

The C Language :

Dynamic Memory

Allocation

David Bouchet

david.bouchet.epita@gmail.com

Static Allocations

- The memory size is known at compile time.
- Some data can be stored in the executable file.
- The allocation is done when the program is loaded into memory.
- The allocated memory cannot be freed.
- The data has the same lifetime as the running program.
- Usually used for global variables and literals.

Dynamic Allocations

- The memory size can be known or unknown at compile time.
- The allocation is done at runtime.
- The memory must be freed.

Stack and heap allocations are two different mechanisms of dynamic allocations.

Stack Allocations

- Fast allocation.
- Safe.
- No memory fragmentation.
- Memory is allocated and freed automatically.
- Allocated memory is limited.
- Variables cannot be resized.
- Local access.

Heap Allocations

- Slow allocation.
- Unsafe.
- Memory fragmentation.
- Memory is allocated and freed manually.
- Allocated memory is not limited.
- Variables can be resized.
- Global access.

Stack Allocation – Example

```
int main()
{
    int a = 10; // stack allocation
    int b = 20; // stack allocation
    int s;      // stack allocation

    // a: passed in (copied into x)
    // b: passed in (copied into y)
    s = sum(a, b);

    printf("%i + %i = %i\n", a, b, s);

    return 0;

    // s: freed
    // b: freed
    // a: freed
}
```

```
int sum(int x, int y)
{
    // x: stack allocation
    // y: stack allocation

    // z: stack allocation
    int z = x + y;

    // z: returned (copied into s)
    return z;

    // z: freed
    // y: freed
    // x: freed
}
```

Heap Allocation – Functions

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

Allocation

```
void *malloc(size_t size);  
void *calloc(size_t nmemb, size_t size);  
void *realloc(void *ptr, size_t size);
```

Deallocation

```
void free(void *ptr);
```

 See also the [man pages](#)


malloc() and *free()*

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

int main()
{
    int *p = malloc(sizeof(int));
    if (p == NULL)
        errx(1, "Not enough memory!");

    *p = 72;
    printf("*p = %i\n", *p);

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



*p = 72

Dynamic Arrays - *malloc()*

```
size_t size = 35;

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

if (p == NULL)
    errx(1, "Not enough memory!");

for (size_t i = 0; i < size; i++)
    *(p + i) = 0;

free(p);
```

malloc(): the memory is not initialized.

Dynamic Arrays - `calloc()`

```
size_t size = 35;

int *p = calloc(size, sizeof(int));

if (p == NULL)
    errx(1, "Not enough memory!");

free(p);
```

`calloc()`: the memory is set to zero.

Dynamic Strings – *malloc()* (1)

```
char *concat(char *str1, char *str2)
{
    size_t size = strlen(str1) + strlen(str2) + 1;


    char *str = malloc(size * sizeof(char));
    if (str == NULL)
        errx(1, "Not enough memory!");

    char *p = str;
    while (*str1 != 0)
        *(p++) = *(str1++);
    while (*str2 != 0)
        *(p++) = *(str2++);
    *p = 0;

    return str;
}
```

Dynamic Strings – *malloc()* (2)

```
char s1[] = "Hello";  
char s2[] = "World!";  
  
char *w = concat(s1, " ");  
char *x = concat(w, s2);  
char *y = concat(x, "\n");  
char *z = concat(y, "Good bye!");  
  
printf("%s\n", z);  
  
free(w);  
free(x);  
free(y);  
free(z);
```



Hello World!
Good bye!

Dynamic Strings – *realloc()* (1)

```
char *append1(char *str1, char *str2)
{
    size_t size1 = strlen(str1);
    size_t size = size1 + strlen(str2) + 1;

    char *str = realloc(str1, size * sizeof(char));
    if (str == NULL)
        errx(1, "Not enough memory!");

    char *p = str + size1;
    while (*str2 != 0)
        *(p++) = *(str2++);

    *p = 0;

    return str;
}
```

Dynamic Strings – *realloc()* (2)

```
char s1[] = "Hello";  
char s2[] = "World!";  
  
char *x = concat(s1, " ");  
x = append1(x, s2);  
x = append1(x, "\n");  
x = append1(x, "Good bye!");  
  
printf("%s\n", x);  
  
free(x);
```



```
Hello World!  
Good bye!
```

Dynamic Strings – Pointers to Pointers (1)

```
void append2(char **str1, char *str2)
{
    size_t size1 = strlen(*str1);
    size_t size = size1 + strlen(str2) + 1;

    char *str = realloc(*str1, size * sizeof(char));
    if (str == NULL)
        errx(1, "Not enough memory!");

    char *p = str + size1;
    while (*str2 != 0)
        *(p++) = *(str2++);
    *p = 0;

    *str1 = str;
}
```

Dynamic Strings – Pointers to Pointers (2)

```
char s1[] = "Hello";  
char s2[] = "World!";  
  
char *x = concat(s1, " ");  
append2(&x, s2);  
append2(&x, "\n");  
append2(&x, "Good bye!");  
  
printf("%s\n", x);  
  
free(x);
```



```
Hello World!  
Good bye!
```