COMPUTER SYSTEMS AND ORGANIZATION

Allocators and Header Files

Daniel G. Graham Ph.D

ENGINEERING

1. Locations in memory (More Details)
2. Malloc Examples
3. Allocators Implementing Malloc
4. Creating our own C library
5. Header files
6. Files IO operations
7. Memory Errors Next time.

STACK VS HEAP

	Permissions	Contents	Managed by
Stack	Read/Write	Local vars etc	Compiler
Heap	Read/Write	Dynamic structures	Programmer: malloc/free
Statics	Read/Write	Global Vars	Compiler
Literals	Read	String Literals	Compiler
Text	Execute	Instructions	Compiler

University of Virginia | ENGINEERING

SCANF (INPUT)

```
int scanf(const char *format, ...)
```

1. ... Indicates that the function accepts variable length arguments
2. char *: This contains the *format specifiers*

University of Virginia | ENGINEERING

SCANF EXAMPLES

```
int a;

printf("Enter a number: ");
scanf("%d", &a);
```


UNIVERSITY of VIRGINIA | ENGINEERING

SCANF EXAMPLES

```
int a;
int b;

printf("Enter two numbers ");
scanf("%d %d", &a, &b);
```


UNIVERSITY of VIRGINIA | ENGINEERING

DON'T DO THIS

```
#include <stdio.h>
#include <stdlib.h>
#define MAXN 15213

int array[MAXN]

int main(){
      int i, n;
      scanf("%d, &n);
      If (n > MAXN){
           app_error("Input file too big");
      }
      for( i = 0; i <n; i++){
         scanf("%d". &array[i]);
       }
}
```

Draw the stack.

