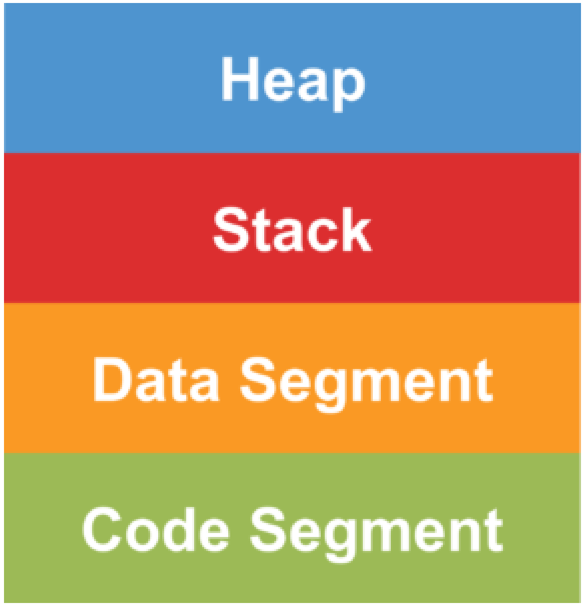


Heap

Stack

Data Segment

Code Segment

Week 6 - Lecture 3

Dynamic Memory Allocation

Edited by: Dr. Wooi Ping Cheah

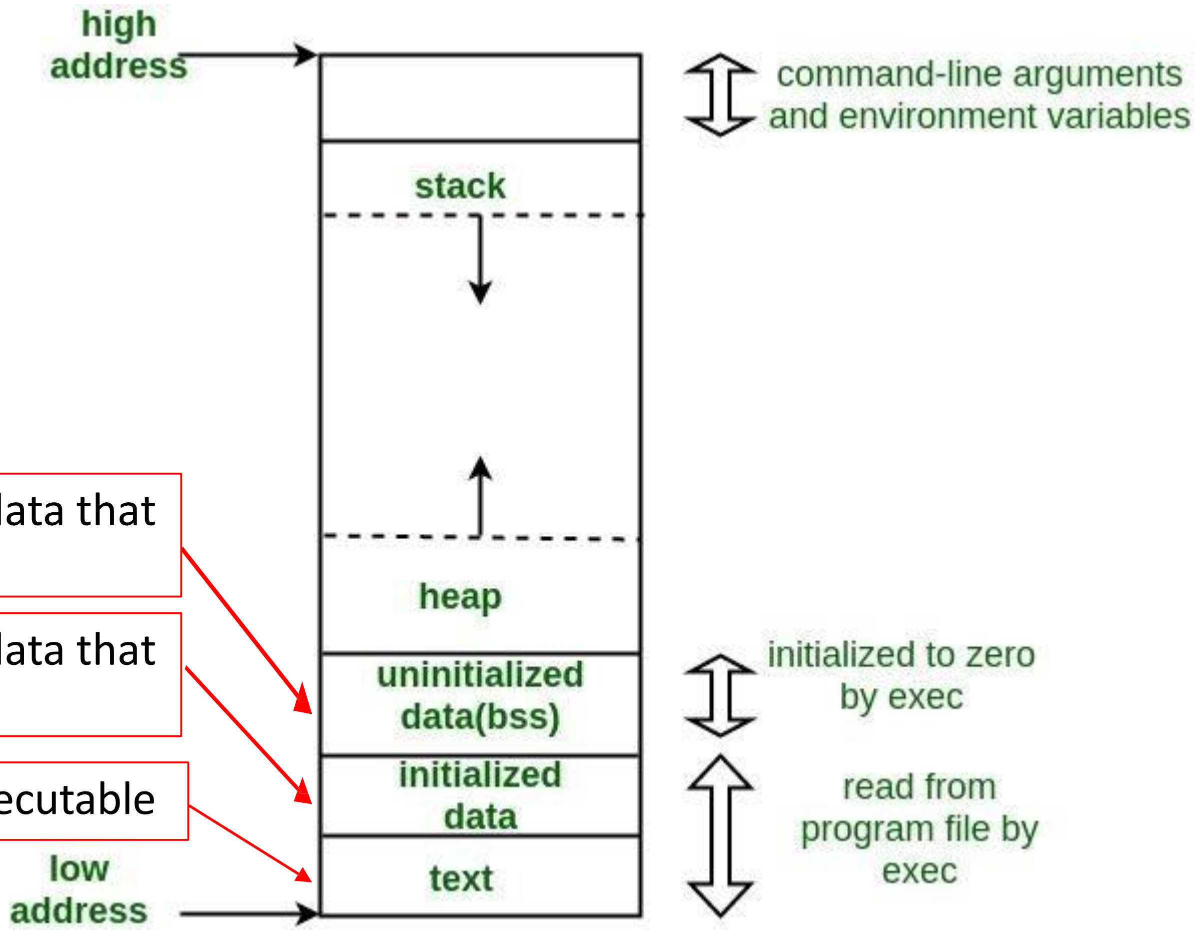
Autumn 2022

Overview

- Heap and Stack
- malloc and free

Memory Layout of C Programs

- Typical layout of a running 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

Global and Static data that are not initialised.

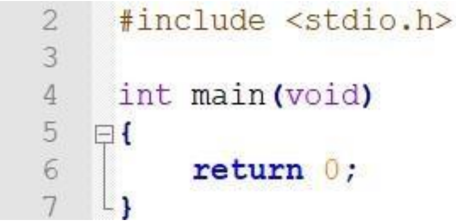
Global and Static data that are initialised.

