8 82 ff ff ff          callq  66a <str_concat>
        6e8:    48 83 c4 10             addq   $0x10,%rsp
        6ec:    c3                      ret

00000000055006ed <main>:
        6ed:    48 83 ec 08             subq   $0x8,%rsp
        6f1:    b8 00 00 00 00          movq   $0x0,%eax
        6f6:    e8 bf ff ff ff          callq  6ba <dangerous>
        6fb:    48 8d 35 a2 00 00 00    leaq   0xa2(%rip),%rsi
        702:    bf 01 00 00 00          movq   $0x1,%edi
        707:    b8 00 00 00 00          movq   $0x0,%eax
        70c:    e8 2f fe ff ff          callq  540 <__printf_chk@plt>
        711:    b8 00 00 00 00          movq   $0x0,%eax
        716:    48 83 c4 08             addq   $0x8,%rsp
        71a:    c3                      ret
```



Suppose the value of %rsp is 0x7ffd6a5e3020 just prior to executing the first instruction of dangerous

Q4

```
000000000550068a <str_concat>:
            68a:      48 83 ec 28              subq    $0x18,%rsp
            ...omitted....
            711:      c3                       ret

00000000055006ba <dangerous>:
            6ba:      48 83 ec 10              subq    $0x10,%rsp
            6be:      48 b8 68 65 6c 6c 6f     movq    $0x6f6c6c6568,%rax
            6c8:      48 89 44 24 08           movq    %rax,0x8(%rsp)
            6cd:      48 b8 77 6f 72 6c 64     movq    $0x646c726f77,%rax
            6d7:      48 89 04 24              movq    %rax,(%rsp)
            6db:      48 89 e6                 movq    %rsp,%rsi
            6de:      48 8d 7c 24 08           leaq    0x8(%rsp),%rdi
            6e3:      e8 82 ff ff ff           callq   66a <str_concat>
            6e8:      48 83 c4 10              addq    $0x10,%rsp
            6ec:      c3                       ret

00000000055006ed <main>:
            6ed:      48 83 ec 08              subq    $0x8,%rsp
            6f1:      b8 00 00 00 00           movq    $0x0,%eax
            6f6:      e8 bf ff ff ff           callq   6ba <dangerous>
            6fb:      48 8d 35 a2 00 00 00     leaq    0xa2(%rip),%rsi
            702:      bf 01 00 00 00           movq    $0x1,%edi
            707:      b8 00 00 00 00           movq    $0x0,%eax
            70c:      e8 2f fe ff ff           callq   540 <__printf_chk@plt>
            711:      b8 00 00 00 00           movq    $0x0,%eax
            716:      48 83 c4 08              addq    $0x8,%rsp
            71a:      c3                       ret
```



Suppose the value of %rsp is 0x7ffd6a5e3020 just prior to executing the first instruction of dangerous

Q4

```
000000000550068a <str_concat>:
         68a:     48 83 ec 28              subq    $0x18,%rsp
         ...omitted....
         711:     c3                       ret

00000000055006ba <dangerous>:
         6ba:     48 83 ec 10              subq    $0x10,%rsp
         6be:     48 b8 68 65 6c 6c 6f     movq    $0x6f6c6c6568,%rax
         6c8:     48 89 44 24 08           movq    %rax,0x8(%rsp)
         6cd:     48 b8 77 6f 72 6c 64     movq    $0x646c726f77,%rax
         6d7:     48 89 04 24              movq    %rax,(%rsp)
         6db:     48 89 e6                 movq    %rsp,%rsi
         6de:     48 8d 7c 24 08           leaq    0x8(%rsp),%rdi
         6e3:     e8 82 ff ff ff           callq  66a <str_concat>
         6e8:     48 83 c4 10              addq    $0x10,%rsp
         6ec:     c3                       ret

