Universidade Estadual de Campinas
Instituto de Computação

MC504 Sistemas Operacionais

T08

Memory Virtualization
Memory API

Referência principal
Ch.14 of Operating Systems: Three Easy Pieces by Remzi and Andrea Arpaci-Dusseau (pages.cs.wisc.edu/~remzi/OSTEP/)

Discutido em classe em

Arthur João Catto, PhD

2º semestre de 2018

Memory API

malloc()

```
#include <stdlib.h>

void* malloc(size_t size)
```

- Allocate a memory region on the heap.
  - Argument
    - `size_t size`: size of the memory block(in bytes)
    - `size_t` is an unsigned integer type.
  - Return
    - Success: a void type pointer to the memory block allocated by `malloc`
    - Fail: a null pointer

Memory API

sizeof()

- Functions and macros are used to provide the `size` parameter in `malloc` instead typing in a number directly.

- You may get different `sizeof` results even with similar variables, e.g.
  - The area is in the heap and its actual size is defined at run-time.

```
int *x = malloc(10 * sizeof(int));
printf("%d %p\n", sizeof(x), x);
```

```
8
```

  - The area is in the stack and its actual size is defined at compile-time.

```
int x[10];
printf("%d %p\n", sizeof(x), x);
```

```
40
```

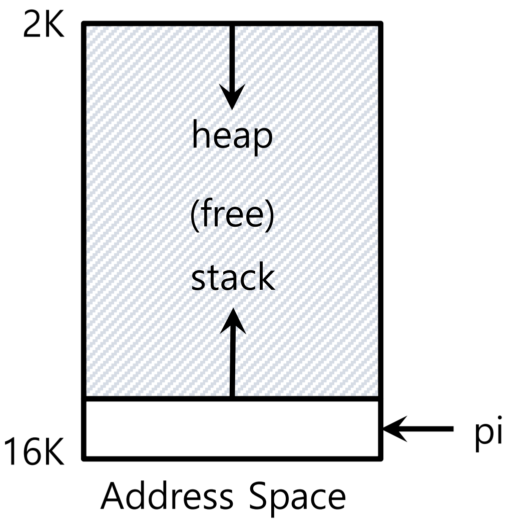
Memory API

free()

```
#include <stdlib.h>

void free(void* ptr)
```

- Free a memory region allocated by a call to `malloc`.
  - Argument
    - `void *ptr`: a pointer to a memory block allocated by `malloc`
  - Return
    - none

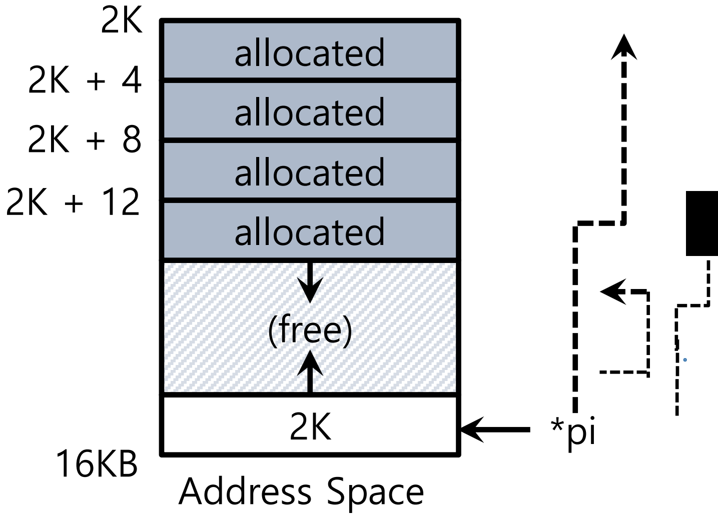
Memory Allocation



⇢ points to

→ Is here

```
int *pi; // local variable
```

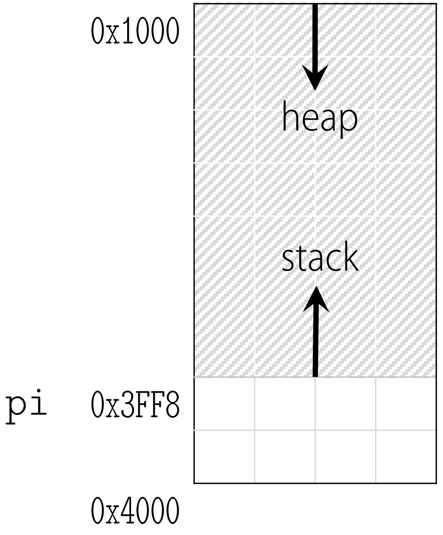


```
pi = (int *) malloc(sizeof(int) * 4);
```

Example

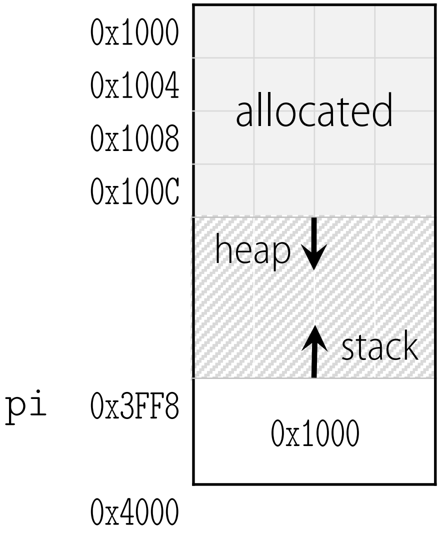
Memory allocation on a very small machine

```
int *pi; // local variable
```