Code segment, executable

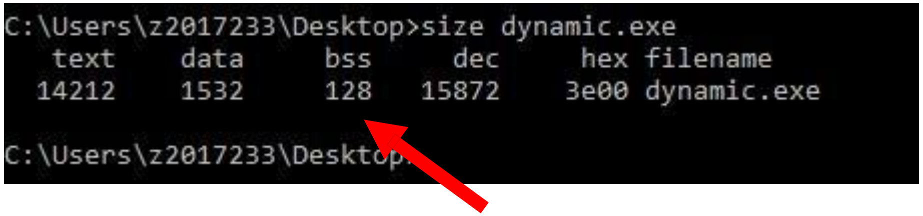
Source: https://www.geeksforgeeks.org/memory-layout-of-c-program/

Memory Layout of C Programs (2)

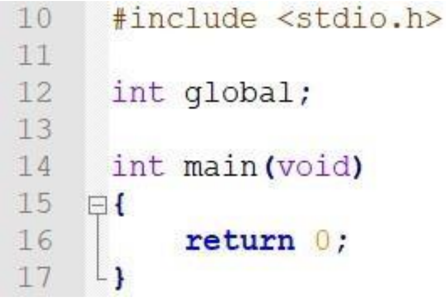
- Note the size of the **uninitialised** data (bss).



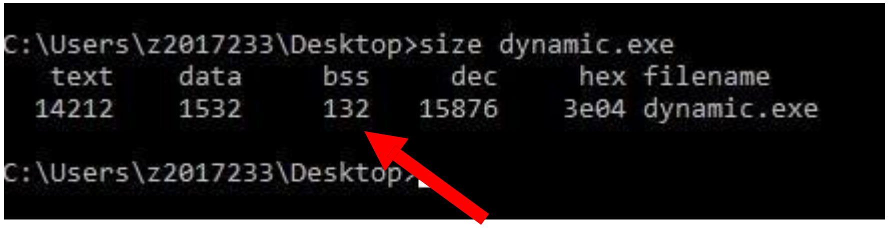
```c
#include <stdio.h>

int main(void)
{
    return 0;
}
```



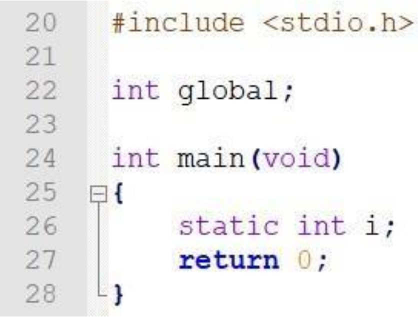
```
C:\Users\z2017233\Desktop>size dynamic.exe
   text    data     bss     dec     hex filename
  14212    1532     128   15872    3e00 dynamic.exe

C:\Users\z2017233\Desktop>
```



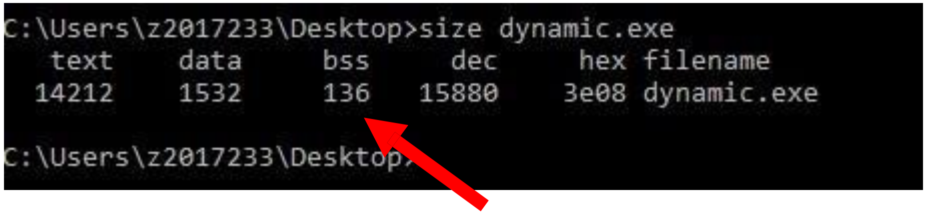
```c
#include <stdio.h>

int global;

int main(void)
{
    return 0;
}
```



```
C:\Users\z2017233\Desktop>size dynamic.exe
   text    data     bss     dec     hex filename
  14212    1532     132   15876    3e04 dynamic.exe

C:\Users\z2017233\Desktop>
```



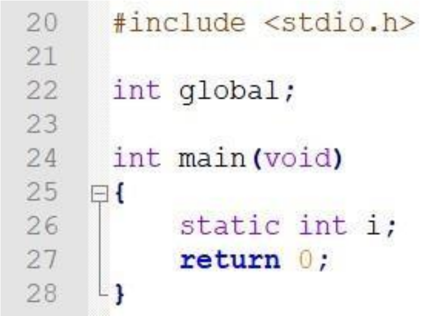
```c
#include <stdio.h>

int global;

int main(void)
{
    static int i;
    return 0;
}
```



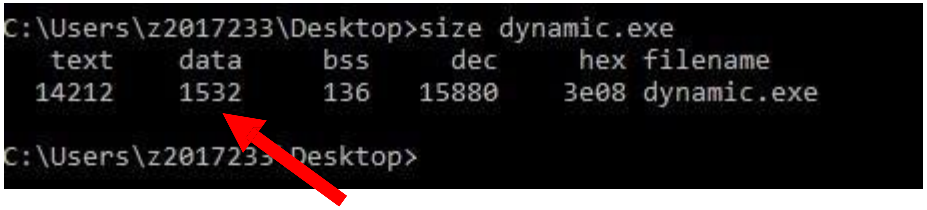
```
C:\Users\z2017233\Desktop>size dynamic.exe
   text    data     bss     dec     hex filename
  14212    1532     136   15880    3e08 dynamic.exe

C:\Users\z2017233\Desktop>
```

Memory Layout of C Programs (3)

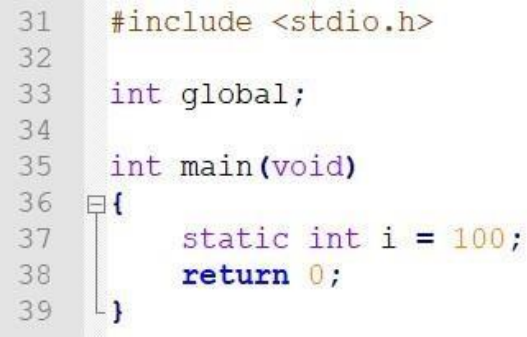
- Note the size of the **initialised** data.



