Dynamic memory allocation

#cisfun

Automatic allocation

When you declare variables or when you use strings within double quotes, the program is taking care of all the memory allocation. You do not have to think about it.

Example

```
int fun(int a)
{
    char s[] = "Hello World\n";
    int ar[3];
    int b;

    [...]
}
```

Address	40	41	42	43	44	45	46	47	48	49	50	51	52	53	54	55	56	57	58	59
Variable	"Hello World\n"														a					
Value	H	e	l	l	o		W	o	r	l	d	\n	\0	?	?				?	?

Address	60	61	62	63	64	65	66	67	68	69	70	71	72	73	74	75	76	77	78	79
Variable	s														b					
Value	H	e	l	l	o		W	o	r	l	d	\n	\0	?	?				?	?

Address	80	81	82	83	84	85	86	87	88	89	90	91	92	93	94	95	96	97	98	99
Variable	ar																			
Value	?				?				?				?	?	?	?	?	?	?	?

Dynamic allocation

So far we have used variables, arrays with fixed size. But what happens if you do not know the size of the array you have to declare and / or if this size depends on another variable?

Note: remember, you can declare arrays only with a constant.

```
type variable[constant]; /* works */
```

```
int n;
n = 10;
type variable[n]; /* does not work */
```

Example

```c
int main(void)
{
	int n;
	n = 5;
	char ar[n]; /* does not compile */
	return (0);
}
```

malloc

#include <stdlib.h>

void *malloc(size_t size);

The malloc() function allocates size bytes and returns a pointer to the allocated memory

The memory is not initialized

Example

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

int main(void)
{
	int n;
	char *ar;

	n = 5;
	ar = malloc(n * sizeof(char));
	ar[0] = 'C';
	ar[1] = 'o';
	ar[2] = 'o';
	ar[3] = 'l';
	ar[4] = '\0';
	printf("%s\n", ar); /* prints Cool\n */
	return (0);
}
```

Memory state before n = 5;

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

int main(void)
{
	int n;
	char *ar;

	n = 5;
	ar = malloc(n * sizeof(char));
	ar[0] = 'C';
	ar[1] = 'o';
	ar[2] = 'o';
	ar[3] = 'l';
	ar[4] = '\0';
	printf("%s\n", ar); /* prints Cool\n */
	return (0);
```

Address	40	41	42	43	44	45	46	47	48	49	50	51	52	53	54	55	56	57	58	59
Variable	"%s\n"																			
Value	%	s	\n	\0	?	?	?	?	?	?	?	?	?	?	?	?	?	?	?	?
Address	60	61	62	63	64	65	66	67	68	69	70	71	72	73	74	75	76	77	78	79
Variable	n				ar															
Value	?				?								?	?	?	?	?	?	?	?

Memory state after the call to malloc

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

int main(void)
{
      int n;
      char *ar;

      n = 5;
      ar = malloc(n * sizeof(char));
      ';
      ';
      ';
      ';
      \0';
      s\n", ar); /* prints Cool\n */
      );
```

Address	40	41	42	43	44	45	46	47	48	49	50	51
Variable	"%s\n"											
Value	%	s	\n	\0	?	?	?	?	?	?	?	?

Address	60	61	62	63	64	65	66	67	68	69	70	71
Variable	n				ar							
Value	5				80							

Address	80	81	82	83	84	85	86	87	88	89	90	91
Variable												
Value	?	?	?	?	?	?	?	?	?	?	?	?

Memory state before the call to printf

Address	40	41	42	43	44	45	46	47	48	49	50	51
Variable	"%s\n"											
Value	%	s	\n	\0	?	?	?	?	?	?	?	?
Address	60	61	62	63	64	65	66	67	68	69	70	71
Variable	n				ar							
Value	5				80							
Address	80	81	82	83	84	85	86	87	88	89	90	91
Variable												
Value	C	o	o	l	\0	?	?	?	?	?	?	?

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

int main(void)
{
	int n;
	char *ar;

	n = 5;
	ar = malloc(n * sizeof(char));
	ar[0] = 'C';
	ar[1] = 'o';
	ar[2] = 'o';
	ar[3] = 'l';
	ar[4] = '\0';
	printf("%s\n", ar); /* prints Cool\n */
	return (0);
}
```

Exercise

What does this program do?