Address Space

```
pi = (int *) malloc(sizeof(int) * 4);
```



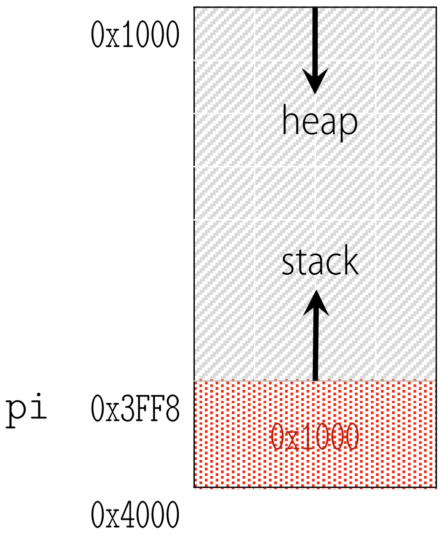
Address Space

Example

Freeing dynamically allocated memory

```
free(pi);
```

The value of pi is now invalid but is still there!



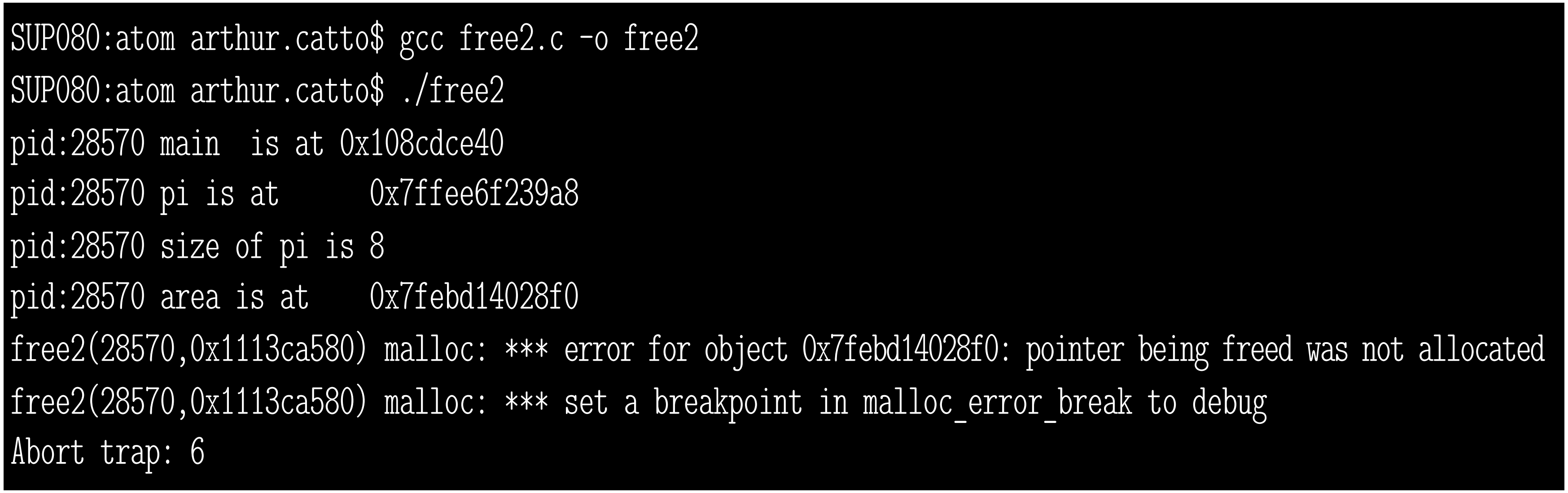
Address Space

Example

Freeing dynamically allocated memory

The process will be aborted if you try to free a dynamically allocated area twice.

```
  free(pi);
  free(pi);
```

