# EEE243 – Applied Computer Programming

Memory Model

ROYAL MILITARY COLLEGE OF CANADA
ELECTRICAL & COMPUTER ENGINEERING

GÉNIE ÉLECTRIQUE ET GÉNIE INFORMATIQUE
COLLÈGE MILITAIRE ROYAL DU CANADA

RMC CMR

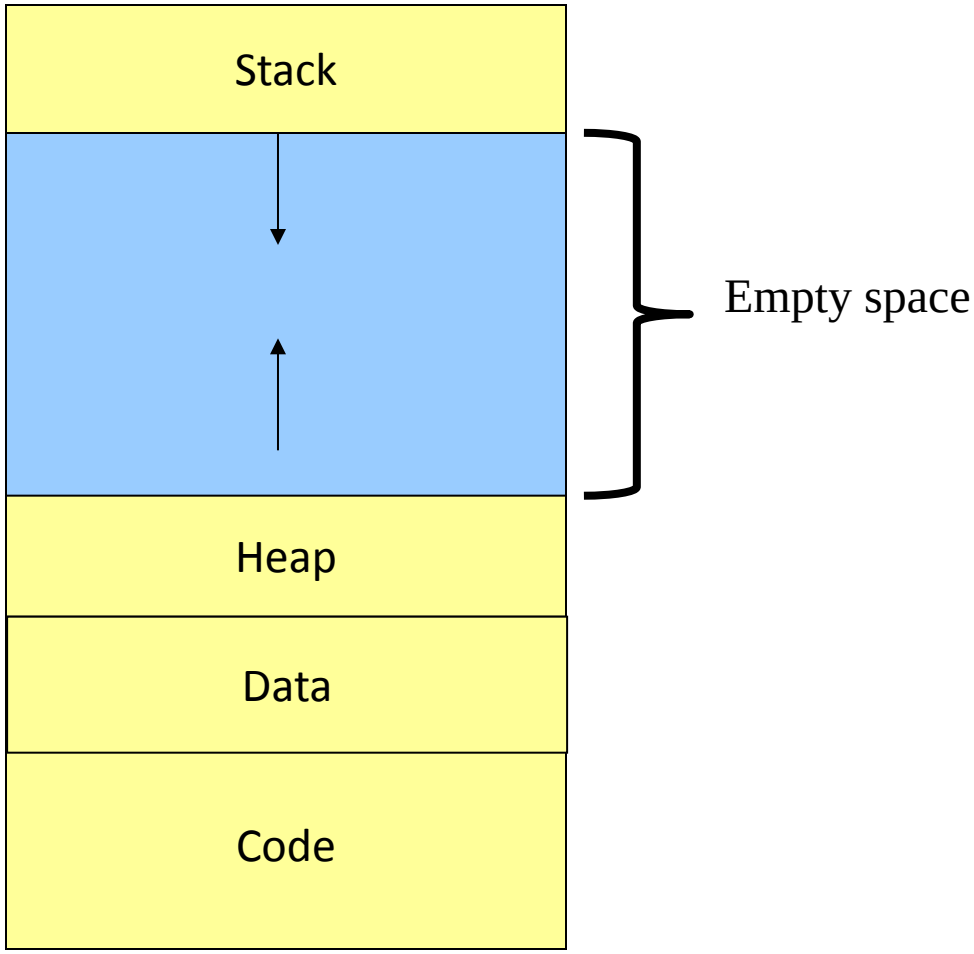
# Outline

1. Memory model
2. Malloc
3. Void Pointers
4. Casting
5. Free
6. Dynamic Array Example
7. Constants

# Memory model

- A program in C has four main segments:
  - Code segment
    - Your program
  - Data
    - Static/global data
  - Heap
    - Dynamic memory
  - Stack
    - Automatic segment
    - Stores local function variables (for `main()` and others)

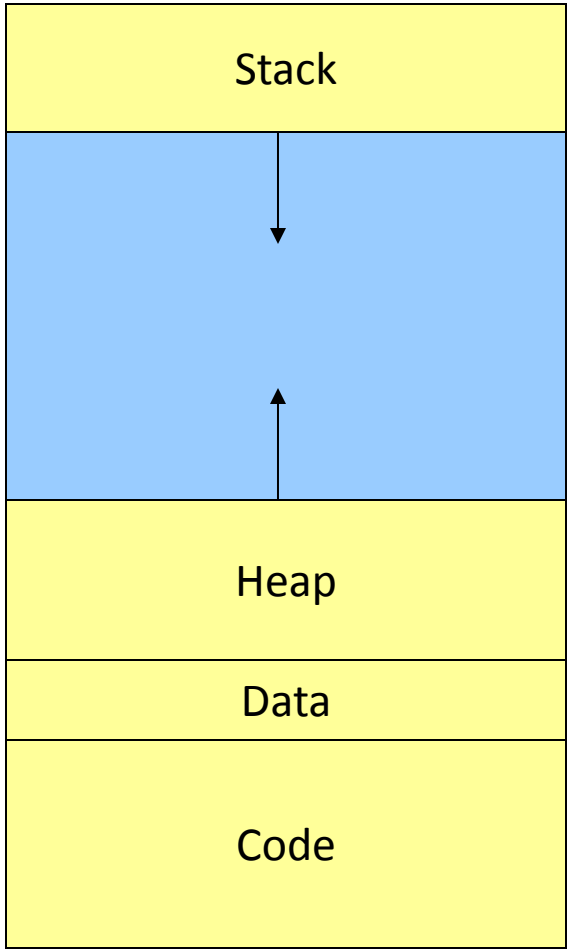
# Memory model



# Memory model

Where are those lines located?