00000000055006ed <main>:
         6ed:     48 83 ec 08              subq    $0x8,%rsp
         6f1:     b8 00 00 00 00           movq    $0x0,%eax
         6f6:     e8 bf ff ff ff           callq  6ba <dangerous>
         6fb:     48 8d 35 a2 00 00 00     leaq    0xa2(%rip),%rsi
         702:     bf 01 00 00 00           movq    $0x1,%edi
         707:     b8 00 00 00 00           movq    $0x0,%eax
         70c:     e8 2f fe ff ff           callq  540 <__printf_chk@plt>
         711:     b8 00 00 00 00           movq    $0x0,%eax
         716:     48 83 c4 08              addq    $0x8,%rsp
         71a:     c3                       ret
```

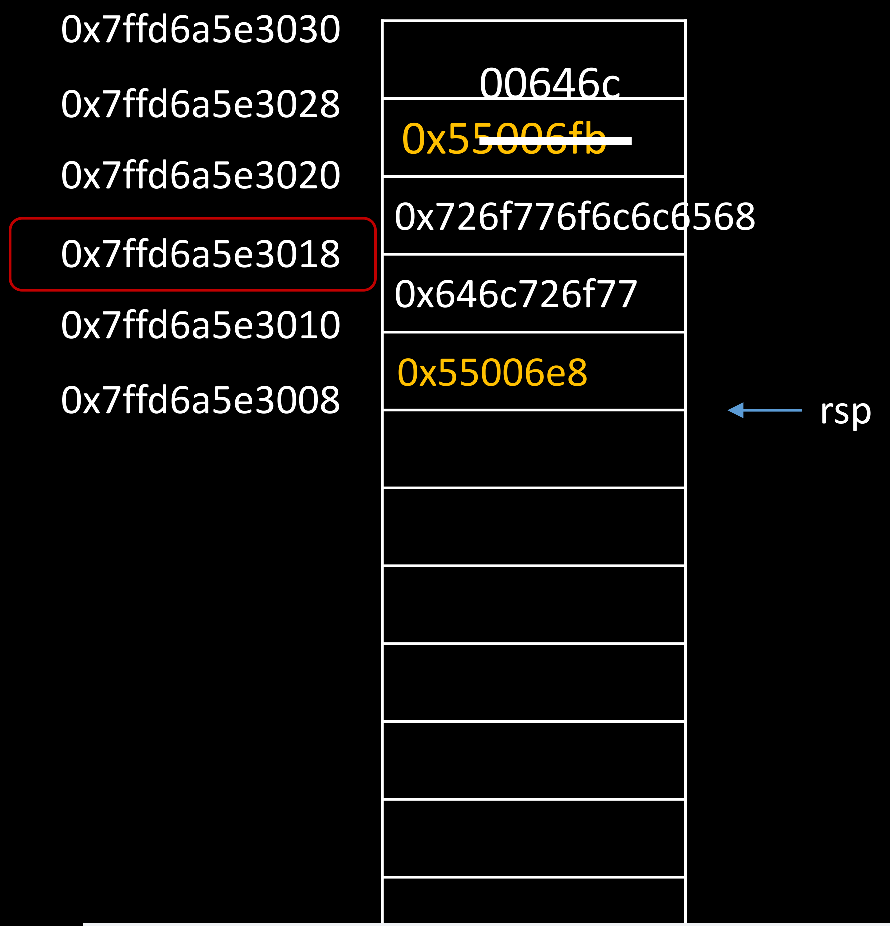

```
void str_concat(char *dst, char *src)
{
   int len = strlen(dst);
   int i;
   for (i = 0; src[i]!='\0'; i++) {
       ???
   }
   ???
}
```

Suppose the value of %rsp is 0x7ffd6a5e3020 just prior to executing the first instruction of dangerous

Q4

```
000000000550068a <str_concat>:
    68a:   48 83 ec 28              subq   $0x18,%rsp
    ...omitted....
    711:   c3                       ret

00000000055006ba <dangerous>:
    6ba:   48 83 ec 10              subq   $0x10,%rsp
    6be:   48 b8 68 65 6c 6c 6f     movq   $0x6f6c6c6568,%rax
    6c8:   48 89 44 24 08           movq   %rax,0x8(%rsp)
    6cd:   48 b8 77 6f 72 6c 64     movq   $0x646c726f77,%rax
    6d7:   48 89 04 24              movq   %rax,(%rsp)
    6db:   48 89 e6                 movq   %rsp,%rsi
    6de:   48 8d 7c 24 08           leaq   0x8(%rsp),%rdi
    6e3:   e8 82 ff ff ff           callq  66a <str_concat>
    6e8:   48 83 c4 10              addq   $0x10,%rsp
    6ec:   c3                       ret