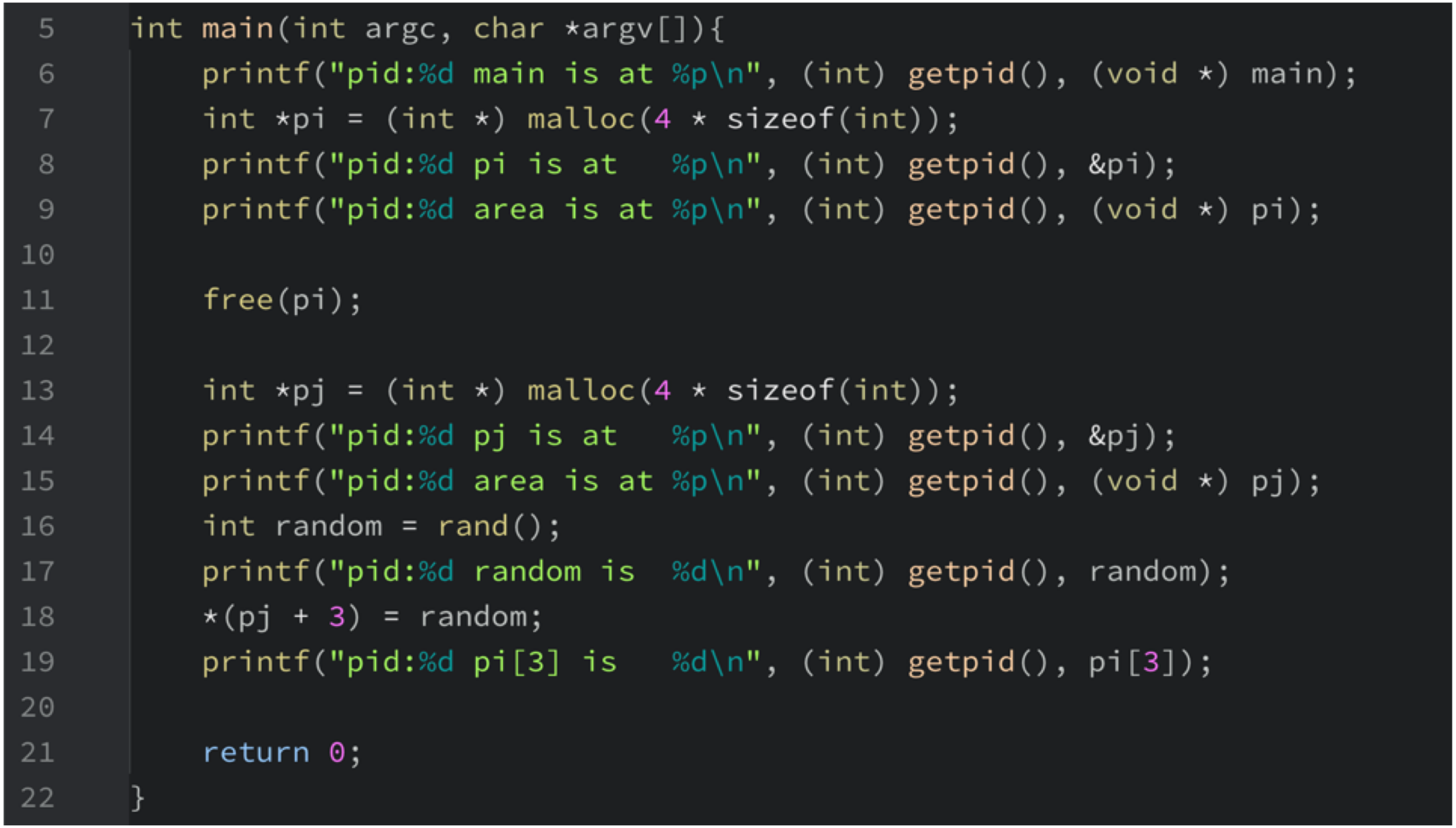


```
SUP080:atom arthur.catto$ gcc free2.c -o free2
SUP080:atom arthur.catto$ ./free2
pid:28570 main  is at 0x108cdce40
pid:28570 pi is at      0x7ffee6f239a8
pid:28570 size of pi is 8
pid:28570 area is at    0x7febd14028f0
free2(28570,0x1113ca580) malloc: *** error for object 0x7febd14028f0: pointer being freed was not allocated
free2(28570,0x1113ca580) malloc: *** set a breakpoint in malloc_error_break to debug
Abort trap: 6
```

Example

Freeing dynamically allocated memory

- What happens when you reuse a pointer that has been freed?

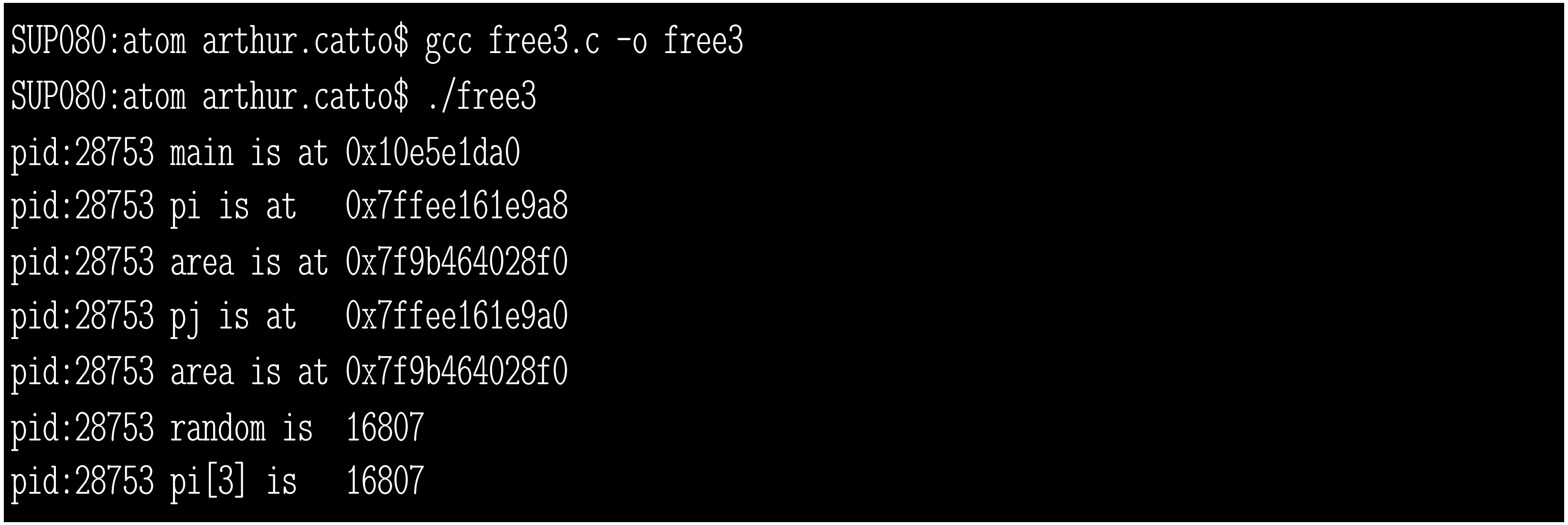


```c
int main(int argc, char *argv[]){
    printf("pid:%d main is at %p\n", (int) getpid(), (void *) main);
    int *pi = (int *) malloc(4 * sizeof(int));
    printf("pid:%d pi is at   %p\n", (int) getpid(), &pi);
    printf("pid:%d area is at %p\n", (int) getpid(), (void *) pi);

    free(pi);

    int *pj = (int *) malloc(4 * sizeof(int));
    printf("pid:%d pj is at   %p\n", (int) getpid(), &pj);
    printf("pid:%d area is at %p\n", (int) getpid(), (void *) pj);
    int random = rand();
    printf("pid:%d random is  %d\n", (int) getpid(), random);
    *(pj + 3) = random;
    printf("pid:%d pi[3] is   %d\n", (int) getpid(), pi[3]);

    return 0;
}
```