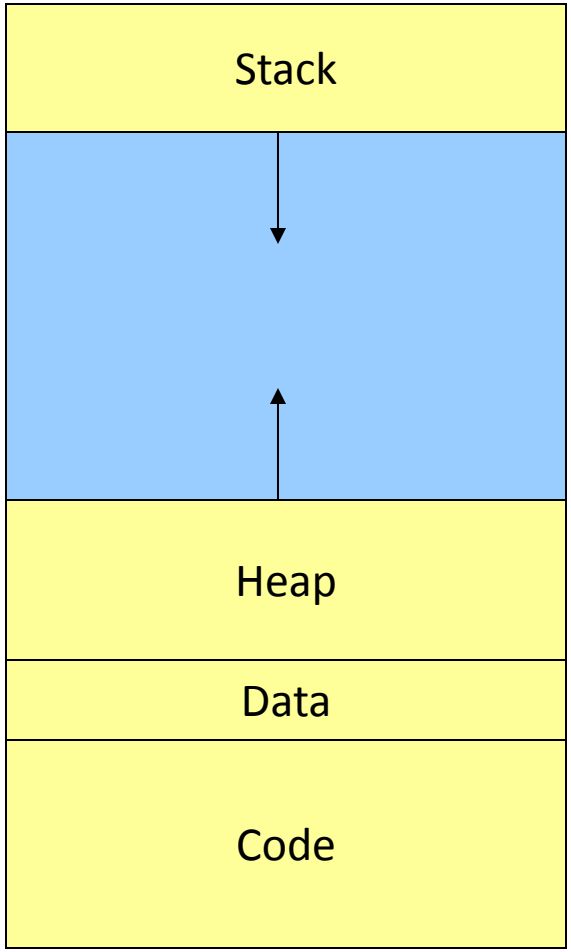
```
int i = 0;

void main() {
  int j = 0;
}

int fctn(){
   int l = 0;
   static int k = 0;
}
```



# Memory model

Where are those lines located?

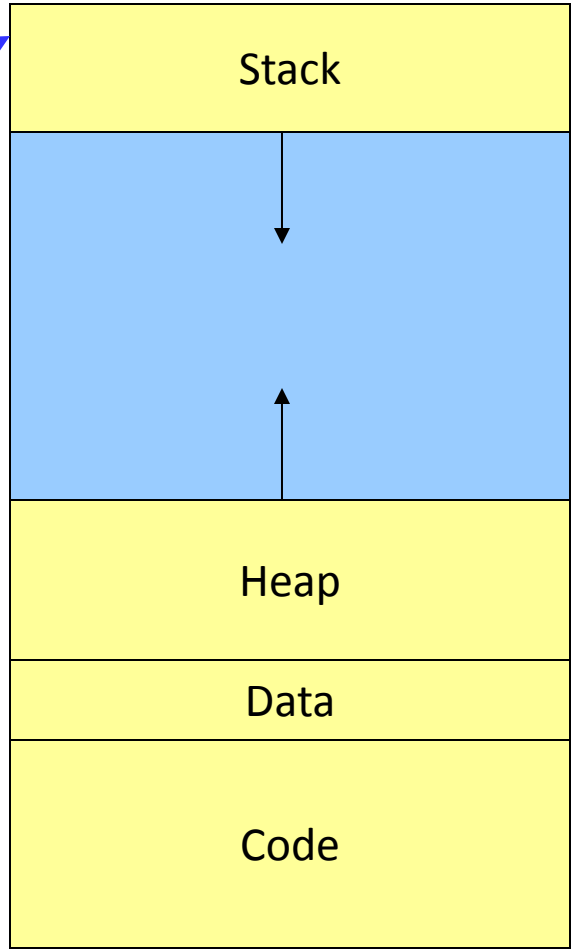
```
int i = 0;

void main() {
  int j = 0;
}

int fctn(){
   int l = 0;
   static int k = 0;
}
```



# Memory model

Where are those lines located?

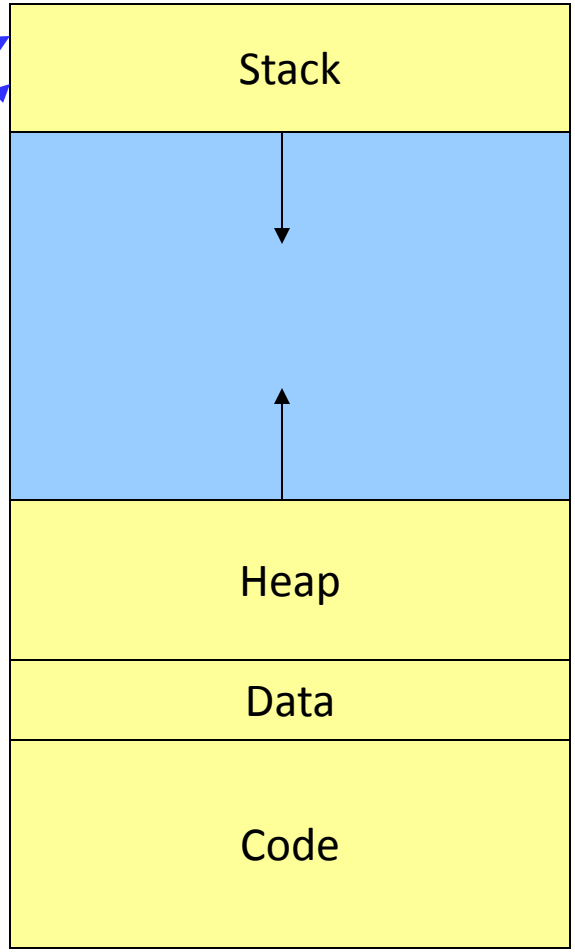
```
int i = 0;

void main() {
  int j = 0;
}

int fctn(){
   int l = 0;
   static int k = 0;
}
```



# Memory model

Where are those lines located?