University of Virginia | ENGINEERING

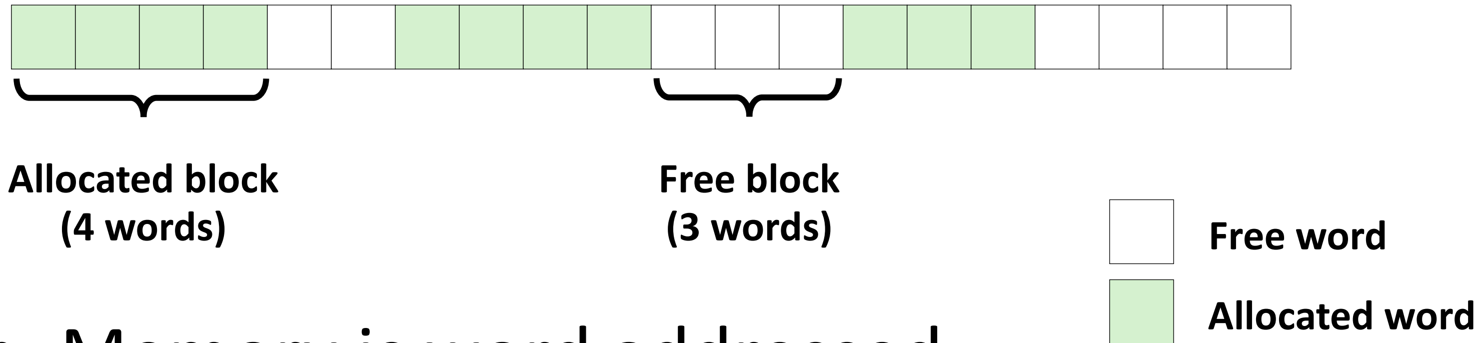
DO THIS INSTEAD

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

int main(){
      int *array, i, n;
      scanf("%d, &n);
      array =  (int *) malloc(n*sizeof(int));
      for( i = 0; i <n; i++){
         scanf("%d". &array[i]);
       }
}
```


UNIVERSITY of VIRGINIA | ENGINEERING

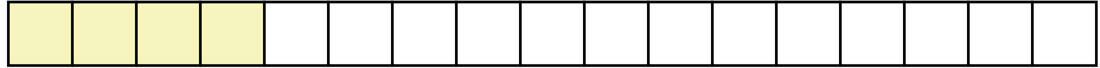
WE'LL ASSUME WORD ADDRESS NOT BYTE ADDRESS



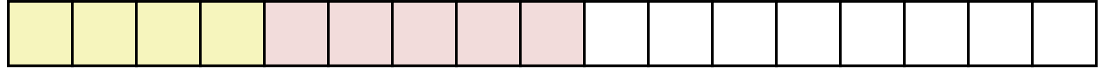
- Memory is word addressed.
- Words are int-sized.

UNIVERSITY of VIRGINIA ENGINEERING

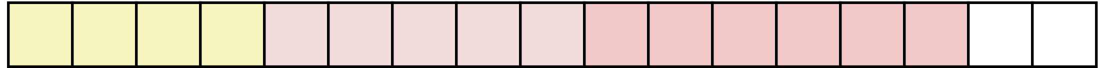
`p1 = malloc(4)`



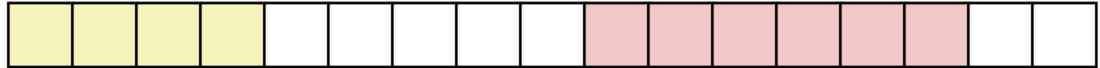
`p2 = malloc(5)`



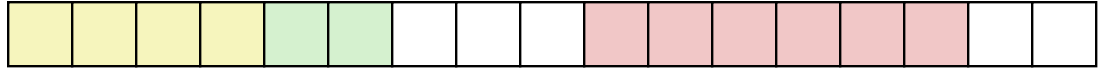
`p3 = malloc(6)`



`free(p2)`



`p4 = malloc(2)`



LET'S TALK ABOUT HOW WE COULD BUILD AND ALLOCATOR

Allocators

- Can't control the number or size of allocated blocks
- Must allocate blocks from free memory
- Can manipulate and modify only free memory
- Can't move the allocated blocks once they are `malloc`'d

UNIVERSITY of VIRGINIA | ENGINEERING

