02_12_ Memory Allocation

- So far you've learned how to declare and reserve a piece of memory space before it is used in your program.

- For instance, you have to specify the size of an array in your program (or the compiler has to figure out the size if you declare an unsized array before you assign any data to it at runtime.

- In this lesson you'll learn to allocate memory space dynamically when your program is running.

Topics:

- The `malloc()` function
- The `calloc()` function
- The `realloc()` function
- The `free()` function

Dynamic memory allocation functions in C

- C language offers 4 dynamic memory allocation functions. They are

S.no	Function	Syntax
1	`malloc ()`	`malloc (number*sizeof(int));`
2	`calloc ()`	`calloc (number, sizeof(int));`
3	`realloc ()`	`realloc (pointer_name, number * sizeof(int));`
4	`free ()`	`free (pointer_name);`

Allocating Memory at Runtime

- There are many cases when you do not know the exact sizes of arrays used in your programs, until much later when your programs are actually being executed.

- You can specify the sizes of arrays in advance, but the arrays can be too small or too big if the numbers of data items you want to put into the arrays change dramatically at runtime.

Allocating Memory at Runtime

- C provides you with four dynamic memory allocation functions that you can employ to allocate or reallocate certain memory spaces while your program is running.

- Also, you can release allocated memory storage as soon as you don't need it.

- These four C functions, `malloc()`, `calloc()`, `realloc()`, and `free()`, are introduced in the following sections.

The `malloc()` Function

- **`malloc ()`** function is used to allocate space in memory during the execution of the program.
- **`malloc ()`** does not initialize the memory allocated during execution. It carries garbage value.
- **`malloc ()`** function returns null pointer if it couldn't able to allocate requested amount of memory.

The `malloc()` Function

- Note that the header file, `stdlib.h`, has to be included before the `malloc()` function can be called.
- If the `malloc()` function fails to allocate a piece of memory space, it returns a null pointer. Normally, this happens when there is not enough memory.
- Therefore, you should always check the returned pointer from `malloc()` before you use it.

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

int main()
{
      char *mem_allocation;

      /* memory is allocated dynamically */
      mem_allocation = malloc( 20 * sizeof(char) );
      if( mem_allocation== NULL )
      {
         printf("Couldn't able to allocate requested memory\n");
      }
      else
      {
         strcpy( mem_allocation,"I believe I can fly");
      }
      printf("Dynamically allocated memory content : %s\n", mem_allocation );
      free(mem_allocation);
}
```

```
Dynamically allocated memory content : I believe I can fly
```

The `malloc()` Function

- There is a potential problem if you keep allocating memory, because there is always a limit.
- You can easily run out of memory when you just allocate memory without releasing it.
- In the next section, you'll learn how to use the `free()` function to free up memory spaces allocated for you when you don't need them.

Releasing Allocated Memory with `free()`

- The function `malloc()` is used to allocate a certain amount of memory during the execution of a program.
- The `malloc()` function will request a block of memory from the heap.
- If the request is granted, the operating system will reserve the requested amount of memory.
- When the amount of memory is not needed anymore, you must return it to the operating system by calling the function `free()`.
- Because memory is a limited resource, you should allocate an exactly sized piece of memory right before you need it, and release it as soon as you don't need it.

The `calloc()` Function

- Besides the **`malloc()`** function, you can also use the **`calloc()`** function to allocate a memory storage dynamically.
- c in **`calloc()`** stands for clear, or The name **`calloc()`** stands for "contiguous allocation"

- **Note: `calloc()` zero-initializes the buffer, while `malloc()` leaves the memory uninitialized.**

- There is no such guarantee that the memory space allocated by **`malloc()`** is initialized to 0.

- The **`calloc()`** function returns a void pointer too.

- If the **`calloc()`** function fails to allocate a piece of memory space, it returns a null pointer.

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

int main()
{
     char *mem_allocation;
     /* memory is allocated dynamically */
     mem_allocation = calloc( 20, sizeof(char) );
     if( mem_allocation== NULL )
     {
        printf("Couldn't able to allocate requested memory\n");
     }
     else
     {
         strcpy( mem_allocation,"I believe I can fly");
     }
         printf("Dynamically allocated memory content: %s\n", mem_allocation );
     free(mem_allocation);
}
```

```
Dynamically allocated memory content : I believe I can fly
```

The `realloc()` Function

- The **`realloc()`** function gives you a means to change the size of a piece of memory space allocated by the **`malloc()`** function, the **`calloc()`** function, or even itself.

- **`realloc ()`** function modifies the allocated memory size by **`malloc ()`** and **`calloc ()`** functions to new size.

- If enough space doesn't exist in memory of current block to extend, new block is allocated for the full size of reallocation, then copies the existing data to new block and then frees the old block.

`free()` function in C

- **`free ()`** function frees the allocated memory by **`malloc (), calloc (), realloc ()`** functions and returns the memory to the system.

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

int main()
{
    char *mem_allocation;
    /* memory is allocated dynamically */
     mem_allocation = malloc( 20 * sizeof(char) );
    if( mem_allocation == NULL )
    {
        printf("Couldn't able to allocate requested memory\n");
    }
    else
    {
       strcpy( mem_allocation, "I believe I can fly");
    }
    printf("Dynamically allocated memory content  : %s\n", mem_allocation );

     mem_allocation = realloc(mem_allocation,100*sizeof(char));
    if( mem_allocation == NULL )
    {
        printf("Couldn't able to allocate requested memory\n");
    }
    else
    {
        strcpy( mem_allocation, "space is extended up to 100 characters");
    }
    printf("Resized memory : %s\n", mem_allocation );
     free(mem_allocation);
}
```

```
Dynamically allocated memory content  : I believe I can fly
Resized memory : space is extended up to 100 characters
```

`realloc()` function in C

- **Can I assume that calling `realloc` with a smaller size ?**

```
#include <stdlib.h>

int main()
{
    char* a = malloc(20000);
    char* b = realloc(a, 5);

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

```c
#include <stdlib.h>

int main()
{
    char* a = malloc(20000);
    char* b = realloc(a, 5);

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

- Yes, guaranteed by the C Standard if the new object can be allocated.
- Yes—if it succeeds.
- The key thing is that as a developer you no longer have responsibility for **`ptr`** if **`realloc`** returns non-null, and you do still have responsibility for it if **`realloc`** returns NULL.

Using realloc to shrink the allocated memory

- **If I use `realloc` to shrink the memory block that a pointer is pointing to, does the "extra" memory get freed? Or does it need to be freed manually somehow?**
- **Will I have a memory leak?**

```
int *myPointer = malloc(100*sizeof(int));
myPointer = realloc(myPointer,50*sizeof(int));
free(myPointer);
```

- No, you won't have a memory leak. **`realloc()`** will simply mark the rest "available" for future **`malloc`**()operations.

- But you still have to free **`myPointer`** later on. As an aside, if you use 0 as the size in **`realloc()`**, it will have the same effect as free on some implementations.

static memory allocation and dynamic memory

	Static memory allocation	Dynamic memory allocation
1	In static memory allocation, memory is allocated while writing the C program. Actually, user requested memory will be allocated at compile time.	In dynamic memory allocation, memory is allocated while executing the program. That means at run time.
2	Memory size can't be modified while execution. Example: `array`	Memory size can be modified while execution. Example: `Linked list`

End of 02_12