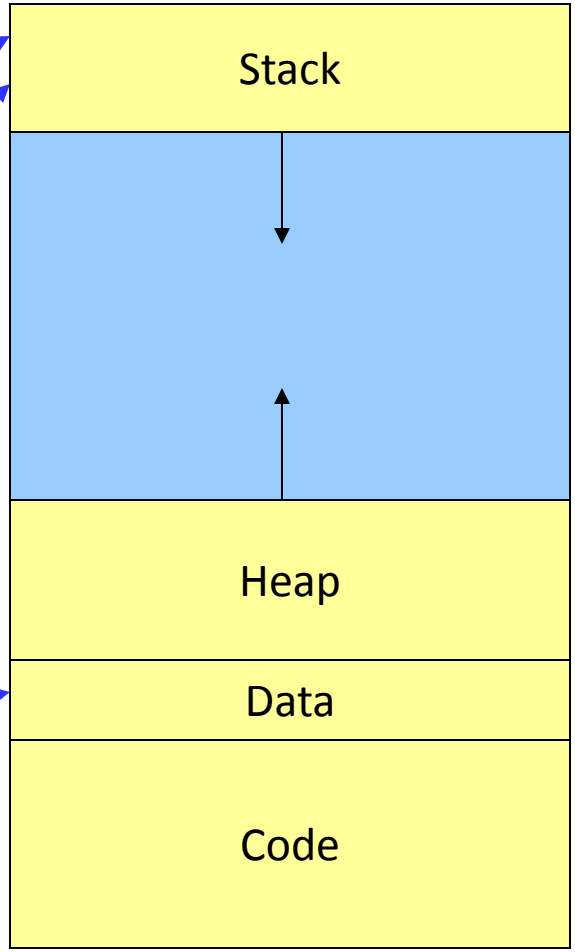
```
int i = 0;

void main() {
  int j = 0;
}

int fctn(){
   int l = 0;
   static int k = 0;
}
```



# Memory model

Where are those lines located?

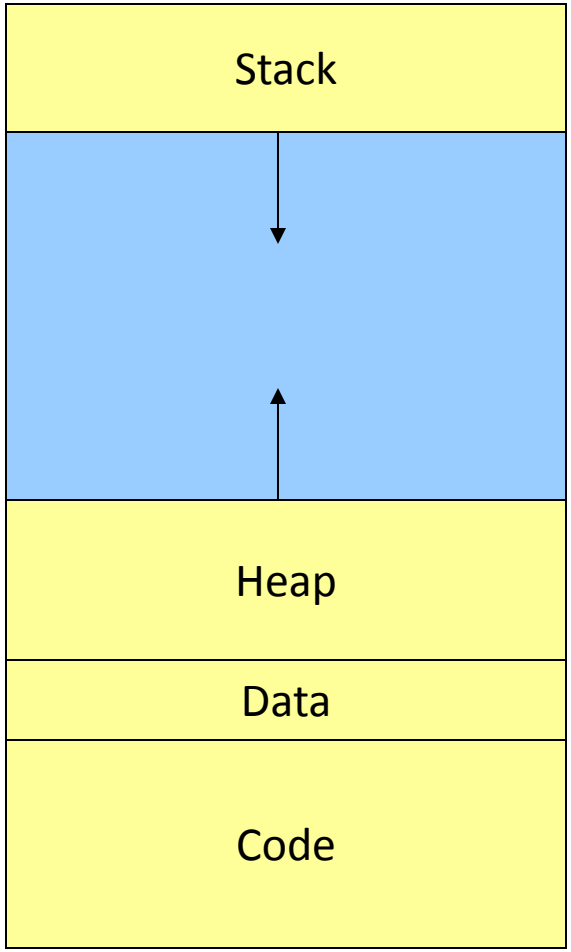
```
int i = 0;

void main() {
  int j = 0;
}

int fctn(){
   int l = 0;
   static int k = 0;
}
```



# Memory model

Where are those lines located?

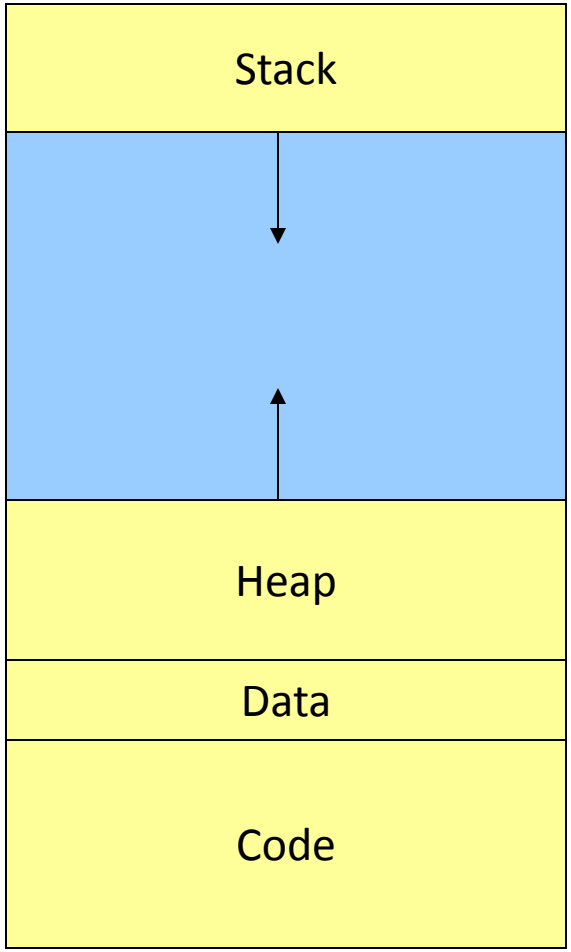
```
int i = 0;

void main () {
  int j = 0;
   int k=5;
  j = j + k;
}

int fctn () {
   int l = 0;
   static int k = 2;
   k = k *k;
}
```