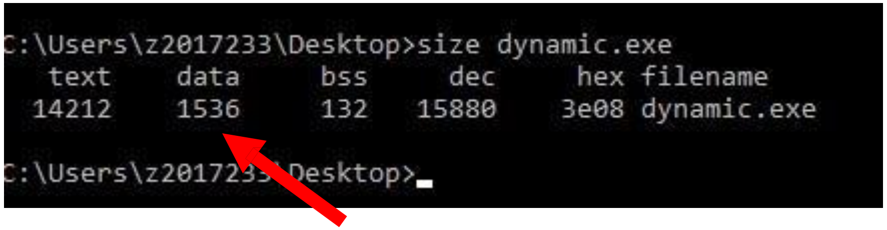
```
#include <stdio.h>

int global;

int main(void)
{
    static int i;
    return 0;
}
```



```
C:\Users\z2017233\Desktop>size dynamic.exe
   text    data     bss     dec     hex filename
  14212    1532     136   15880    3e08 dynamic.exe

C:\Users\z2017233\Desktop>
```



```
#include <stdio.h>

int global;

int main(void)
{
    static int i = 100;
    return 0;
}
```

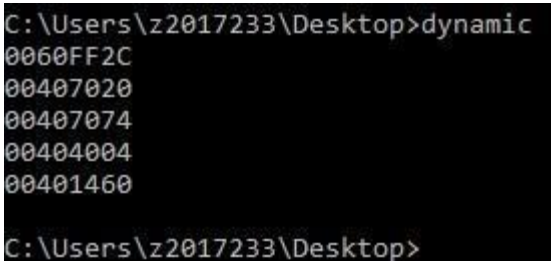


```
C:\Users\z2017233\Desktop>size dynamic.exe
   text    data     bss     dec     hex filename
  14212    1536     132   15880    3e08 dynamic.exe

C:\Users\z2017233\Desktop>
```

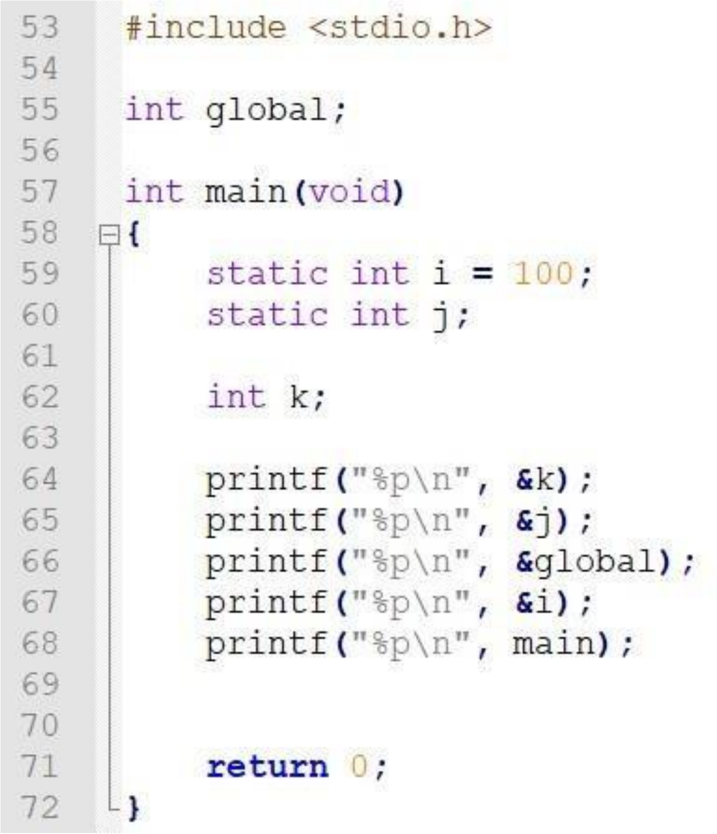
Remember this!?

- The compiler allocates memory (i.e. stack) to store the function’s parameters and the variables when the function is called.
- Once it’s terminated, the memory is automatically deallocated.
- … and **YES**, main is a function!!

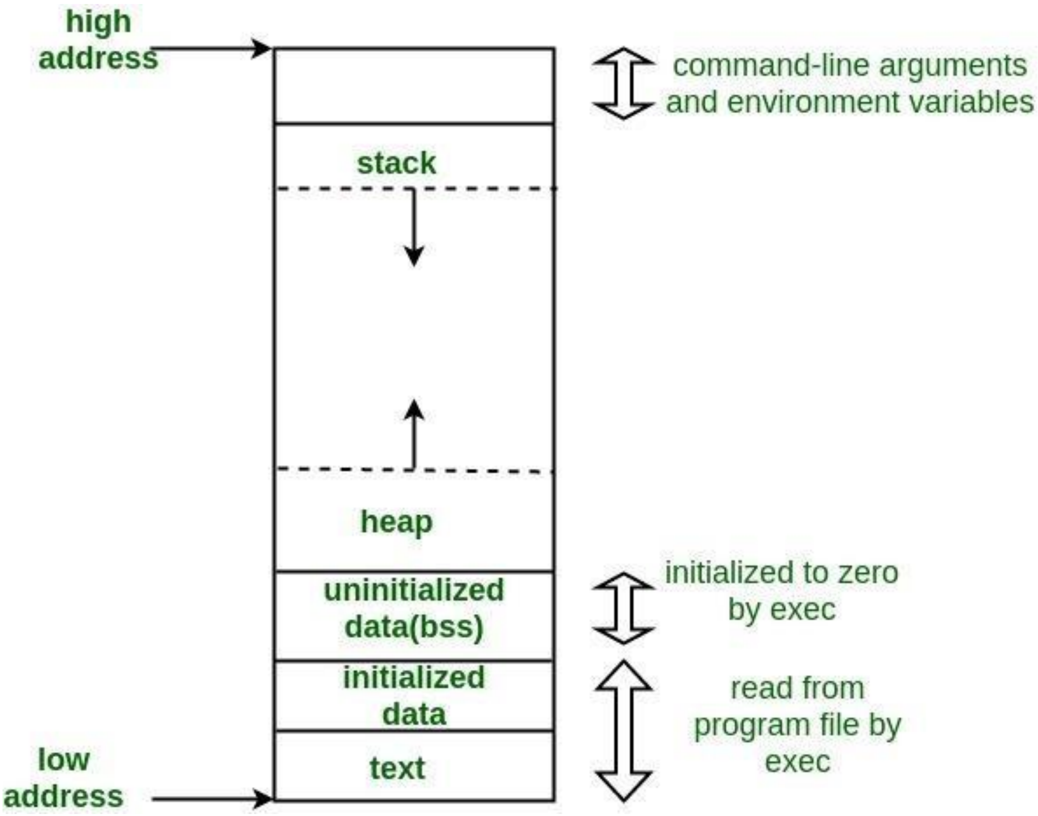
Memory Layout of C Programs (5)



```
C:\Users\z2017233\Desktop>dynamic
0060FF2C
00407020
00407074
00404004
00401460

C:\Users\z2017233\Desktop>
```



```c
#include <stdio.h>

int global;

int main(void)
{
    static int i = 100;
    static int j;

    int k;

    printf("%p\n", &k);
    printf("%p\n", &j);
    printf("%p\n", &global);
    printf("%p\n", &i);
    printf("%p\n", main);


    return 0;
}
```



