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.



"Hello World\n"

W

S

W

ar

?

?

d

d

Example

H

H

e

е

?

Address

Variable

Address

Variable

Address

Variable

Value

Value

Value

fun(int a)

int ar[3];

int b;

 $[\ldots]$

?

?

?

?

\n

\n

char s[] = "Hello World\n";

?

a

b

?

?

?

?

?

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

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

malloc

#include <stdlib.h>

void *malloc(size_t size);

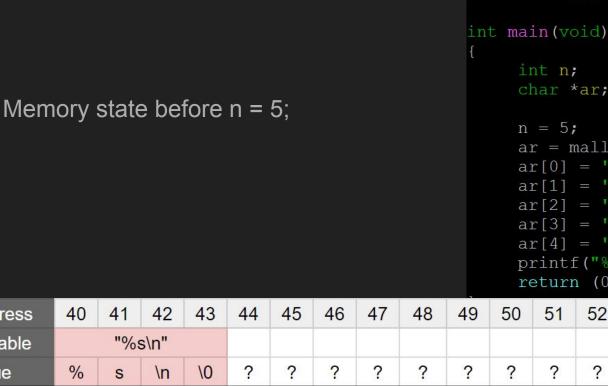
The malloc() function allocates size bytes and returns a pointer to the allocated memory

The memory is not initialized

```
Example
               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] = ' \setminus 0';
                    printf("%s\n", ar); /* prints Cool\n */
                    return (0);
```

#include <stdio.h>

#include <stdlib.h>



Address

Variable

Address

Variable

Value

Value

40

%

60

41

S

61

42

\n

62

"%s\n"

n

?

43

10

63

64

65

66

67

68

ar

?

69

n = 5;ar = malloc(n * sizeof(char)); ar[0] = 'C';ar[1] = 'o';ar[2] = 'o';ar[3] = 'l';

50

70

51

?

71

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

> int n; char *ar;

 $ar[4] = ' \ 0';$ printf("%s\n", ar); /* prints Cool\n */ return (0);

52

?

72

?

53

73

54

?

74

?

55

?

75

?

56

?

76

?

57

?

77

?

58

?

78

?

59

79

#include <stdio.h> #include <stdlib.h> int main (void) int n; char *ar; Memory state after the call to malloc n = 5;ar = malloc(n * sizeof(char)); 43 45 46 47 48 51 40 41 42 44 49 50 Address "%s\n" Variable ? ? % ? ? ? ? ? Value 10 ? S \n Address 60 61 62 63 64 65 66 67 68 69 70 71 Variable n ar Value 5 80 82 83 87 Address 80 81 84 85 86 88 89 90 91

\n", ar); /* prints Cool\n */

?

?

Variable

Value

Memory state before the call to printf

44

64

84

10

42

\n

62

82

0

41

S

61

81

0

"%s\n"

n

5

43

10

63

83

Address

Variable

Address

Variable

Address

Variable Value

Value

Value

40

%

60

80

C

45

?

65

85

46

?

66

86

?

47

67

87

48

68

88

ar

80

50

70

90

51

?

71

91

?

49

69

89

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

int main (void)

int n; char *ar;

n = 5;

ar[0] = 'C';

ar[1] = 'o';

ar[2] = 'o';ar[3] = '1';

 $ar[4] = ' \ 0';$

return (0);

ar = malloc(n * sizeof(char));

printf("%s\n", ar); /* prints Cool\n */

Exercise

What does this program do?

Represent the memory state, step by step.

```
#include <stdlib.h>
#include <stdio.h>
void print int array(int *a, int size)
     while (i < size)
         printf("%d\n", a[i]);
         i++;
int main(int ac, char **av)
    int *a;
     int asize;
    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));
    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																					#include <stdlib.h></stdlib.h>
Value		?	?	?	?	?	?	?	?	?	?	?	?	?	?	?	?	?	?	?	#include <stdio.h></stdio.h>
																					void print int array(int *a, int size)
Address	20	21	22	23	24	25	26	27	28	29	30	31	32	33	34	35	36	37	38	39	int i;
Variable																					
Value	?	?	?	?	?	?	?	?	?	?	?	?	?	?	?	?	?	?	?	?	<pre>i = 0; while (i < size)</pre>
				1000				1000			10.00	70.11							10,171		{
Address	40	41	42	43	44	45	46	47	48	49	50	51	52	53	54	55	56	57	58	59	printf("%d\n", a[i]); i++;
Variable																					· }
Value	?	?	?	?	?	?	?	?	?	?	?	?	?	?	?	?	?	?	?	?	
		-		-						•							-	_			<pre>int main(int ac, char **av) {</pre>
Address	60	61	62	63	64	65	66	67	68	69	70	71	72	73	74	75	76	77	78	79	int *a; int asize;
Variable			-		•	00		0.								, ,			, ,	, 0	int i;
Value	?	?	?	?	?	?	?	?	?	?	?	?	?	?	?	?	?	?	?	?	if (ac < 2)
valuo			***		•	•	•		17.4	•		-		•				•			<pre>printf("Please give me at least one number\n");</pre>
Address	80	81	82	83	84	85	86	87	88	89	90	91	92	93	94	95	96	97	98	99	<pre>printf("Usage: %s number [NUMBER]\n", av[0]);</pre>
Variable	00		OL.		01	00	00	01	00	00	00	01	UZ.	00	01		00	01	00	00	return (1);
Value	?	?	?	?	?	?	?	?	?	?	?	?	?	?	?	?	?	?	?	?	<pre>asize = ac - 1; a = malloc(asize * sizeof(int));</pre>
value		•	•		•	•	•	•	•	•	•				•						i = 0;
Address	100	101	102	103	104	105	106	107	108	100	110	111	112	113	11/	115	116	117	118	110	<pre>while (i < asize) {</pre>
Variable	100	101	102	103	104	103	100	107	100	103	110	111	112	113	114	113	110	117	110	113	a[i] = atoi(av[i + 1]);
Value	?	?	?	?	?	?	?	?	?	?	?	?	?	?	?	?	?	?	?	?	i++; }
value			9.7	•			- 1		1	:	•		:	5.7	- 1	1 2 2			31		print_int_array(a, asize); return (0);
Address	120	121	122	122	124	125	126	127	128	129	120	121	122	122	12/	125	126	127	120	139	}
	120	121	122	123	124	123	120	121	120	129	130	131	132	133	134	133	130	137	138	139	
Variable	_	0		_	_	_	_	_	_	_	_	_	_	_	_	_	_	_	_	_	
Value	?	?	?	?	?	?	?	?	?	?	?	?	?	?	?	?	?	?	?	?	

Exercise

Represent the memory state, step by step.

What does this program do?