Example

Freeing dynamically allocated memory

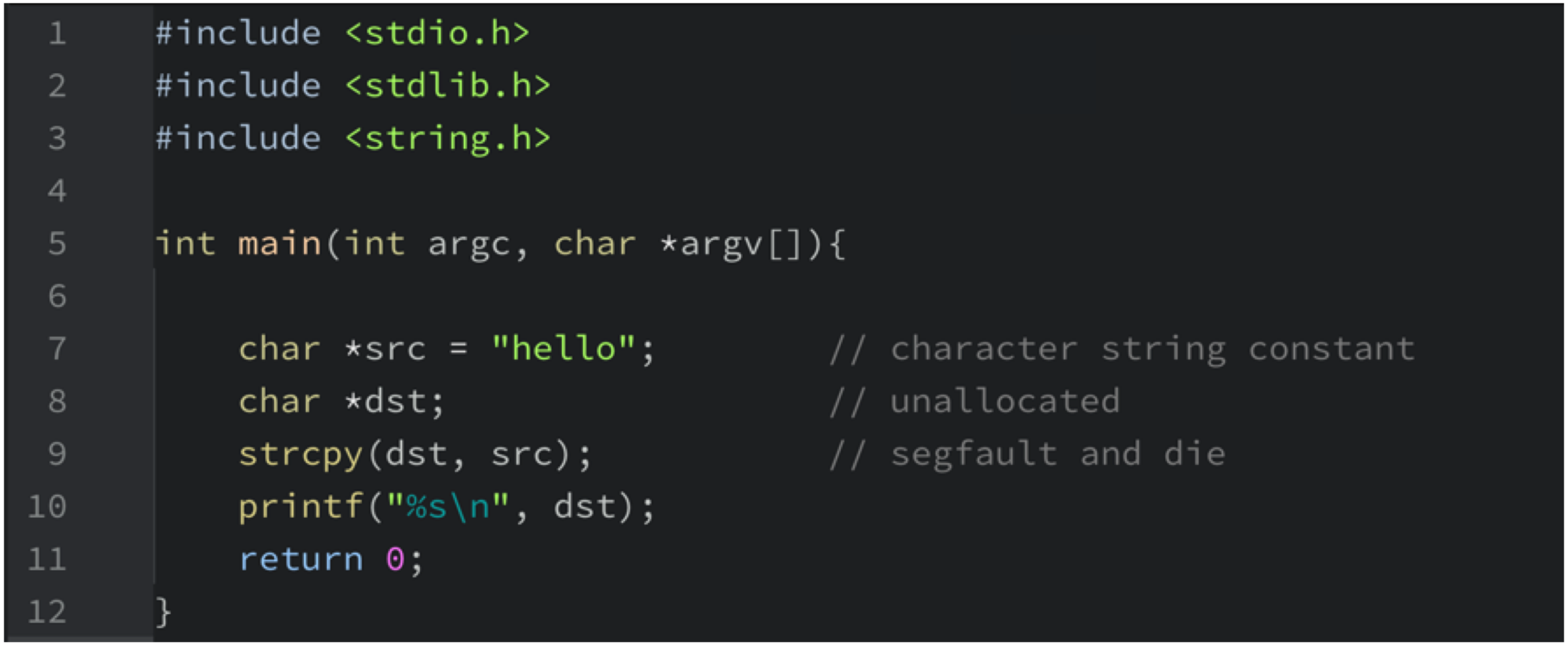
- What happens when you reuse a pointer that has been freed?
  - The system may be quite willing to do it!