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

Source: https://www.geeksforgeeks.org/memory-layout-of-c-program/

Overview

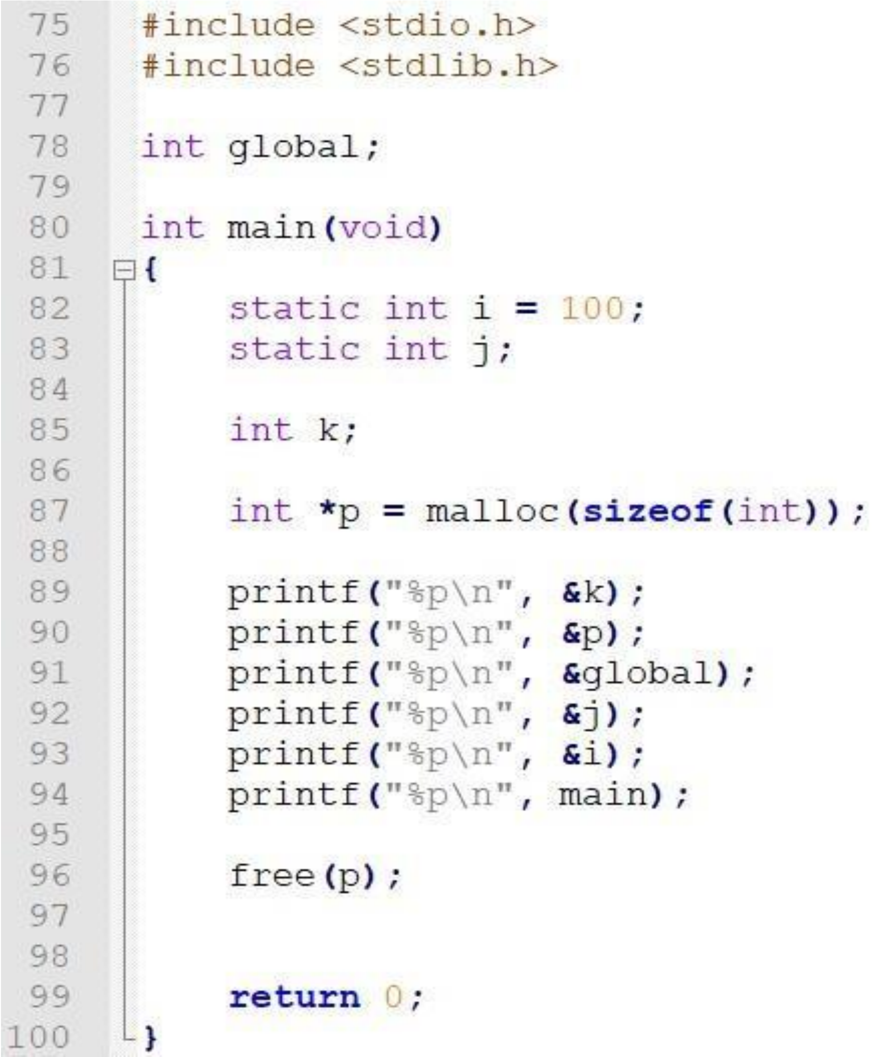
- Heap and Stack
- **malloc and free**

memory allocation

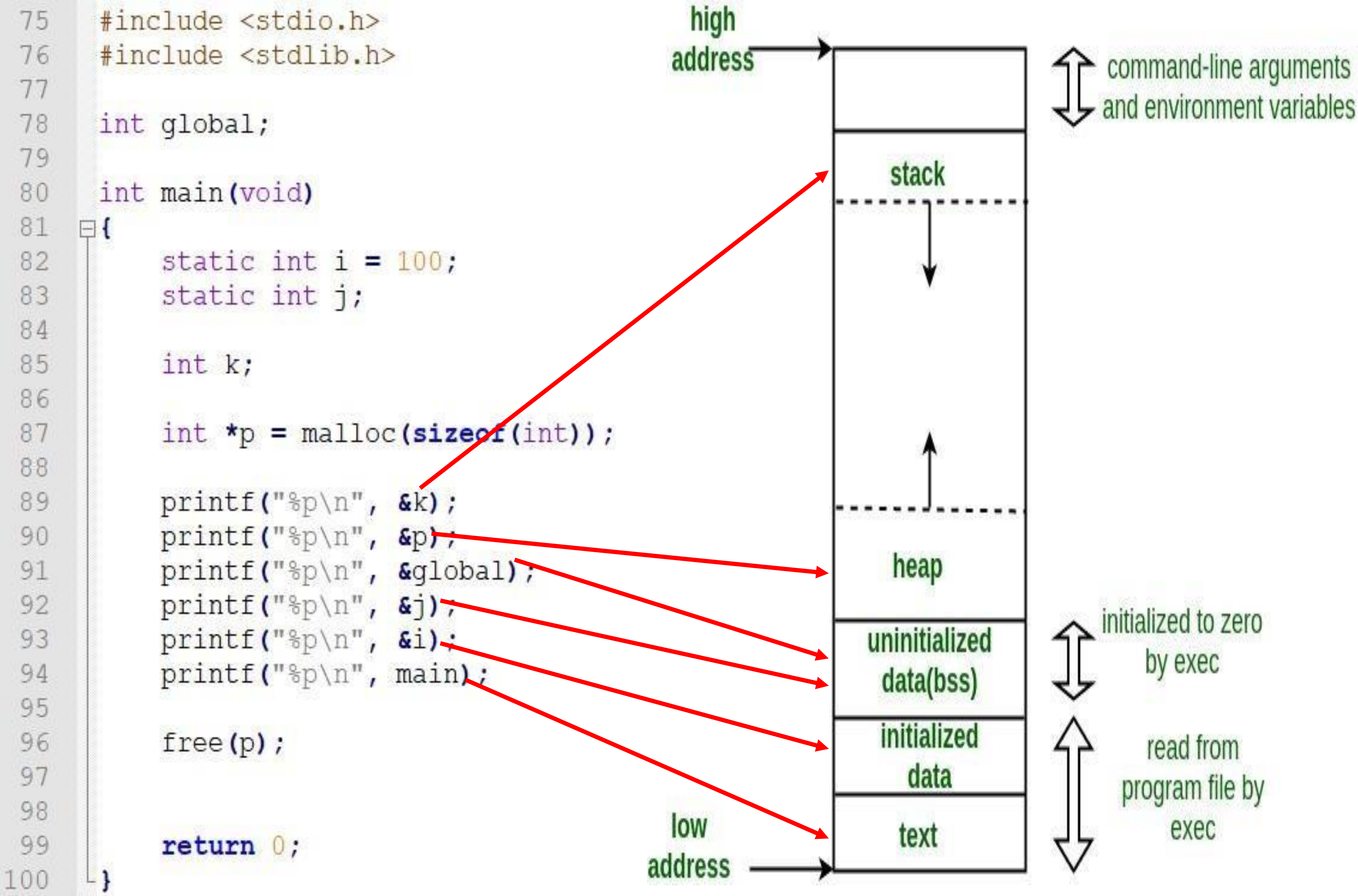
Situation 1: Large amount of data
Situation 2: 用户的交互

Heap (Unlike Stack …)

- The segment where dynamic memory allocation usually takes place.
- Memory doesn’t get deallocated at the end of a function call.