# Memory model

Where are those lines located?

```
int i = 0;

void main () {
  int j = 0;
  j = j + 5;
}

int fctn () {
   int l = 0;
   static int k = 0;
   k = k + 10;
}
```



# Strings and Pointers

How do we assign a length to a string declared as a pointer?

30 physical char space allocated

```
int main() {
   char hello[30] = "Hello World!\n";
   char *hello_ptr = "Hello pointers!\n";
   return EXIT_SUCCESS;
}
```

17 physical char space allocated

What if we want to assign a specific amount of memory to string declared as a pointer?

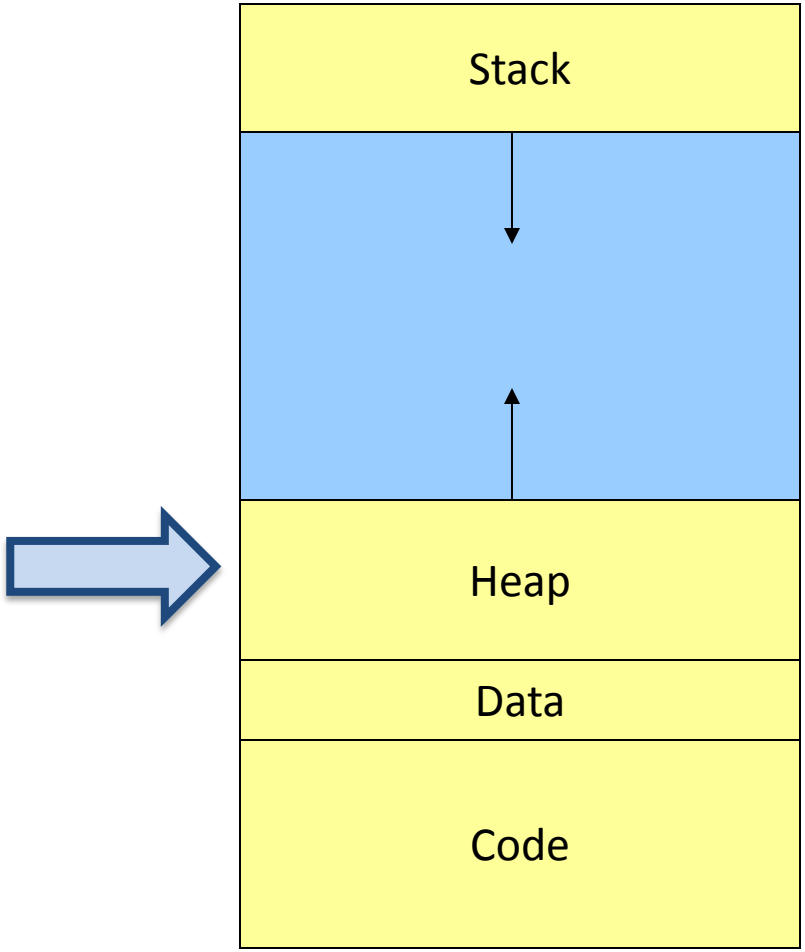
# Dynamic memory allocation

- The `malloc` function
  - returns a block of memory that contains the number of bytes specified in its parameter.
  - returns a `void` pointer to the first byte of the block of newly allocated memory
- Allocated memory contains garbage

```
int *p_int = NULL;
p_int = (int*)malloc (sizeof(int));
```

# Dynamic memory allocation

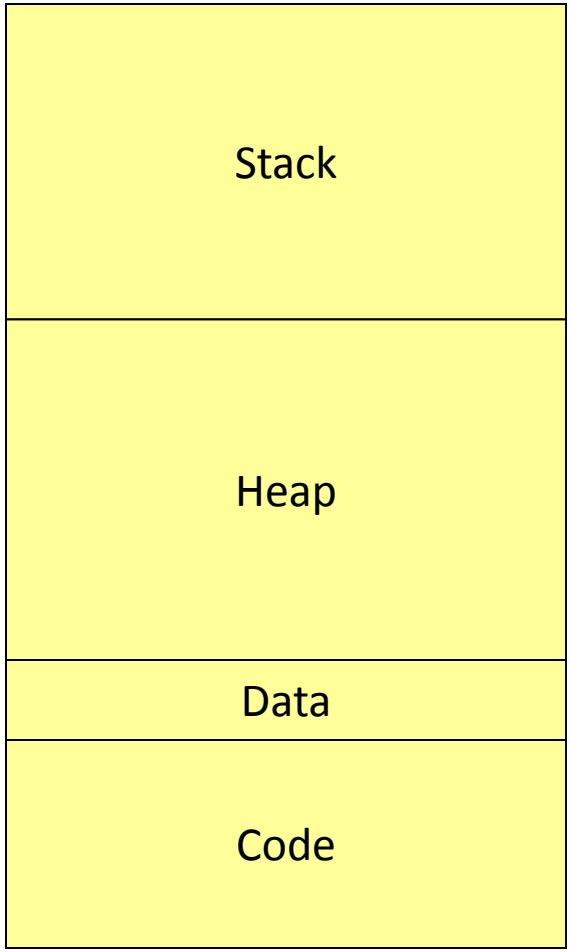
A call to `malloc` requests memory from the heap.