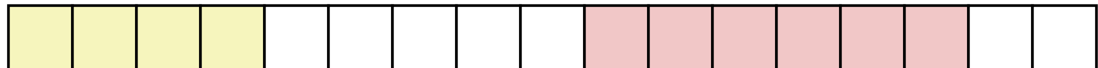
ALSO, WANT TO REDUCE FRAGMENTATION

`p1 = malloc(4)`

`p2 = malloc(5)`

`p3 = malloc(6)`

`free(p2)`



`p4 = malloc(6)`

Oops! (what would happen now?)

Malloc turns NULL (No space left ☹)

UNIVERSITY of VIRGINIA | ENGINEERING

IMPLEMENTATION CHALLENGES

1. How do we know how much memory to free given just a pointer?

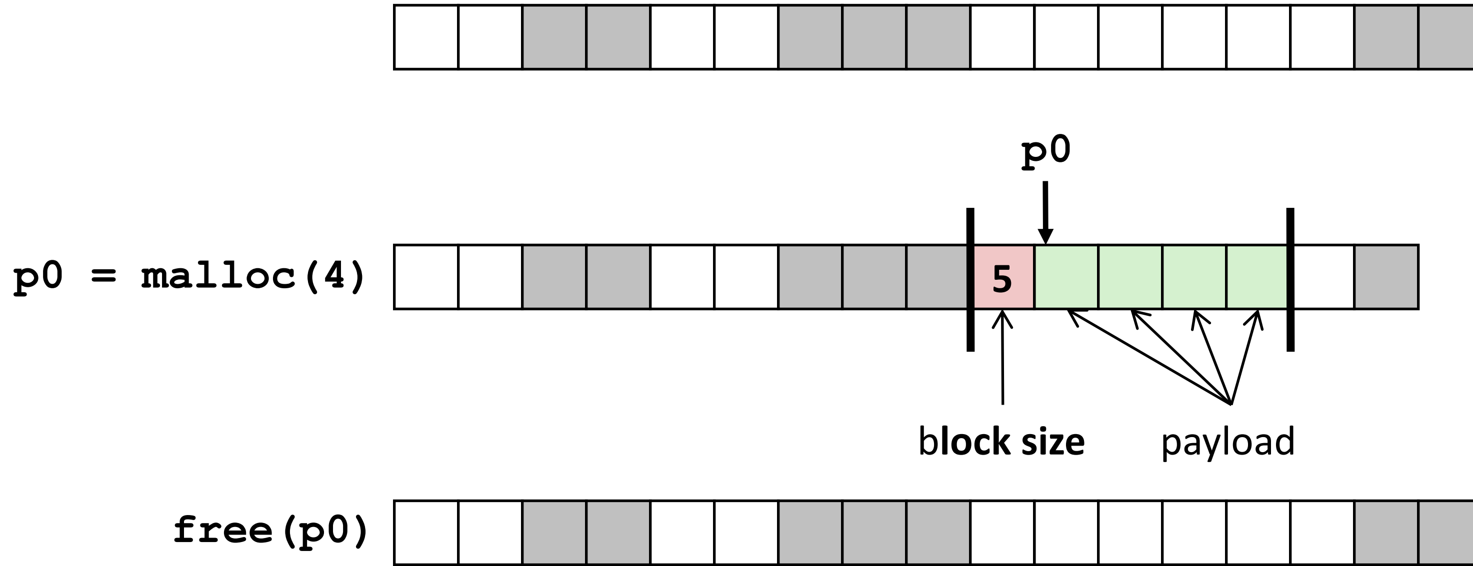
2. How do we keep track of the free blocks?

3. How do we pick a block to use for allocation?
    1. Best Fit
    2. Next Fit
        1. Sadly both still have fragmentation.

KNOWING HOW MUCH TO FREE

Keep the length of a block in the word preceding the block.

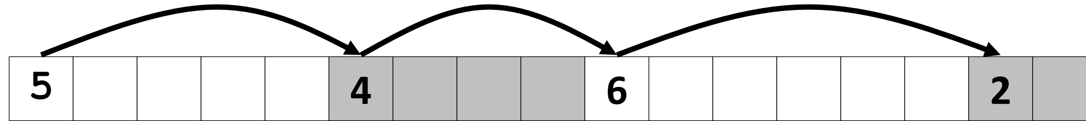
This word is often called the ***header field*** or ***header***



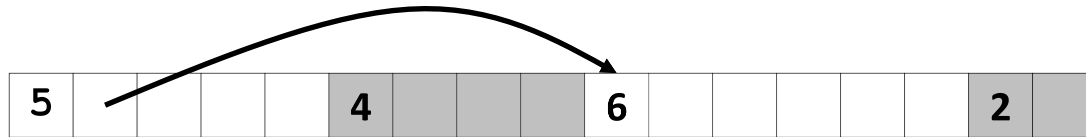
UNIVERSITY of VIRGINIA | ENGINEERING

KEEP TRACK OF FREE BLOCKS

Method 1: *Implicit list* using length—links all blocks



Method 2: *Explicit list* among the free blocks using pointers



UNIVERSITY of VIRGINIA | ENGINEERING

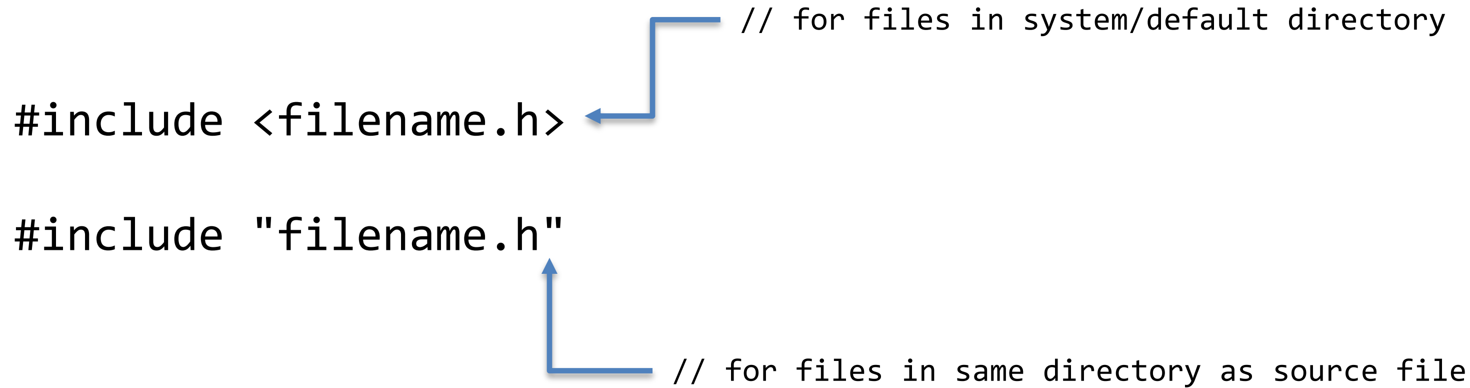
GNU nano 6.3 crypto.c