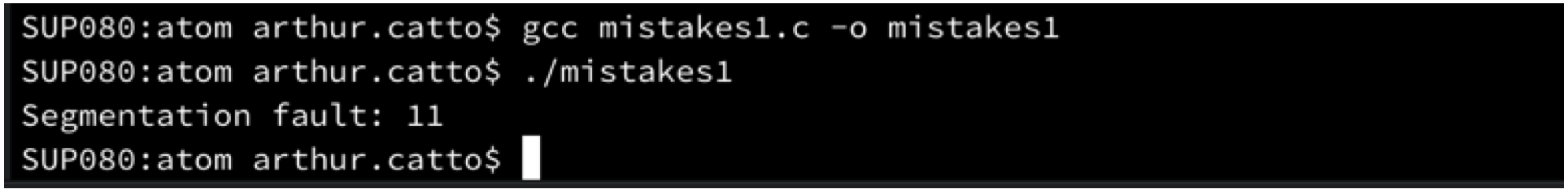
```
SUP080:atom arthur.catto$ gcc free3.c -o free3
SUP080:atom arthur.catto$ ./free3
pid:28753 main is at 0x10e5e1da0
pid:28753 pi is at   0x7ffee161e9a8
pid:28753 area is at 0x7f9b464028f0
pid:28753 pj is at   0x7ffee161e9a0
pid:28753 area is at 0x7f9b464028f0
pid:28753 random is  16807
pid:28753 pi[3] is   16807
```

Common mistakes

Forgetting to Allocate Memory



```c
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

int main(int argc, char *argv[]){

    char *src = "hello";        // character string constant
    char *dst;                  // unallocated
    strcpy(dst, src);           // segfault and die
    printf("%s\n", dst);
    return 0;
}
```

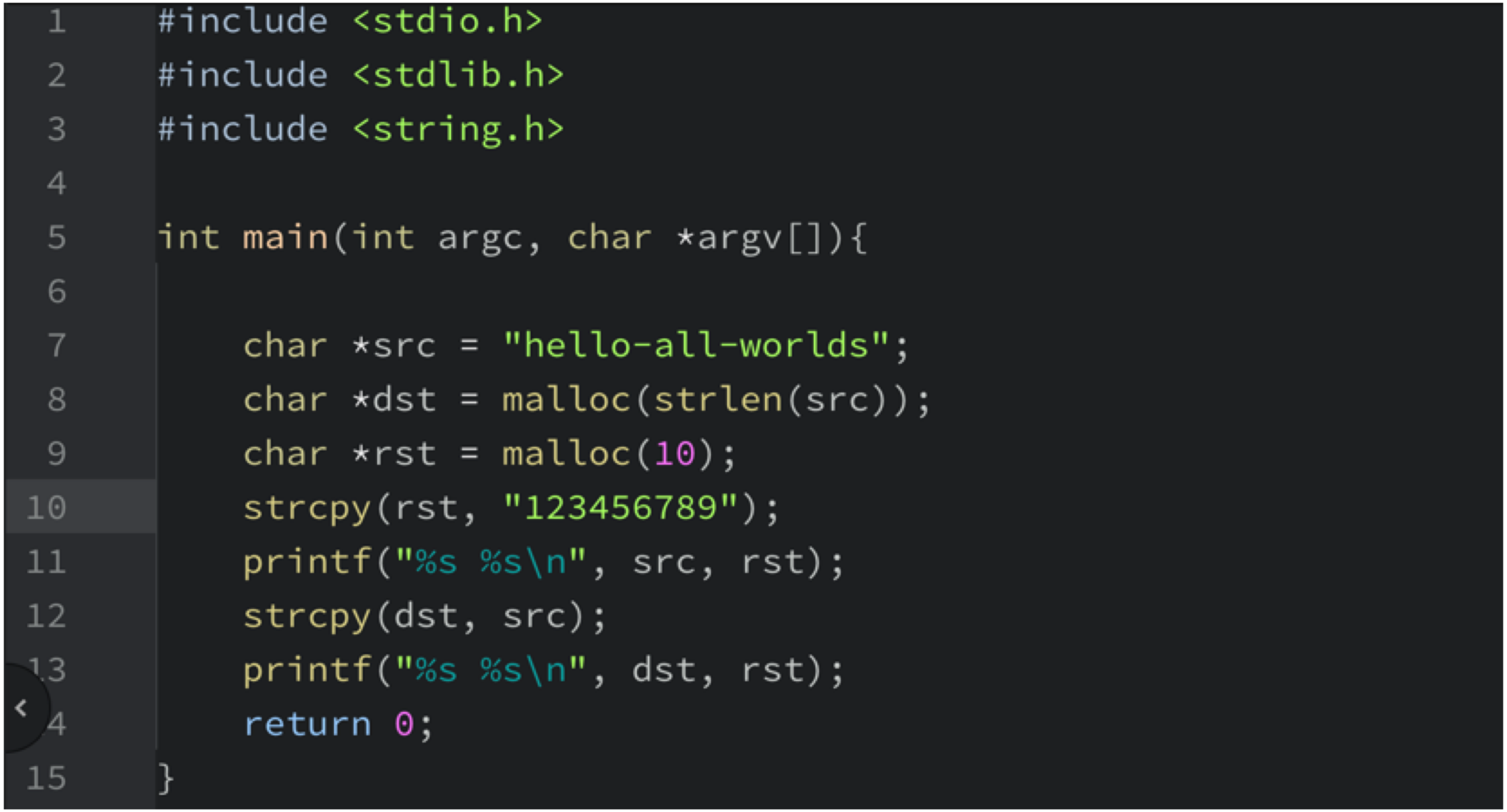


```
SUP080:atom arthur.catto$ gcc mistakes1.c -o mistakes1
SUP080:atom arthur.catto$ ./mistakes1
Segmentation fault: 11
SUP080:atom arthur.catto$ 
```

Common mistakes

Not Allocating Enough Memory

- What's the output of this program?