00000000055006ed <main>:
    6ed:   48 83 ec 08              subq   $0x8,%rsp
    6f1:   b8 00 00 00 00           movq   $0x0,%eax
    6f6:   e8 bf ff ff ff           callq  6ba <dangerous>
    6fb:   48 8d 35 a2 00 00 00     leaq   0xa2(%rip),%rsi
    702:   bf 01 00 00 00           movq   $0x1,%edi
    707:   b8 00 00 00 00           movq   $0x0,%eax
    70c:   e8 2f fe ff ff           callq  540 <__printf_chk@plt>
    711:   b8 00 00 00 00           movq   $0x0,%eax
    716:   48 83 c4 08              addq   $0x8,%rsp
    71a:   c3                       ret
```

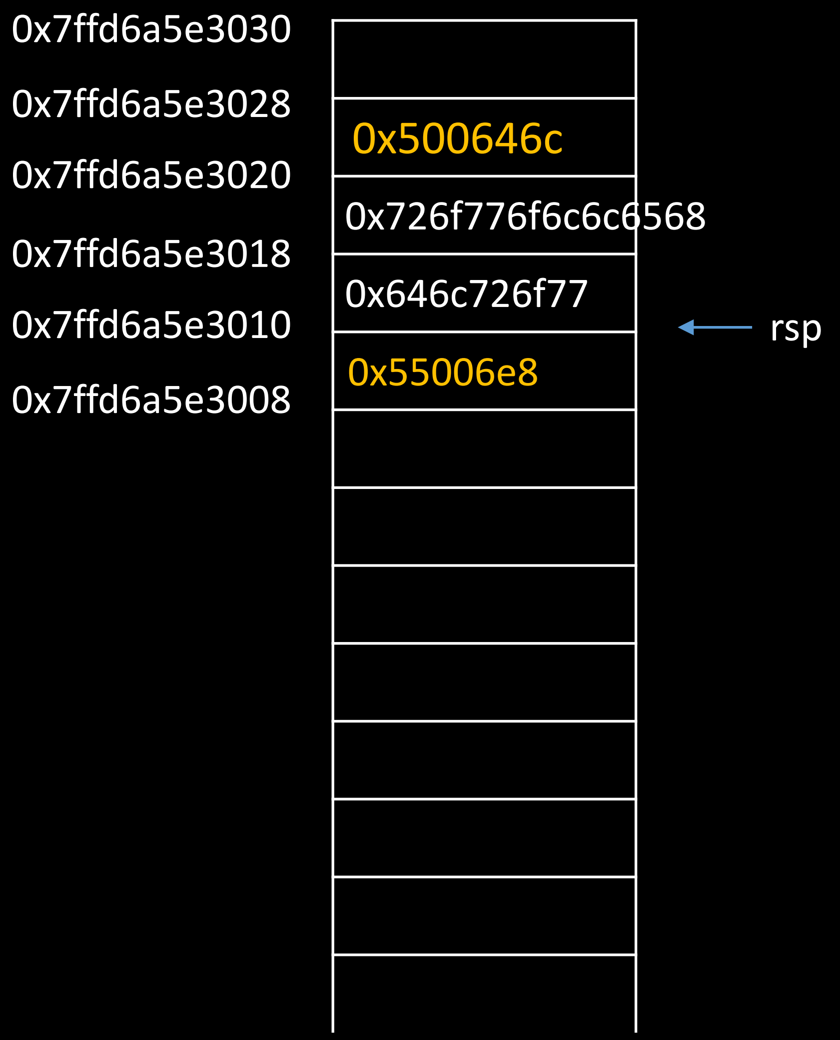
Suppose the value of %rsp is 0x7ffd6a5e3020 just prior to executing the first instruction of dangerous