- Manage by the programmer using e.g. malloc, and free.



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

int global;

int main(void)
{
    static int i = 100;
    static int j;

    int k;

    int *p = malloc(sizeof(int));

    printf("%p\n", &k);
    printf("%p\n", &p);
    printf("%p\n", &global);
    printf("%p\n", &j);
    printf("%p\n", &i);
    printf("%p\n", main);

    free(p);


    return 0;
}
```



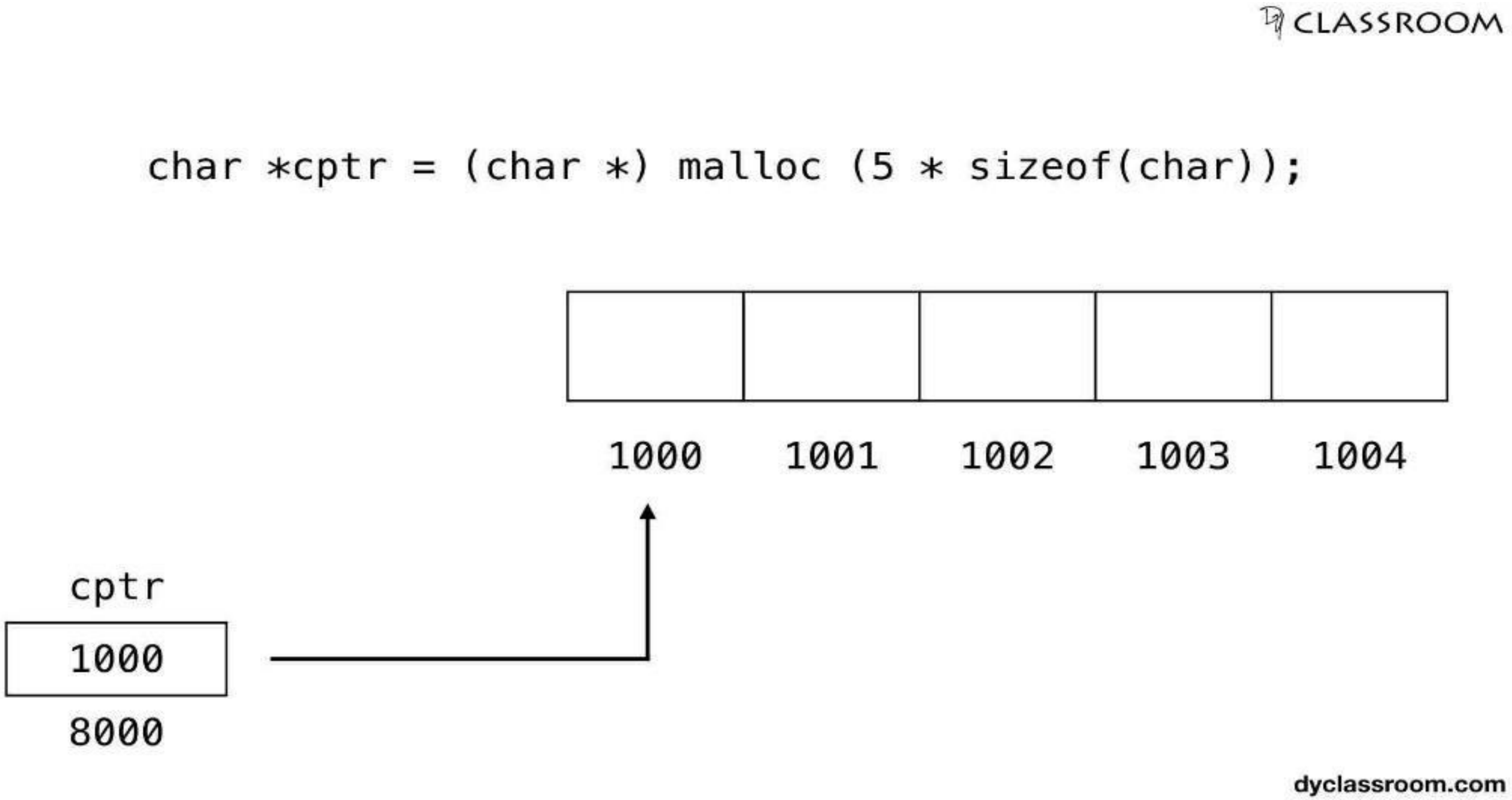
```c
#include <stdio.h>
#include <stdlib.h>