Represent the memory state, step by step.

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

void print_int_array(int *a, int size)
{
	int i;

	i = 0;
	while (i < size)
	{
		printf("%d\n", a[i]);
		i++;
	}
}

int main(int ac, char **av)
{
	int *a;
	int asize;
	int i;

	if (ac < 2)
	{
		printf("Please give me at least one number\n");
		printf("Usage: %s number [NUMBER]\n", av[0]);
		return (1);
	}
	asize = ac - 1;
	a = malloc(asize * sizeof(int));
	i = 0;
	while (i < asize)
	{
		a[i] = atoi(av[i + 1]);
		i++;
	}
	print_int_array(a, asize);
	return (0);
}
```

Address		1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19
Variable																				
Value		?	?	?	?	?	?	?	?	?	?	?	?	?	?	?	?	?	?	?

Address	20	21	22	23	24	25	26	27	28	29	30	31	32	33	34	35	36	37	38	39
Variable																				
Value	?	?	?	?	?	?	?	?	?	?	?	?	?	?	?	?	?	?	?	?

Address	40	41	42	43	44	45	46	47	48	49	50	51	52	53	54	55	56	57	58	59
Variable																				
Value	?	?	?	?	?	?	?	?	?	?	?	?	?	?	?	?	?	?	?	?

Address	60	61	62	63	64	65	66	67	68	69	70	71	72	73	74	75	76	77	78	79
Variable																				
Value	?	?	?	?	?	?	?	?	?	?	?	?	?	?	?	?	?	?	?	?

Address	80	81	82	83	84	85	86	87	88	89	90	91	92	93	94	95	96	97	98	99
Variable																				
Value	?	?	?	?	?	?	?	?	?	?	?	?	?	?	?	?	?	?	?	?

Address	100	101	102	103	104	105	106	107	108	109	110	111	112	113	114	115	116	117	118	119
Variable																				
Value	?	?	?	?	?	?	?	?	?	?	?	?	?	?	?	?	?	?	?	?

Address	120	121	122	123	124	125	126	127	128	129	130	131	132	133	134	135	136	137	138	139
Variable																				
Value	?	?	?	?	?	?	?	?	?	?	?	?	?	?	?	?	?	?	?	?

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

void print_int_array(int *a, int size)
{
	int i;

	i = 0;
	while (i < size)
	{
		printf("%d\n", a[i]);
		i++;
	}
}

int main(int ac, char **av)
{
	int *a;
	int asize;
	int i;

	if (ac < 2)
	{
		printf("Please give me at least one number\n");
		printf("Usage: %s number [NUMBER]\n", av[0]);
		return (1);
	}
	asize = ac - 1;
	a = malloc(asize * sizeof(int));
	i = 0;
	while (i < asize)
	{
		a[i] = atoi(av[i + 1]);
		i++;
	}
	print_int_array(a, asize);
	return (0);
}
```

Exercise

What does this program do?

Represent the memory state, step by step.

```c
char *create_string(int len, char c)
{
	char *s;
	int i;

	s = malloc((len + 1) * sizeof(char));
	if (s == NULL)
	{
		return (NULL);
	}
	i = 0;
	while (i < len)
	{
		s[i] = c;
		i++
	}
	s[i] = '\0';
	return (s);
}

int main(void)
{
	char *s;

	s = create_string(5, 'H');
	[...]
```

Address		1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19
Variable																				
Value		?	?	?	?	?	?	?	?	?	?	?	?	?	?	?	?	?	?	?

Address	20	21	22	23	24	25	26	27	28	29	30	31	32	33	34	35	36	37	38	39
Variable																				
Value	?	?	?	?	?	?	?	?	?	?	?	?	?	?	?	?	?	?	?	?

Address	40	41	42	43	44	45	46	47	48	49	50	51	52	53	54	55	56	57	58	59
Variable																				
Value	?	?	?	?	?	?	?	?	?	?	?	?	?	?	?	?	?	?	?	?

Address	60	61	62	63	64	65	66	67	68	69	70	71	72	73	74	75	76	77	78	79
Variable																				
Value	?	?	?	?	?	?	?	?	?	?	?	?	?	?	?	?	?	?	?	?