```
/**
This is library runs implements
Simple Crypto Library (Don't write your own crypto libraries)
//For onetime pad the key needs  to be random and they same lenght
// as the message this is a bad implementation of one-time pad
//Because the key is reused for different parts of the message
*/
#include <string.h>
#include <stdio.h>

void encrypt(char * input_string, const char * key ){
        int length_of_key = strlen(key);
        for (int i = 0; i < strlen(input_string); i++){
                input_string[i] =  input_string[i] ^ key[i%length_of_key];
        }
}
```


UNIVERSITY of VIRGINIA | ENGINEERING

HEADER FILES



UNIVERSITY of VIRGINIA | ENGINEERING

WHY DO HEADER FILES EXIST?

You can think of header files as an API that provides the method definition that tells us so we can use a library

UNIVERSITY of VIRGINIA | ENGINEERING

EXAMPLE HEADER FILES

```
#ifndef EXAMPLE_HEADER_FILE
#define EXAMPLE_HEADER_FILE

----- Contents of Header file.


#endif
```

Header Guard

UNIVERSITY of VIRGINIA | ENGINEERING

EXAMPLE HEADER FILES

```
#ifndef EXAMPLE_HEADER_FILE
#define EXAMPLE_HEADER_FILE

/** Encrypts a string using a key
@param input_string String to be encrypted
@part key key to use for encryption.
**/
#include<string.h>
  void encrypt(char * input_string, const char * key );


#endif
```


UNIVERSITY of VIRGINIA | ENGINEERING

USING THE LIBRARY

```
  GNU nano 6.3                         main.c                          Modified
#include <stdio.h>
#include <string.h>
#include "crypto.h"




int main(){
        char * message = strdup("A secure, connected and intelligent future");
        char * key = strdup("Love");
        encrypt(message, key);
}
```


UNIVERSITY of VIRGINIA | ENGINEERING

FILE IO

```
FILE *fopen( const char * filename, const char * mode );
```


UNIVERSITY of VIRGINIA | ENGINEERING

FILE IO

r	Opens an existing text file for reading purposes.
w	Opens a text file for writing. If it does not exist, then a new file is created. Here your program will start writing content from the beginning of the file.
a	Opens a text file for writing in appending mode. If it does not exist, then a new file is created. Here your program will start appending content in the existing file content.
r+	Opens a text file for both reading and writing.
w+	Opens a text file for both reading and writing. It first truncates the file to zero length if it exists, otherwise creates a file if it does not exist.
a+	Opens a text file for both reading and writing. It creates the file if it does not exist. The reading will start from the beginning but writing can only be appended.

UNIVERSITY of VIRGINIA | ENGINEERING

WHAT IS THIS FILE TYPE

```
FILE *fopen( const char * filename, const char * mode );
```

Let's look at the header file. (It is open source)

Notice that NULL is defined.

https://www.gnu.org/software/m68hc11/examples/stdio_8h-source.html

UNIVERSITY of VIRGINIA | ENGINEERING

FILE IO

```
int fclose( FILE *fp );
```


UNIVERSITY of VIRGINIA | ENGINEERING

FIO EXAMPLE

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

int main()
{
    // file pointer variable to store the value returned by
    // fopen
    FILE* fptr;

    // opening the file in read mode
    fptr = fopen("filename.txt", "r");

    // checking if the file is opened successfully
    if (fptr == NULL) {
        printf("The file is not opened. The program will "
               "now exit.");
        exit(0);
    }

    return 0;
}
```