int global;

int main(void)
{
    static int i = 100;
    static int j;

    int k;

    int *p = malloc(sizeof(int));

    printf("%p\n", &k);
    printf("%p\n", &p);
    printf("%p\n", &global);
    printf("%p\n", &j);
    printf("%p\n", &i);
    printf("%p\n", main);

    free(p);


    return 0;
}
```

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

def: the segment of memory defined and managed by the program

Dynamic Memory Allocation

- Create dynamic data structures that can change size e.g., lists, trees, graphs.



```
char *cptr = (char *) malloc (5 * sizeof(char));
```

Source: https://www.dyclassroom.com/c/c-dynamic-memory-allocation-malloc-function

malloc

- Returns a pointer to a newly allocated block of memory in the heap.
- Size is determined in bytes.

- Use

```
int *p = malloc(sizeof(int));
char *q = malloc(sizeof(char));
```

free

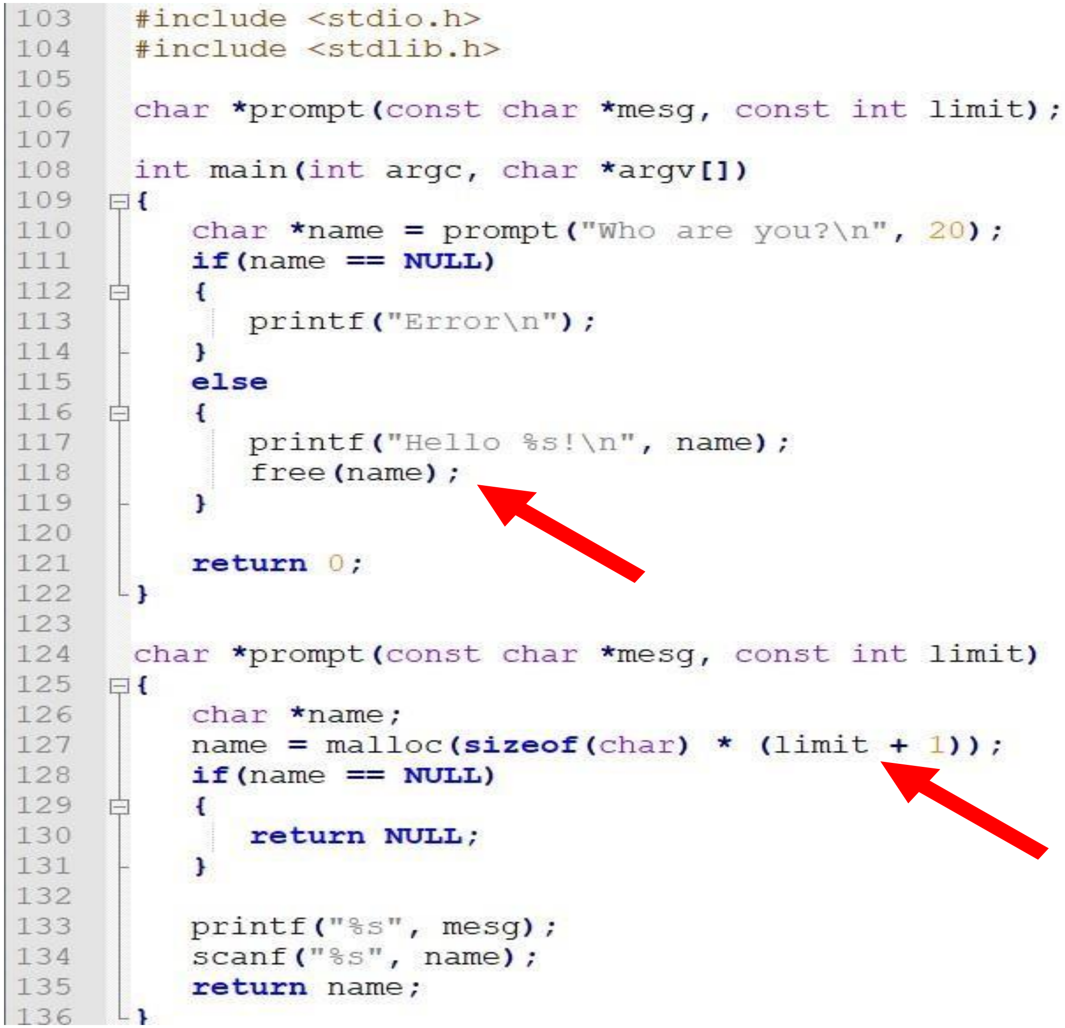
- To deallocate the block of memory after you have finished using.
- Trying to free memory not allocated by malloc is an error.
- Trying to free the same memory multiple times is an error.
- free(p);

free (2)

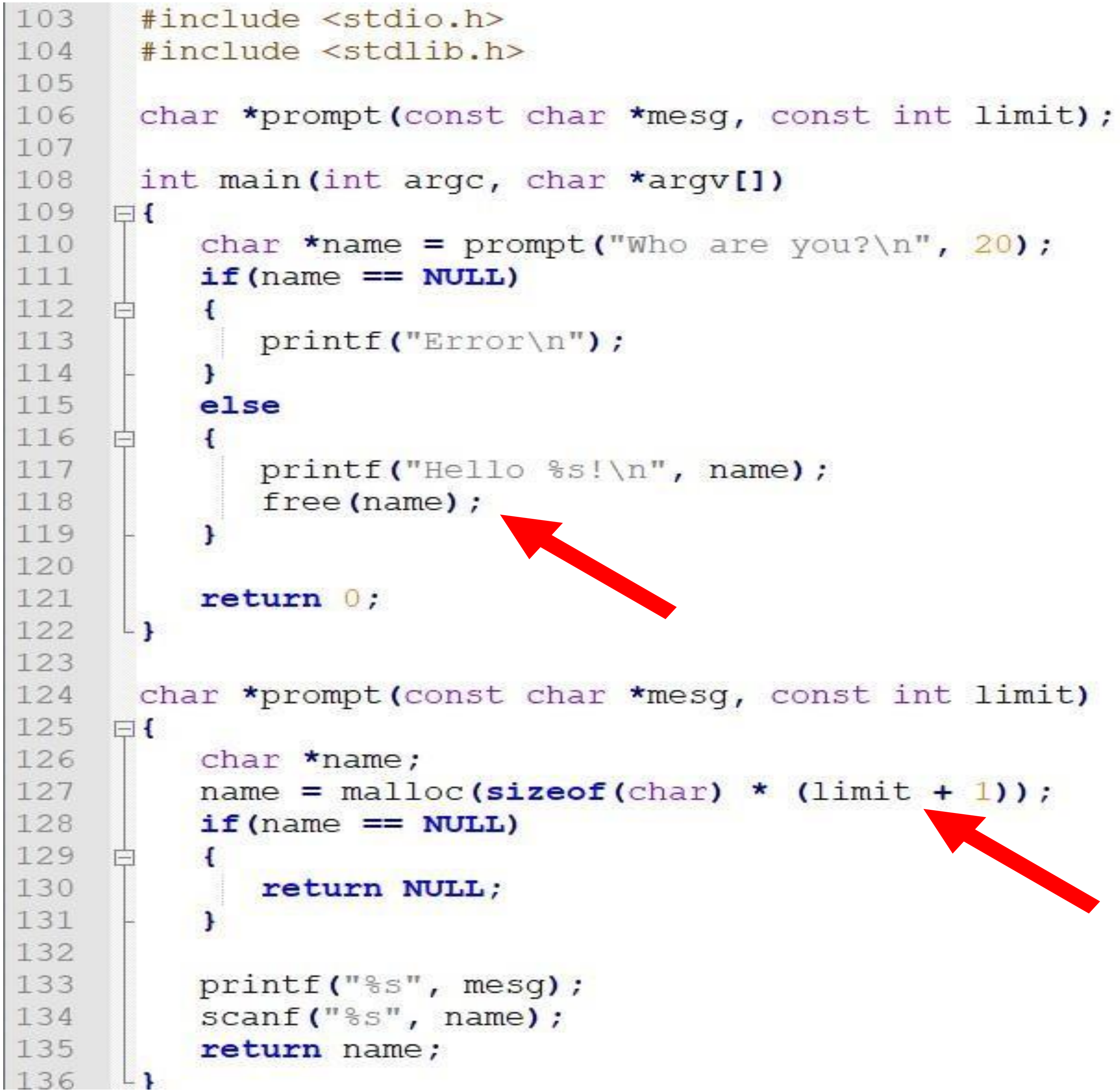
- If forget to free memory which no longer required, it can make your program use more and more memory the longer it is running.
- When the program exits, the OS will reclaim all of the memory, even if it has not been freed.

Example: Reusable Prompt

- To print a prompt then read in a string.



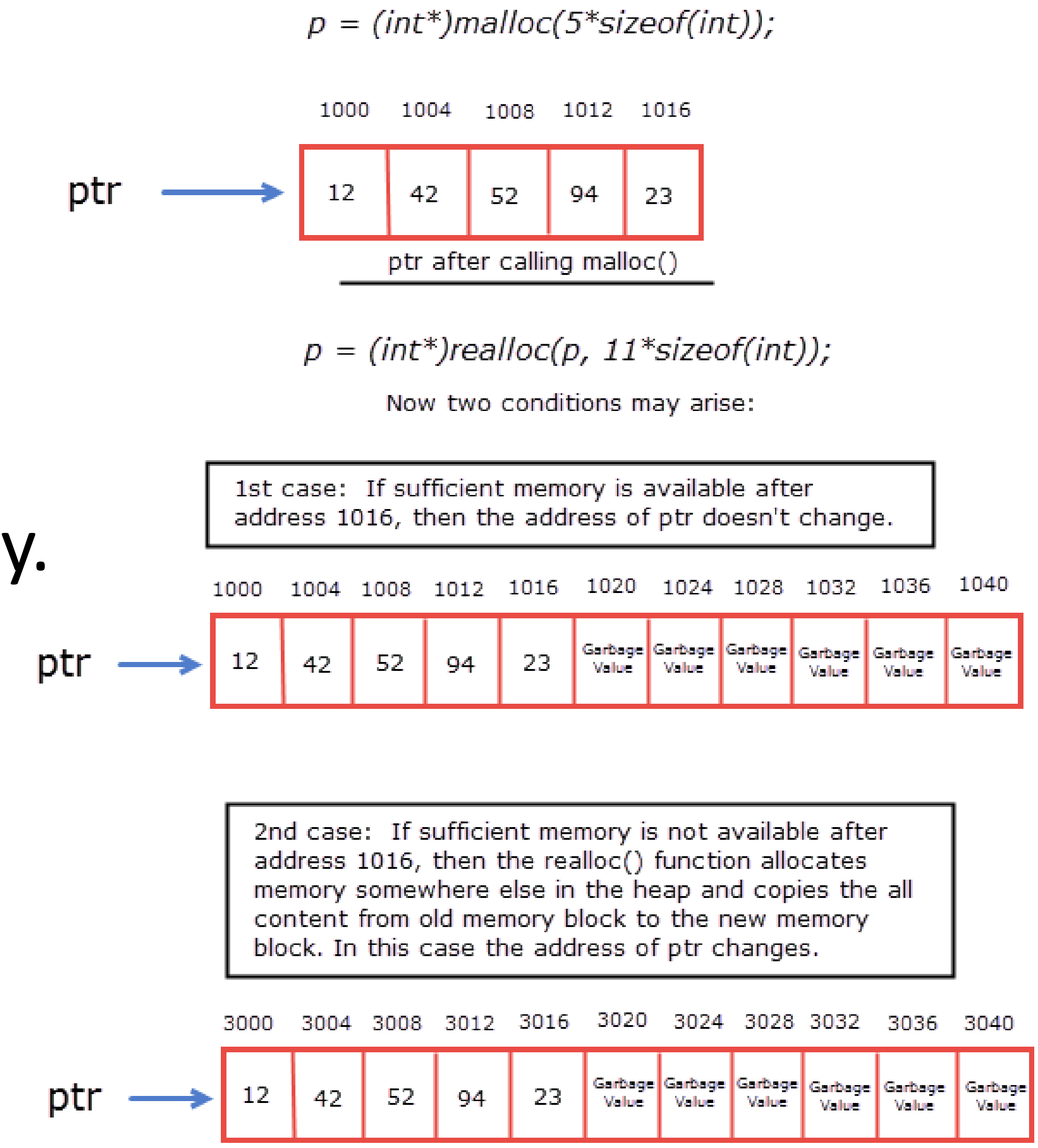
```c
#include <stdio.h>
#include <stdlib.h>