# Dynamic memory allocation

- If there is not enough memory on the program heap you get what is called an *overflow*
  - It is up to the programmer to handle an overflow if it occurs

```
p_int = (int*)malloc(sizeof(int))
if(p_int == NULL) {
    return EXIT_FAILURE;
}
```



# void Pointers and Casting

- C does not generally allow for comparison or mixing of pointer types.
  - Except for `void` pointer

```
void *sort_list(void *array, int type);
```

- The `void` pointer is a *generic* or *universal* pointer
  - Such as with `malloc`

```
(int*)malloc(sizeof(int))
```

casting

# void Pointers and Casting

Casting also applies to other types

```
int x = 10;
int y = 9;

printf("result: %f", y/x);
```

output

result: 0.000000

---

```
int x = 10;
int y = 9;

printf("result: %f", (float)y/x);
```

output

result: 0.900000

# Strings and Pointers

How do we assign a length to a string declared as a pointer?

30 physical char space allocated

```
int main() {
   char hello[30] = "Hello World!\n";
   char *hello_ptr =(char*)malloc(sizeof(char)*30);
   hello_ptr = "Hello pointers!\n";
   return EXIT_SUCCESS;
}
```

30 physical char space allocated

# Strings and Pointers

How do we assign a length to a string declared as a pointer?

```
int main() {
   char hello[30] = "Hello World!\n";
   char *hello_ptr =(char*)malloc(sizeof(char)*30);
   hello_ptr = "Hello pointers!\n”;
   return EXIT_SUCCESS;
}
```

memory leak!

# Strings and Pointers

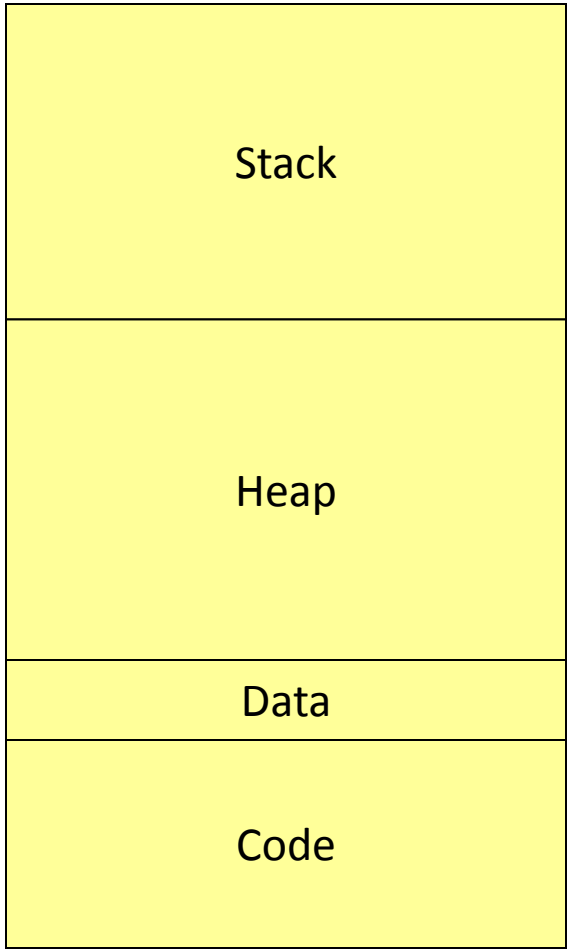
How do we assign a length to a string declared as a pointer?

```
int main() {
   char hello[30] = "Hello World!\n";
   char *hello_ptr =(char*)malloc(sizeof(char)*30);
   strcpy(hello_ptr, "Hello pointers!\n");
   return EXIT_SUCCESS;
}
```

# Dynamic memory allocation

- You should always release memory when it is no longer needed

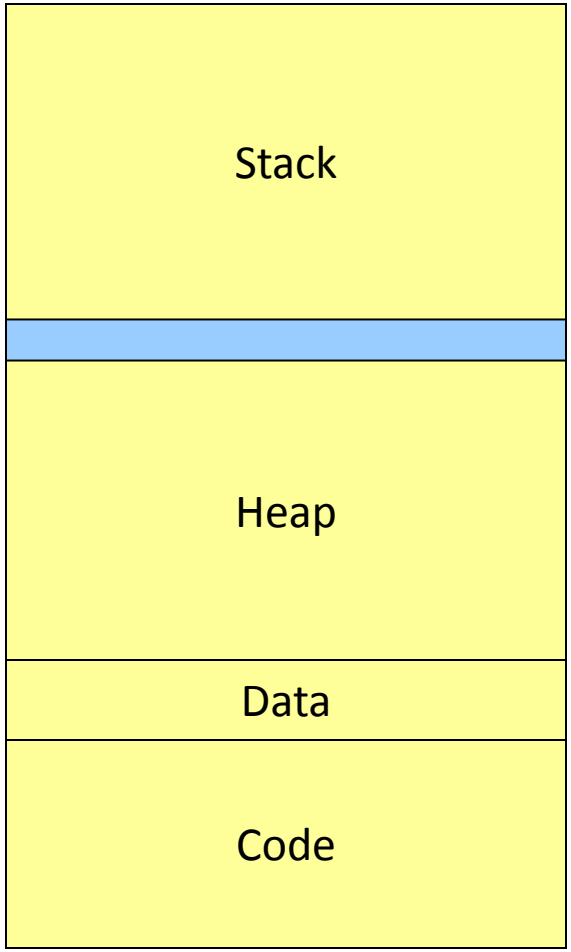
```
p_int = (int*)malloc(sizeof(int))
if(p_int == NULL) {
    return EXIT_FAILURE;
}

free(p_int);
```