Address	80	81	82	83	84	85	86	87	88	89	90	91	92	93	94	95	96	97	98	99
Variable																				
Value	?	?	?	?	?	?	?	?	?	?	?	?	?	?	?	?	?	?	?	?

Address	100	101	102	103	104	105	106	107	108	109	110	111	112	113	114	115	116	117	118	119
Variable																				
Value	?	?	?	?	?	?	?	?	?	?	?	?	?	?	?	?	?	?	?	?

Address	120	121	122	123	124	125	126	127	128	129	130	131	132	133	134	135	136	137	138	139
Variable																				
Value	?	?	?	?	?	?	?	?	?	?	?	?	?	?	?	?	?	?	?	?

```c
char *create_string(int len, char c)
{
	char *s;
	int i;

	s = malloc((len + 1) * sizeof(char));
	if (s == NULL)
	{
		return (NULL);
	}
	i = 0;
	while (i < len)
	{
		s[i] = c;
		i++
	}
	s[i] = '\0';
	return (s);
}

int main(void)
{
	char *s;

	s = create_string(5, 'H');
	[...]
```

free

When the allocated memory is not needed anymore, you must return it to the operating system by calling the function free.

void free(void *ptr);

The free() function frees the memory space pointed to by ptr, which must have been returned by a previous call to malloc(), calloc() or realloc().

Address	40	41	42	43	44	45	46	47	48	49	50	51
Variable	"%s\n"											
Value	%	s	\n	\0	?	?	?	?	?	?	?	?

Address	60	61	62	63	64	65	66	67	68	69
Variable	n				ar					
Value	5				80					

Address	80	81	82	83	84	85	86	87	88	89
Variable										
Value	C	o	o	l	\0	?	?	?	?	?

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

int main(void)
{
	int n;
	char *ar;

	n = 5;
	ar = malloc(n * sizeof(char));
	ar[0] = 'C';
	ar[1] = 'o';
	ar[2] = 'o';
	ar[3] = 'l';
	ar[4] = '\0';
	printf("%s\n", ar); /* prints Cool\n */
	free(ar);
	return (0);
}
```

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

void print_int_array(int *a, int size)
{
	int i;

	i = 0;
	while (i < size)
	{
		printf("%d\n", a[i]);
		i++;
	}
}

int main(int ac, char **av)
{
	int *a;
	int asize;
	int i;

	if (ac < 2)
	{
		printf("Please give me at least one number\n");
		printf("Usage: %s number [NUMBER]\n", av[0]);
		return (1);
	}
	asize = ac - 1;
	a = malloc(asize * sizeof(int));
	i = 0;
	while (i < asize)
	{
		a[i] = atoi(av[i + 1]);
		i++;
	}
	print_int_array(a, asize);
	free(a);
	return (0);
}
```

Never trust anyone

Sometimes, malloc fails. On error, malloc returns NULL.

Always check its return value.

```
ubuntu@ip-172-31-63-244:/tmp/sf$ cat main5.c
#include <stdlib.h>
#include <stdio.h>
#include <limits.h>

int main(void)
{
  char *s;

  while (1)
    {
      s = malloc(INT_MAX);
      s[0] = 'H';
    }
  return (0);
}
ubuntu@ip-172-31-63-244:/tmp/sf$ gcc main5.c -Wall -Werror -Wextra -pedantic
ubuntu@ip-172-31-63-244:/tmp/sf$ ./a.out
Segmentation fault (core dumped)
```

```
ubuntu@ip-172-31-63-244:/tmp/sf$ cat main6.c
#include <stdlib.h>
#include <stdio.h>
#include <limits.h>

int main(void)
{
  char *s;

  while (1)
    {
      s = malloc(INT_MAX);
      if (s == NULL)
        {
          fprintf(stderr, "Not enough memory left!\n");
          return (1);
        }
    }
  return (0);
}
ubuntu@ip-172-31-63-244:/tmp/sf$ gcc main6.c -Wall -Werror -Wextra -pedantic
ubuntu@ip-172-31-63-244:/tmp/sf$ ./a.out
Not enough memory left!
```