```
void str_concat(char *dst, char *src)
{
   int len = strlen(dst);
   int i;
   for (i = 0; src[i]!='\0'; i++) {
      ???
   }
   ???
}
```

Q4

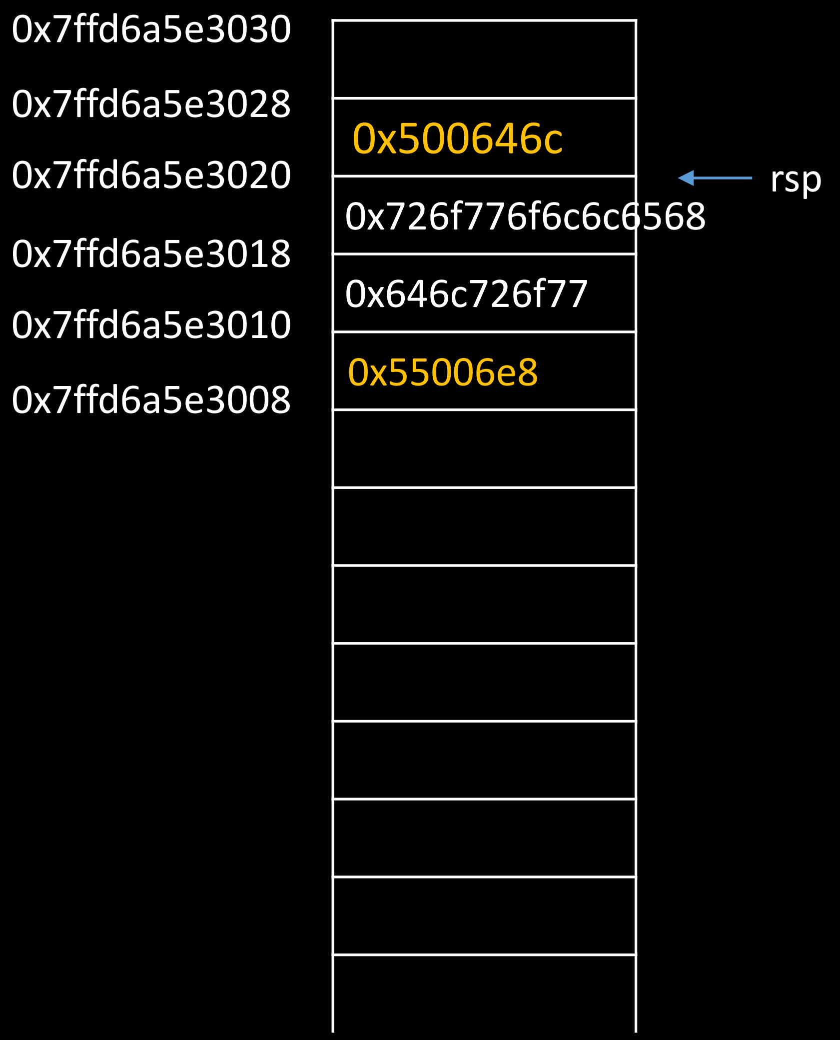
```
000000000550068a <str_concat>:
        68a:    48 83 ec 28             subq   $0x18,%rsp
        ...omitted....
        711:    c3                      ret

00000000055006ba <dangerous>:
        6ba:    48 83 ec 10             subq   $0x10,%rsp
        6be:    48 b8 68 65 6c 6c 6f    movq   $0x6f6c6c6568,%rax
        6c8:    48 89 44 24 08          movq   %rax,0x8(%rsp)
        6cd:    48 b8 77 6f 72 6c 64    movq   $0x646c726f77,%rax
        6d7:    48 89 04 24             movq   %rax,(%rsp)
        6db:    48 89 e6                movq   %rsp,%rsi
        6de:    48 8d 7c 24 08          leaq   0x8(%rsp),%rdi
        6e3:    e8 82 ff ff ff          callq  66a <str_concat>
        6e8:    48 83 c4 10             addq   $0x10,%rsp
        6ec:    c3                      ret

00000000055006ed <main>:
        6ed:    48 83 ec 08             subq   $0x8,%rsp
        6f1:    b8 00 00 00 00          movq   $0x0,%eax
        6f6:    e8 bf ff ff ff          callq  6ba <dangerous>
        6fb:    48 8d 35 a2 00 00 00    leaq   0xa2(%rip),%rsi
        702:    bf 01 00 00 00          movq   $0x1,%edi
        707:    b8 00 00 00 00          movq   $0x0,%eax
        70c:    e8 2f fe ff ff          callq  540 <__printf_chk@plt>
        711:    b8 00 00 00 00          movq   $0x0,%eax
        716:    48 83 c4 08             addq   $0x8,%rsp
        71a:    c3                      ret
```



Suppose the value of %rsp is 0x7ffd6a5e3020 just prior to executing the first instruction of dangerous

Q4

```
000000000550068a <str_concat>:
        68a:    48 83 ec 28             subq   $0x18,%rsp
        ...omitted....
        711:    c3                      ret