# Dynamic memory allocation

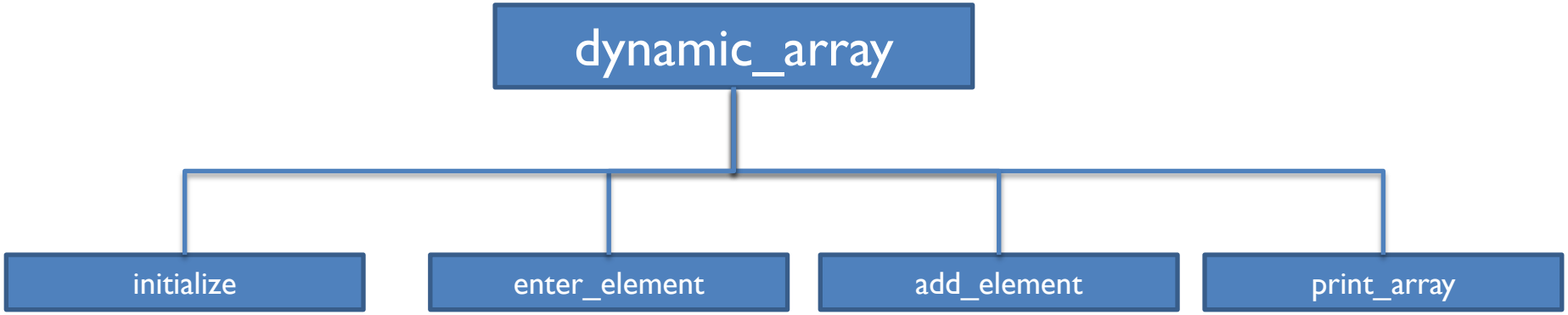
- You should always release memory when it is no longer needed

```
p_int = (int*)malloc(sizeof(int))
if(p_int == NULL) {
    return EXIT_FAILURE;
}

free(p_int);
p_int = NULL;
```



# Dynamic Array Example

We want to write a program where the user can enter an integer that is then put in an array.



# Dynamic Array Example

We want to write a program where the user can enter an integer that is then put in an array.

# Dynamic Array Example

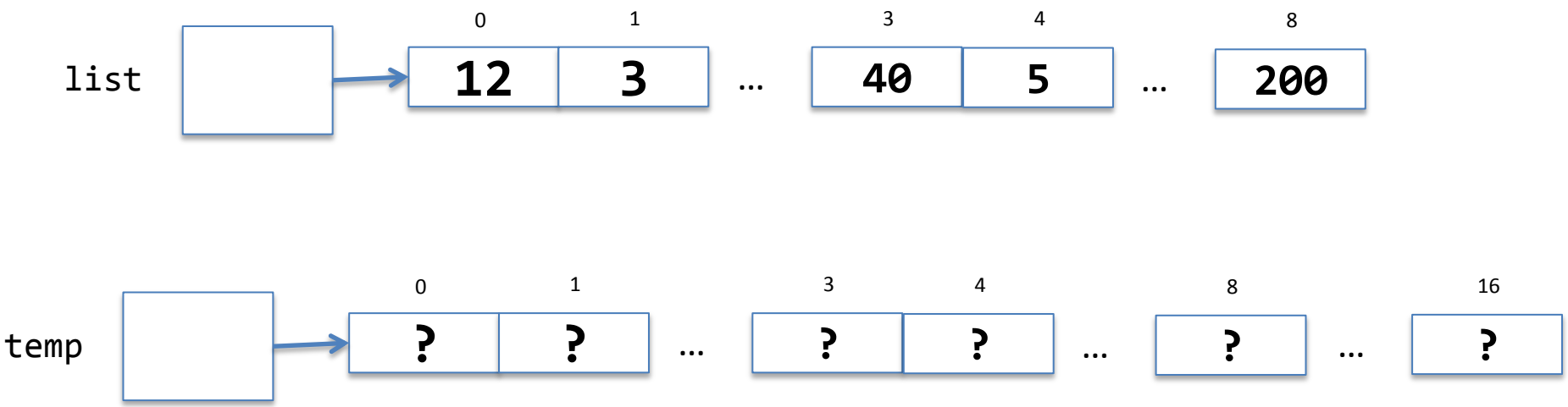
We want to write a program where the user can enter an integer that is then put in an array.

```
#define MAX_LENGTH_LIST 8
```