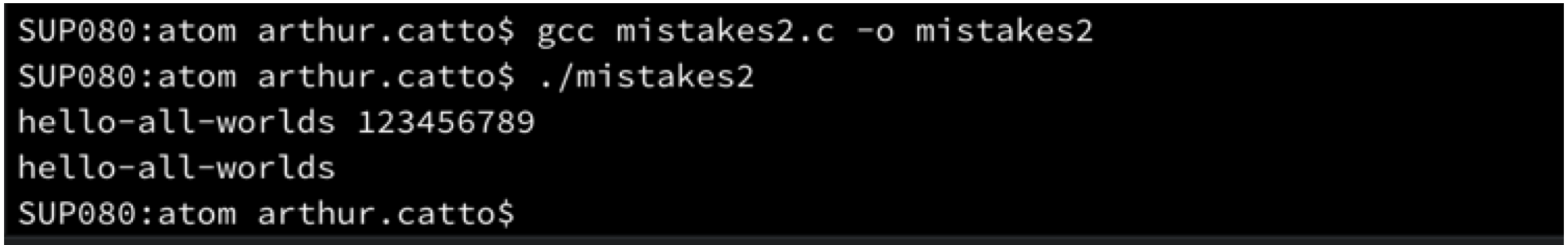


```c
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

int main(int argc, char *argv[]){

    char *src = "hello-all-worlds";
    char *dst = malloc(strlen(src));
    char *rst = malloc(10);
    strcpy(rst, "123456789");
    printf("%s %s\n", src, rst);
    strcpy(dst, src);
    printf("%s %s\n", dst, rst);
    return 0;
}
```

Common mistakes

Not Allocating Enough Memory

- Where has **`rst`** gone?

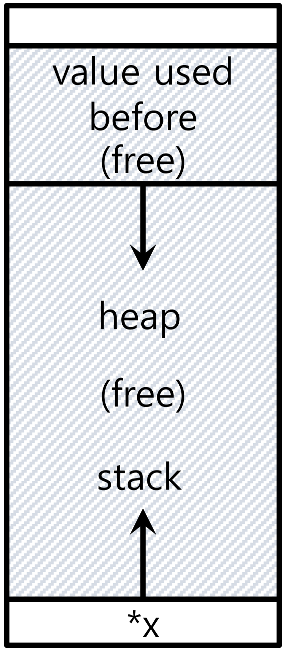


```
SUP080:atom arthur.catto$ gcc mistakes2.c -o mistakes2
SUP080:atom arthur.catto$ ./mistakes2
hello-all-worlds 123456789
hello-all-worlds
SUP080:atom arthur.catto$
```

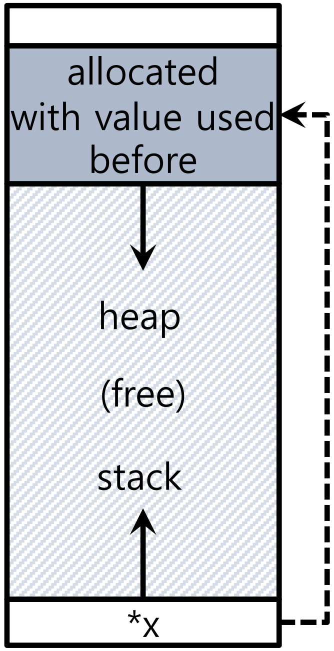
Forgetting to Initialize

- Encounter an uninitialized read

```
int *x = (int *)malloc(sizeof(int)); // allocated
printf(“*x = %d\n”, *x); // uninitialized memory access
```