00000000055006ba <dangerous>:
        6ba:    48 83 ec 10             subq   $0x10,%rsp
        6be:    48 b8 68 65 6c 6c 6f    movq   $0x6f6c6c6568,%rax
        6c8:    48 89 44 24 08          movq   %rax,0x8(%rsp)
        6cd:    48 b8 77 6f 72 6c 64    movq   $0x646c726f77,%rax
        6d7:    48 89 04 24             movq   %rax,(%rsp)
        6db:    48 89 e6                movq   %rsp,%rsi
        6de:    48 8d 7c 24 08          leaq   0x8(%rsp),%rdi
        6e3:    e8 82 ff ff ff          callq  66a <str_concat>
        6e8:    48 83 c4 10             addq   $0x10,%rsp
        6ec:    c3                      ret

00000000055006ed <main>:
        6ed:    48 83 ec 08             subq   $0x8,%rsp
        6f1:    b8 00 00 00 00          movq   $0x0,%eax
        6f6:    e8 bf ff ff ff          callq  6ba <dangerous>
        6fb:    48 8d 35 a2 00 00 00    leaq   0xa2(%rip),%rsi
        702:    bf 01 00 00 00          movq   $0x1,%edi
        707:    b8 00 00 00 00          movq   $0x0,%eax
        70c:    e8 2f fe ff ff          callq  540 <__printf_chk@plt>
        711:    b8 00 00 00 00          movq   $0x0,%eax
        716:    48 83 c4 08             addq   $0x8,%rsp
        71a:    c3                      ret
```



Suppose the value of %rsp is 0x7ffd6a5e3020 just prior to executing the first instruction of dangerous

%rip -> 0x500646c

Q4

- **Q4.1** &buf1[0]
- Suppose the value of %rsp is 0x7ffd6a5e3020 just prior to executing the first instruction of dangerous, what is the address of the first element of buf1 (aka &buf1[0])?
- 0x7ffd6a5e3018
- **Q4.2** &buf2[0]
- Using the same premise of Q4.1 earlier, what is the address of the first element of buf2 (aka &buf2[0])?
- 0x7ffd6a5e3010

Q4

- **Q4.3**
- Using the same premise of Q4.1 earlier, what are the 8 bytes stored in the memory address 0x7ffd6a5e3020 (which is the value of %rsp just prior to executing the first instruction of dangerous)?
- 0x55006fb

Q4

- **Q4.4** Which of the following statements are true?

A. This program has no buffer overflow bugs and will execute correctly.

B. This program has a buffer overflow bug, but it will nevertheless execute without a problem because the compiler has protected the stack using a canary.

C. This program has a buffer overflow bug, but it will nevertheless execute without a problem because the compiler has allocated extra space on the stack that cushions the overflow.

D. This program has a buffer overflow bug which is likely to manifest as a segmentation fault.

E. buf1 is overflown during execution.

F. buf2 is overflown during execution.

Q4

- **Q4.5** last instruction