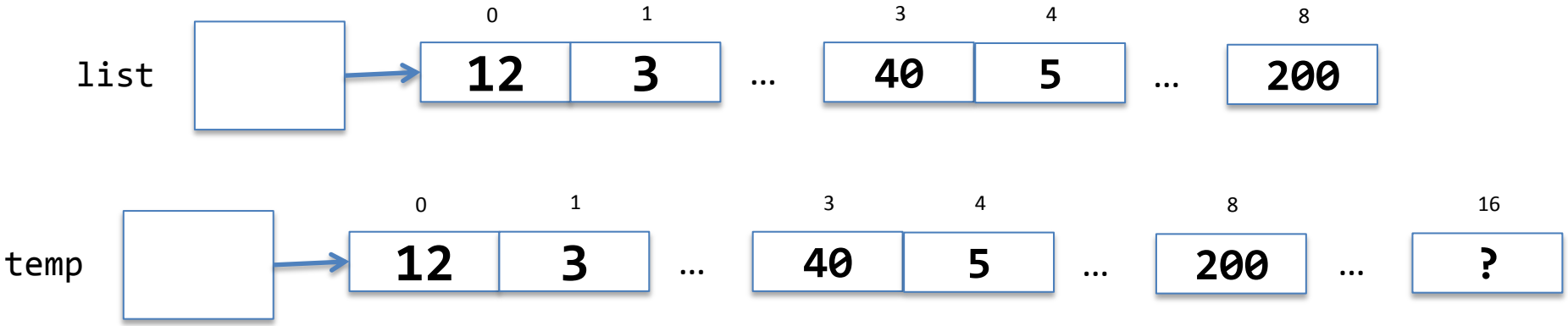

```
int *list;
list = (int*)malloc(MAX_LENGTH_LIST * sizeof(int));

if (list == NULL) {
    return EXIT_FAILURE;
}
```

# Dynamic Array Example

We want to write a program where the user can enter an integer that is then put in an array.

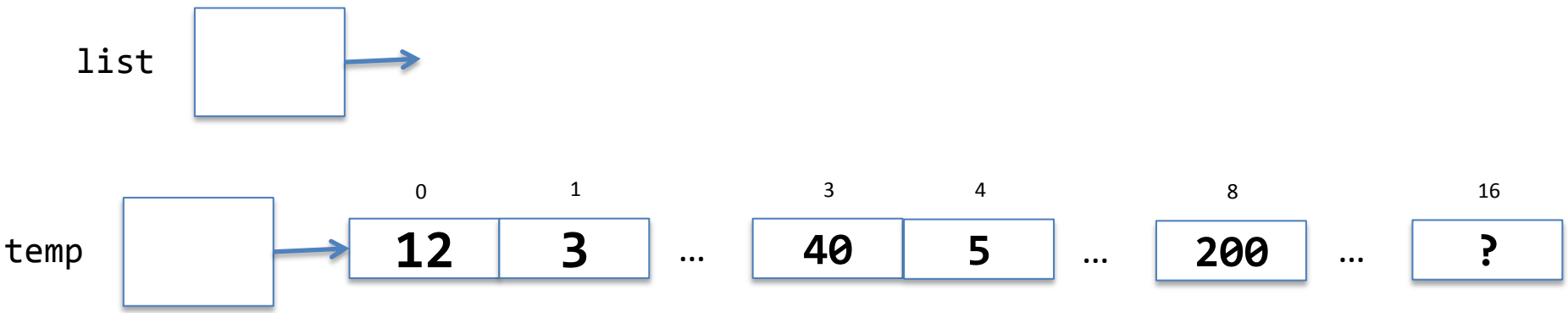


# Dynamic Array Example



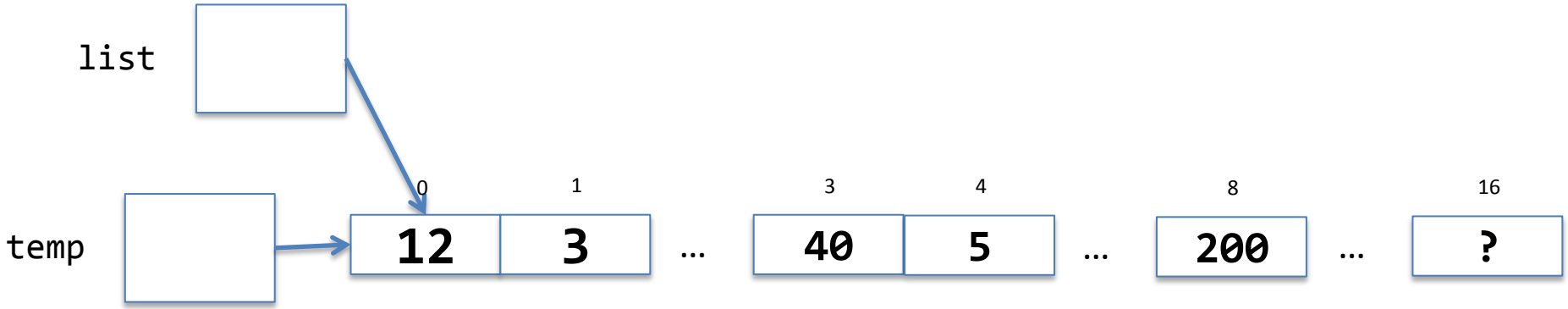

```
if (num_element >= length) {
    int old_length = length;
    length *= 2;
    int *temp_list = malloc(length * sizeof(int));
    if (temp_list == NULL) {
        exit(EXIT_FAILURE);
    }
    for (int i = 0; i < old_length; i++) {
        temp_list[i] = list[i];
    }
    free(list);
    list = temp_list;
}
list[num_element] = num;
num_element++;
```

# Dynamic Array Example




```
if (num_element >= length) {
     int old_length = length;
     length *= 2;
     int *temp_list = malloc(length * sizeof(int));
     if (temp_list == NULL) {
          exit(EXIT_FAILURE);
     }
     for (int i = 0; i < old_length; i++) {
          temp_list[i] = list[i];
     }
     free(list);
     list = temp_list;
}
list[num_element] = num;
num_element++;
```