Address Space

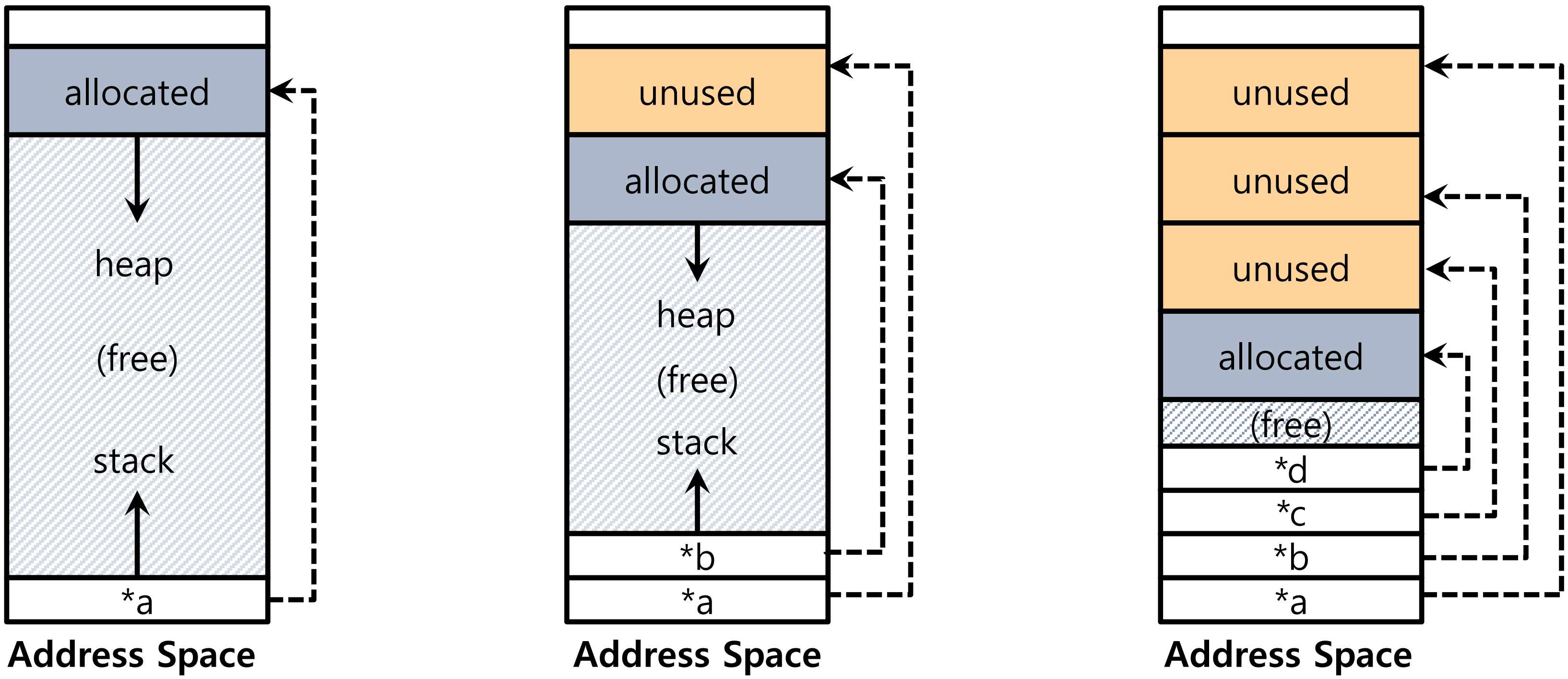


Address Space

Memory Leak

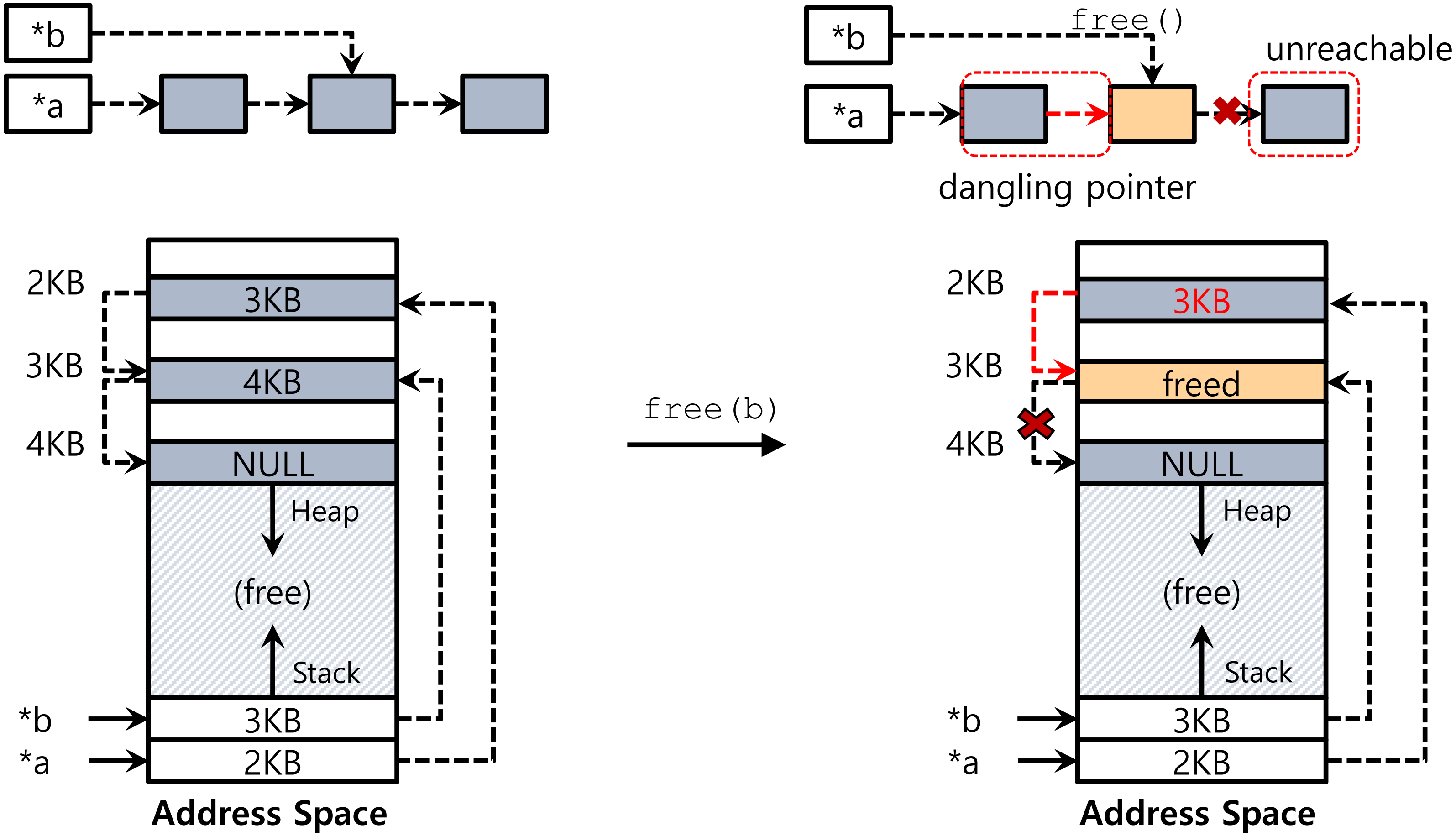
- A program runs out of memory and eventually dies.