If running this program results in a segmentation fault. What is the last instruction executed before the segmentation fault occurs?

A. The ret instruction in main function

B. The ret instruction in dangerous function

C. The ret instruction in str_concat function

D. The instruction to deallocate stack in dangerous, i.e. addq $0x10,%rsp.

Q4

- **Q4.6** Bonus question
- If running this program results in a segmentation fault, what is the memory address that corresponds to the illegal memory access? You should assume the same premise as Q4.1.
- 0x500646c

Dynamic Memory Allocation

For when static memory isn’t enough

Why Dynamic Memory?

- You don’t always know how much memory you will need for your program
- What if you want to write a program that finds the average value in a column?
- If you did write such a program, how do you handle a user giving you a really big file, bigger than you expected?
- Even if you made sure you specified a really big global variable as a static buffer, people might still give you bigger files
  - And why go through that trouble anyway instead of just having dynamic memory?

Dynamic memory and the stack

- Does the stack give us dynamic memory?
  - In a sense, yes
  - However, it isn’t always suitable, because the memory gets reused after we return from a function call
  - By default the stack is also only a few megabytes in size

Dynamic memory on the heap

- We can use the sbrk syscall to ask the operating system to give us more heap space
- We can also use it to give back to the operating system
- However, in the real world, programmers don’t often do this themselves
  - Why?
- Instead, we usually use a library that handles things for us
  - API: malloc and free
  - Dynamic memory allocator

Malloc and Free

- Malloc allocates us a contiguous section of memory
  - It returns a void*, which is just “pointer to anything”
  - So you cast the result of malloc to what you want, e.g. int *
  - Malloc can return NULL if there was an error
- Free gives the memory back to the allocator
  - DO NOT call free twice on the same section of memory
    - This is undefined behavior
  - What you call free on must be the result of malloc
    - (or calloc or realloc, but I won’t discuss those)

Allocators

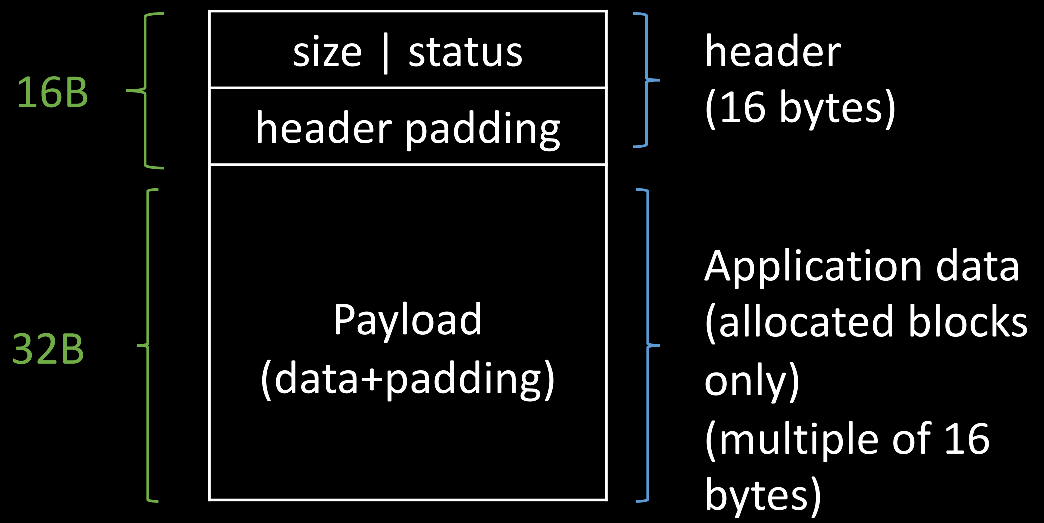
- Allocators can’t move data around
- How do you track what parts of the heap are freed or malloced? How do you track their sizes?
  - The trick is to store metadata along with the data in the heap to create a “linked list”
    - implicit list, explicit list
  - You store the status of the chunk (free or allocated), and the size of the data (which effectively points to the next chunk)
- When someone asks for memory, what do you give them?