```
if (s == NULL)
           return (NULL);
     i = 0;
     while (i < len)
           s[i] = c;
           i++
     s[i] = ' \setminus 0';
     return (s);
int main (void)
     char *s;
     s = create string(5, 'H');
```

char *create string(int len, char c)

s = malloc((len + 1) * sizeof(char));

char *s;

int i;

 $[\ldots]$

Address		1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	
Variable																					
Value		?	?	?	?	?	?	?	?	?	?	?	?	?	?	?	?	?	?	?	
																					<pre>char *create_string(int len, char c)</pre>
Address	20	21	22	23	24	25	26	27	28	29	30	31	32	33	34	35	36	37	38	39	t char *s;
Variable																					int i;
Value	?	?	?	?	?	?	?	?	?	?	?	?	?	?	?	?	?	?	?	?	
																					s = malloc((len + 1) * sizeof(char));
Address	40	41	42	43	44	45	46	47	48	49	50	51	52	53	54	55	56	57	58	59	if (s == NULL)
Variable																					return (NULL);
Value	?	?	?	?	?	?	?	?	?	?	?	?	?	?	?	?	?	?	?	?	}
																					i = 0;
Address	60	61	62	63	64	65	66	67	68	69	70	71	72	73	74	75	76	77	78	79	while (i < len)
Variable																					s[i] = c;
Value	?	?	?	?	?	?	?	?	?	?	?	?	?	?	?	?	?	?	?	?	i++
																					}
Address	80	81	82	83	84	85	86	87	88	89	90	91	92	93	94	95	96	97	98	99	s[i] = '\0'; return (s);
Variable																					}
Value	?	?	?	?	?	?	?	?	?	?	?	?	?	?	?	?	?	?	?	?	
																					int main(void)
Address	100	101	102	103	104	105	106	107	108	109	110	111	112	113	114	115	116	117	118	119	{ char *s;
Variable																					Clid1 "b;
Value	?	?	?	?	?	?	?	?	?	?	?	?	?	?	?	?	?	?	?	?	<pre>s = create_string(5, 'H');</pre>
																					[]
Address	120	121	122	123	124	125	126	127	128	129	130	131	132	133	134	135	136	137	138	139	
Variable																					
Value	?	?	?	?	?	?	?	?	?	?	?	?	?	?	?	?	?	?	?	?	

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	?	?	?	?	?	?	?	?
												ude <stdio.h> ude <stdlib.h></stdlib.h></stdio.h>
Address	60	61	62	63	64	65	66	67	68	69		
Variable		l)	n					a	ar		{	ain(void)
Value		,	5					8	80			int n; char *ar;
9												n = 5;
Address	80	81	82	83	84	85	86	87	88	89		<pre>ar = malloc(n * sizeof(char)); ar[0] = 'C';</pre>
Address Variable	80	81	82	83	84	85	86	87	88	89		ar = malloc(n * sizeof(char));
	80 C	81	82 o	83 I	84	85	86	87	88	89		<pre>ar = malloc(n * sizeof(char)); ar[0] = 'C'; ar[1] = 'o';</pre>
Variable		100201	12-40	83 I								<pre>ar = malloc(n * sizeof(char)); ar[0] = 'C'; ar[1] = 'o'; ar[2] = 'o'; ar[3] = '1';</pre>

```
#include <stdlib.h>
#include <stdio.h>
void print int array(int *a, int size)
    while (i < size)
         printf("%d\n", a[i]);
         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));
    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!
```