unused : unused, but not freed



Dangling Pointer

- Freeing memory that is still needed
  - A program accesses memory with an invalid pointer



Other Memory APIs

calloc()

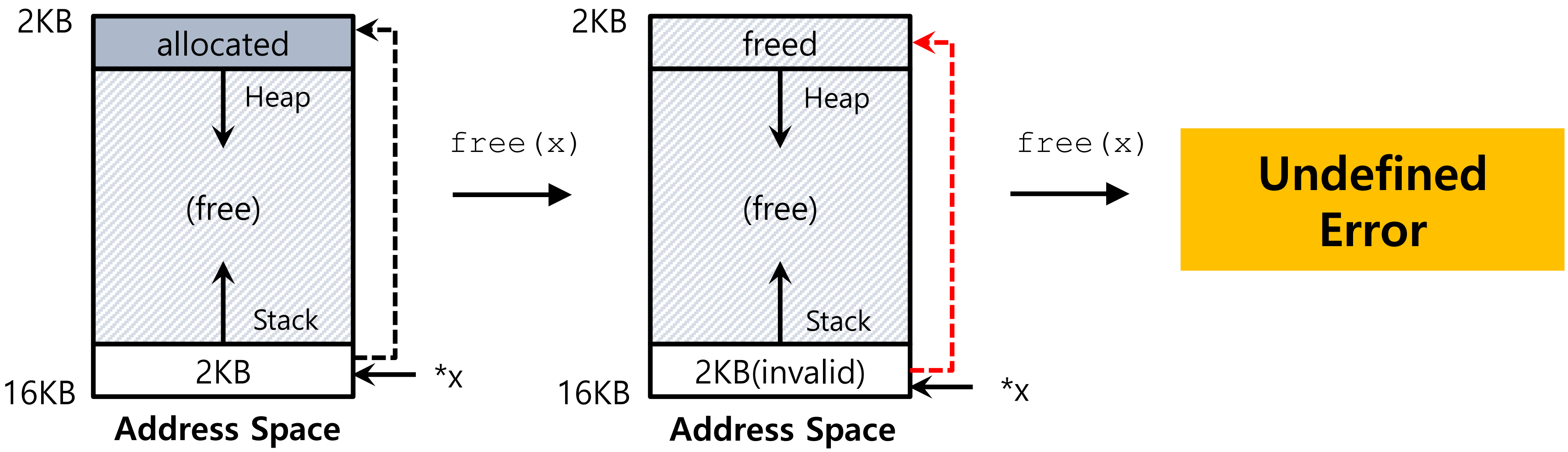
```
#include <stdlib.h>

void *calloc(size_t num, size_t size)
```

- Allocate memory on the heap and zeroes it before returning.
  - Argument
    - `size_t num` : number of blocks to allocate
    - `size_t size` : size of each block(in bytes)
  - Return
    - Success: a void type pointer to the memory block allocated by `calloc`
    - Fail: a null pointer

Free memory that was freed already.

Double Free

```
int *x = (int *)malloc(sizeof(int)); // allocated
free(x); // free memory
free(x); // free repeatedly
```



2KB

allocated

Heap

(free)

Stack

2KB

16KB

*x

Address Space

free(x)

2KB

freed

Heap

(free)

Stack

2KB(invalid)

16KB

*x

Address Space

free(x)

Undefined Error

Other Memory APIs

realloc()

```
#include <stdlib.h>

void *realloc(void *ptr, size_t size)
```

- Change the size of memory block.
  - A pointer returned by `realloc` may be either the same as `ptr` or a new one.
  - Argument
    - `void *ptr`: Pointer to memory block allocated with `malloc, calloc` or `realloc`
    - `size_t size`: New size for the memory block(in bytes)
  - Return
    - Success: Void type pointer to the memory block
    - Fail : Null pointer