  - There are a number of different strategies
- How to give back the memory?

Malloc using Implicit list

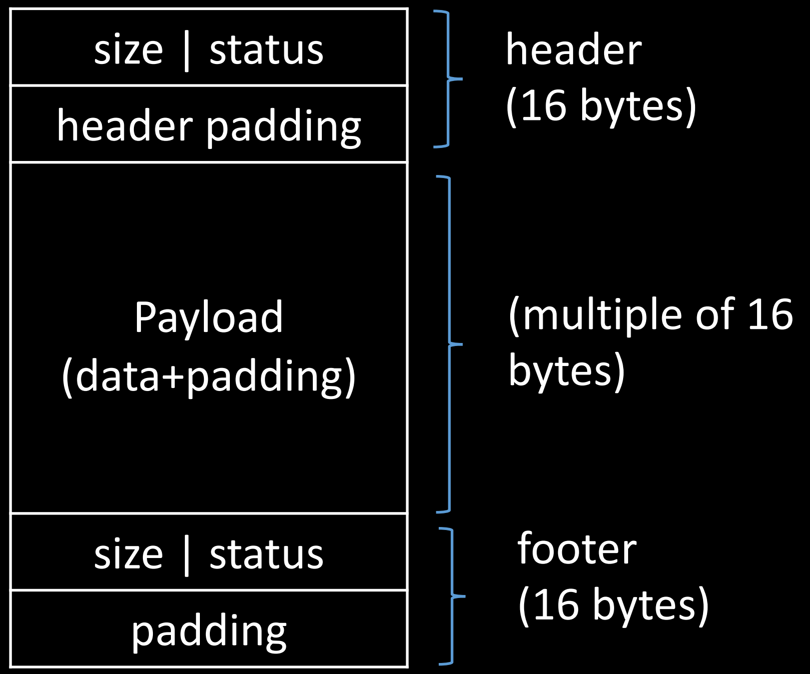
1. Structure of implicit list
2. Where to place an allocation?
3. Splitting a free block
4. Coalescing a free block

Malloc using Implicit list

- Structure of implicit list
  - Implicit list means that it does not use pointers explicitly, but it can find the next node just like a linked list.



e.g. p=malloc(20);



Malloc using Implicit list

- Where to place an allocation?
- Different algorithms:
  - First fit → easy & fast; cause fragmentation at beginning of the heap
  - Best fit → good for utilization; slower
  - Next fit → faster than first fit; even worse fragmentation

Malloc using Implicit list

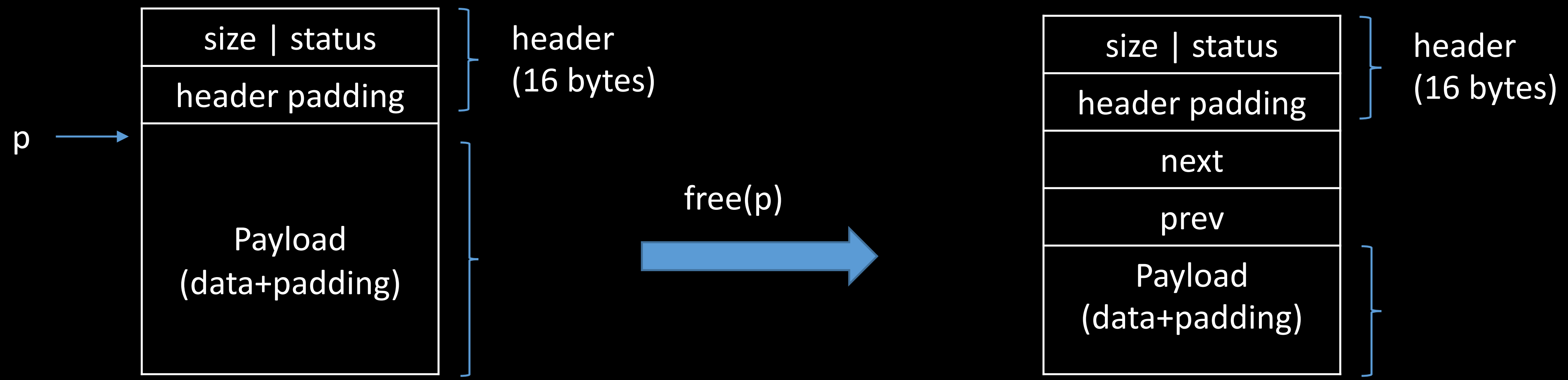
- Splitting a free block
  - Happens when we do memory allocation, e.g. p10=malloc(16)
  - (I have found a suitable free block)
  - ➢ find the next chunk (according to the size you want)
  - ➢ set size & status
- Coalescing a free block
  - Happens when we do free memory, e.g. free(p10)
  - After free, merge this free block with its next(& prev) free neighbor
  - ➢ find the next and prev chunk, and check its status
  - ➢ set size & status
    - ➢ (don’t forget the footer)

Implement Malloc using Implicit list

- Malloc <malloc(size)>
    - get the size of the chunk to allocate
    - find a free chunk
    - ask the OS for chunk
        - Split chunk if necessary
    - set this chunk status to be allocated
    - return pointer to the payload
- Free <free(p)>
    - go to the header from payload
    - set this chunk status to be free
    - coalesce

Malloc using Explicit free list

- Based on the implicit list, because the implicit list is too slow
- Structure of explicit free list
  - Maintain a linked list by adding 2 pointers: next & prev
    - points to the next/previous free chunk
    - only the free chunk needs to be recorded, and the allocated ones do not need



Implement Malloc using Explicit free list

- Malloc <malloc(size)>
  - get the size of the chunk to allocate
  - find a free chunk (in your linked list – linked list traverse)
    - delete this chunk from the linked list
  - ask the OS for chunk
    - Split chunk if necessary
    - insert the new free chunk to the linked list
  - set this chunk status to be allocated
  - return pointer to the payload
- Free <free(p)>
  - go to the header from payload
  - free the chunk
    - set this chunk status to be free
    - initial the next & prev pointer
  - coalesce
    - delete some chunk(s) from the linked list
  - insert this new free block into the linked list