# Dynamic Array Example




```
if (num_element >= length) {
     int old_length = length;
     length *= 2;
     int *temp= malloc(length * sizeof(int));
     if (temp == NULL) {
          exit(EXIT_FAILURE);
     }
     for (int i = 0; i < old_length; i++) {
          temp[i] = list[i];
     }
     free(list);
     list = temp;
}
list[num_element] = num;
num_element++;
```

# Constants

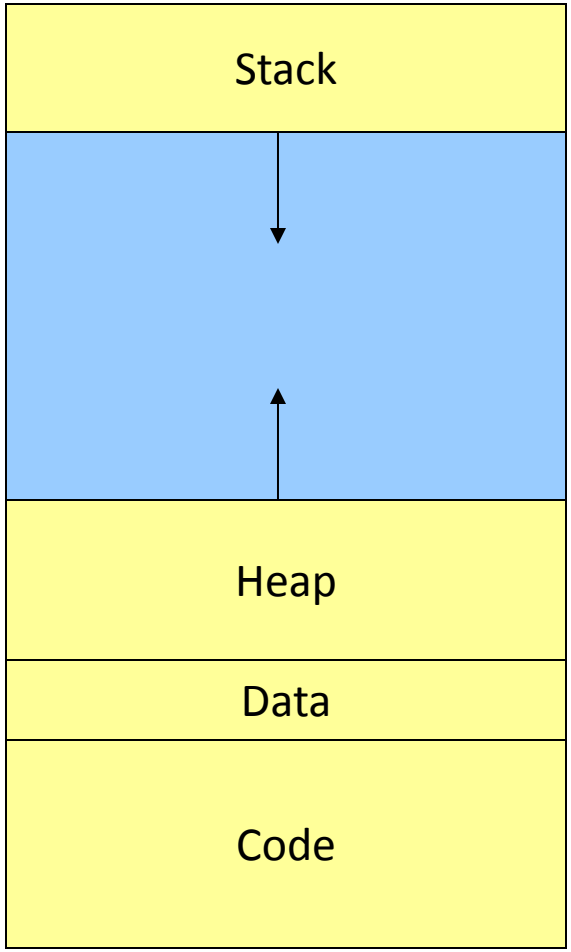
- Constants are data values that cannot be changed during the execution of a program

- There are three ways to code constants in C:
  - Literal Constants: unnamed constant used to specify data
    - 'a' , 5, “Hello World”, 3.25
  - Defined Constants
    - #define SALES_TAX 0.07 // Note there is no semi-colon ;
  - Memory Constants
    - const float PI = 3.14159

# Memory model

Where are those lines located?

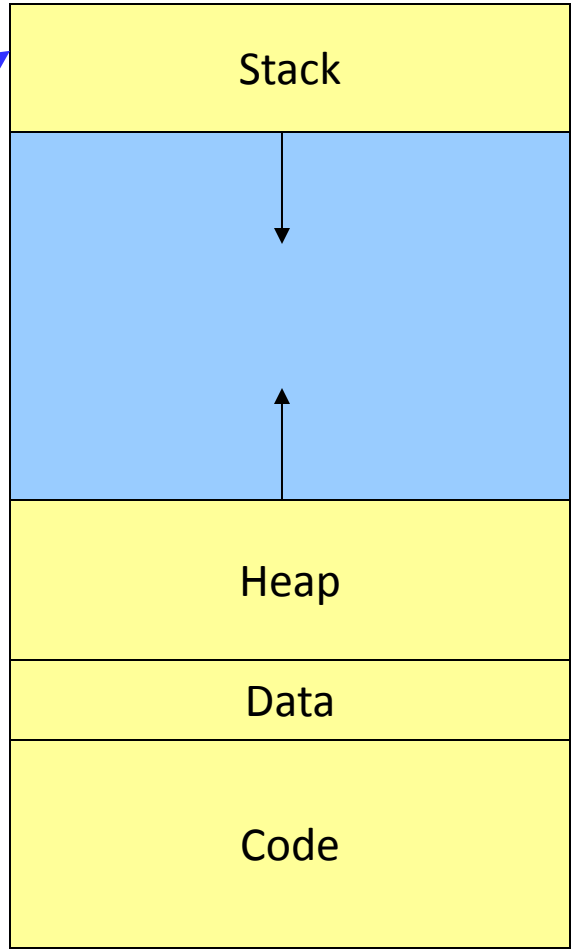
```
void main (void) {
   int *i_ptr;
   …



  i_ptr = (int *)malloc(10 * sizeof(int));
}
```



# Memory model

Where are those lines located?

```
void main (void) {
   int *i_ptr;
   …

  i_ptr = (int *)malloc(10 * sizeof(int));
}
```



# Example

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

int my_global=0;

const int constante = 12;

int main(void) {
  char local_string[]="TEST";
  char *p_char="World";
  static int i_static=100;
  int *heap_var = (int*)malloc(sizeof(int));

  printf("localString                - A stack address: %p\n", local_string);
  printf("pChar                      - A stack address: %p\n", &p_char);
  printf("&heap_var                  - A stack address:  %p\n", &heap_var);
  printf("\"Hello\"                  - A data address:  %p\n", "Hello");
  printf("*pChar points to a literal - A data address:  %p\n", p_char);
  printf("iStatic                    - A data address:  %p\n", &i_static);
  printf("myGlobal                   - A data address:  %p\n", &my_global);
  printf("constante                  - A data address:  %p\n", &constante);
  printf("heap_var                   - A heap address:  %p\n", heap_var);

  return EXIT_SUCCESS;
}
```

# Questions?