char *prompt(const char *mesg, const int limit);

int main(int argc, char *argv[])
{
    char *name = prompt("Who are you?\n", 20);
    if(name == NULL)
    {
        printf("Error\n");
    }
    else
    {
        printf("Hello %s!\n", name);
        free(name);
    }

    return 0;
}

char *prompt(const char *mesg, const int limit)
{
    char *name;
    name = malloc(sizeof(char) * (limit + 1));
    if(name == NULL)
    {
        return NULL;
    }

    printf("%s", mesg);
    scanf("%s", name);
    return name;
}
```



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

char *prompt(const char *mesg, const int limit);

int main(int argc, char *argv[])
{
    char *name = prompt("Who are you?\n", 20);
    if(name == NULL)
    {
        printf("Error\n");
    }
    else
    {
        printf("Hello %s!\n", name);
        free(name);
    }

    return 0;
}

char *prompt(const char *mesg, const int limit)
{
    char *name;
    name = malloc(sizeof(char) * (limit + 1));
    if(name == NULL)
    {
        return NULL;
    }

    printf("%s", mesg);
    scanf("%s", name);
    return name;
}
```

realloc

- To resize the previously allocated memory.



```
p = (int*)malloc(5*sizeof(int));
```

1000	1004	1008	1012	1016
12	42	52	94	23

ptr after calling malloc()

```
p = (int*)realloc(p, 11*sizeof(int));
```

Now two conditions may arise:

1st case: If sufficient memory is available after address 1016, then the address of ptr doesn't change.

1000	1004	1008	1012	1016	1020	1024	1028	1032	1036	1040
12	42	52	94	23	Garbage Value	Garbage Value	Garbage Value	Garbage Value	Garbage Value	Garbage Value

2nd case: If sufficient memory is not available after address 1016, then the realloc() function allocates memory somewhere else in the heap and copies the all content from old memory block to the new memory block. In this case the address of ptr changes.

3000	3004	3008	3012	3016	3020	3024	3028	3032	3036	3040
12	42	52	94	23	Garbage Value	Garbage Value	Garbage Value	Garbage Value	Garbage Value	Garbage Value

Source: https://overiq.com/c-programming-101/the-realloc-function-in-c/

TheCguru.com

Example: realloc

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

int main () {
   char *str;

   str = (char *) malloc(sizeof(char)*15);
   strcpy(str, "tutorialspoint");
   printf("String = %s,  Address = %p\n", str, str);

   str = (char *) realloc(str, 25*sizeof(char));
   strcat(str, ".com");
   printf("String = %s,  Address = %p\n", str, str);

   free(str);
   return(0);
}
```

Output:

```
String = tutorialspoint, Address = 0xd204010
String = tutorialspoint.com, Address = 0xd204010
```

Summary

- Heap and Stack
- malloc and free