

To:
Tenouk

MODULE Z

THE C STORAGE CLASSES, SCOPE AND MEMORY ALLOCATION 2

My Training Period: zz hours

Note: [gcc](#) compilation examples are given at the end of this Module.

C abilities that supposed to be acquired:

- Understand and use the auto, register, extern and static keywords.
- Understand the basic of the **process address space**.
- Understand and appreciate the static, automatic and dynamic memory allocations.
- Understand how the memory is laid out for a running process.
- Understand and use the [malloc\(\)](#), [calloc\(\)](#), [realloc\(\)](#) and [free\(\)](#) functions.

Z.6 PROGRAM EXAMPLES

- The following program example allocates memory using [malloc\(\)](#). In this program we cast the `void *` to `int` type. There is no data stored in the memory requested; just an empty memory request and we do not de-allocate the memory as well.

```
/* playing with malloc(), memory on the heap */
#include <stdio.h>
#include <stdlib.h>

void main()
{
    int x;
    int *y;
    /* do 100000 times iteration, 100000 blocks */
    for(x=0; x<100000; x++)
    {
        /* For every iteration/block, allocate 16K,
        system will truncate to the nearest value */
        y = (int *)malloc(16384);
        /* if no more memory */
        if(y == NULL)
```

```

    {
        puts("No more memory lol!");
        /* exit peacefully */
        exit(0);
    }
    /* allocate the memory block, print the block and the address */
    printf("Allocating-->block: %i address: %p\n", x, y);
}
/* here, we do not free up the allocation */
}

```

Output:

```

E:\TEST\testproga\Debug\testproga.exe
Allocating-->block: 372 address: 00A027C8
Allocating-->block: 373 address: 00A06800
Allocating-->block: 374 address: 00A0A838
Allocating-->block: 375 address: 00A0E870
Allocating-->block: 376 address: 00A128A8
Allocating-->block: 377 address: 00A168E0
Allocating-->block: 378 address: 00A1A918
Allocating-->block: 379 address: 00A1E950
Allocating-->block: 380 address: 00A22988
Allocating-->block: 381 address: 00A269C0
Allocating-->block: 382 address: 00A2A9F8
Allocating-->block: 383 address: 00A2EA30
Allocating-->block: 384 address: 00A32A68
Allocating-->block: 385 address: 00A36AA0
Allocating-->block: 386 address: 00A3AAD8
Allocating-->block: 387 address: 00A3EB10

```

- Eventually, at some point in the run, the program will stop because there is no more memory to be allocated.
- During the program run, the paging or swapping activity will be obvious because system is serving the memory request. For Windows there may be message box indicating your system having low virtual memory. The worst case, your system may hang.
- The program uses the `(int *)` prototype to allocate 16,384 bytes (16K) of memory for every loop's iteration. `malloc()` returns the address of the memory block that successfully allocated.
- When `malloc()` returns `NULL`, it means no more memory could be allocated. The actual size of 16K chunks the program allocates depends upon which memory module/model your compiler is using.
- The `free()` function de-allocates memory allocated by `malloc()` and every time you allocate memory that's not used again in a program; you should use the `free()` function to release that memory to heap.
- The following program example de-allocates the memory for the previous program.

```

/* playing with free(), memory on the heap */
#include <stdio.h>
#include <stdlib.h>

```

```
void main()
```

```

{
    int x;
    int *y;
    int *buffer = NULL;
    /* do 100 times iteration, 100 blocks */
    for(x=0; x<100; x++)
    {
        /* for every iteration/block, allocate 16K,
        system will truncate to the nearest value */
        y = (int *)malloc(16384);
        /* if there is a problem */
        if(y == NULL)
        {
            puts("No more memory for allocation lol!");
            /* exit peacefully */
            exit(0);
        }
        /* allocate the memory block, print the block and the address */
        printf("Allocating-->block: %i address: %p\n", x, y);
        printf("---->Freeing the memory block: %i address: %p\n", x, y);
        free((void *)buffer);
    }
}

```

Output:

```

Select "E:\TEST\testproga\Debug\testproga.exe"
Allocating-->block: 1897 address: 021F4E78
---->Freeing the memory block: 1897 address: 021F4E78
Allocating-->block: 1898 address: 021F8EB0
---->Freeing the memory block: 1898 address: 021F8EB0
Allocating-->block: 1899 address: 021FCEE8
---->Freeing the memory block: 1899 address: 021FCEE8
Allocating-->block: 1900 address: 02200F20
---->Freeing the memory block: 1900 address: 02200F20
Allocating-->block: 1901 address: 02204F58
---->Freeing the memory block: 1901 address: 02204F58
Allocating-->block: 1902 address: 02208F90
---->Freeing the memory block: 1902 address: 02208F90
Allocating-->block: 1903 address: 0220CFC8
---->Freeing the memory block: 1903 address: 0220CFC8
Allocating-->block: 1904 address: 02211000
---->Freeing the memory block: 1904 address: 02211000

```

- You'll notice that the program runs all the way through, allocating memory and freeing it up so that you really never run out of memory.
- `free()` actually doesn't erase memory; it merely flags a chunk of memory as available for re-allocation by another `malloc()` function on the heap.
- The following program example demonstrates the use of `malloc()` and `calloc()` to allocate memory for an array of integers.
- You should always verify if the return value from `malloc()` and `calloc()` are `NULL` or not because the system may have run out of memory.

```

/* malloc() and struct */
#include <stdio.h>
#include <stdlib.h>

struct record{
char name[15];
int age;
int id_num;
};

int main()
{
    struct record *ptr;
    printf("\n--malloc() & struct--\n");
    ptr = (struct record *)malloc(sizeof(struct record));

    if(ptr)
    {
        printf("\nStudent Name: ");
        gets(ptr->name);
        printf("Student Age: ");
        scanf("%d", &ptr->age);
        printf("Student Id: ");
        scanf("%d", &ptr->id_num);
        printf("\nStudent Name: %s", ptr->name);
        printf("\nStudent Age: %d", ptr->age);
        printf("\nStudent Id Number: %d\n", ptr->id_num);
        free(ptr);
    }
    else
        printf("\nMemory allocation fails!!!\n");
    return 0;
}

```

Output:

```

C:\d:\testprog\Debug\testprog...
--malloc() & struct--
Student Name: Dumb Ass Mickey
Student Age: 19
Student Id: 2087

Student Name: Dumb Ass Mickey
Student Age: 19
Student Id Number: 2087
Press any key to continue

```

- Another malloc() and calloc() program example.

```

/* playing with malloc() and calloc() */
#include <stdio.h>

```

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

```
#define END 10
```

```
int main()
```

```
{
```

```
    int *ptr1, *ptr2, *ptr3;
```

```
    int i;
```

```
    /* get memory for an array using malloc() - 1 parameter */
```

```
    ptr1 = (int *) malloc(END*sizeof(int));
```

```
    /* if memory allocation fails... */
```

```
    if (ptr1 == NULL)
```

```
    {
```

```
        fprintf(stderr, "malloc() failed!\n");
```

```
        /* exit with an error message */
```

```
        exit(1);
```

```
    }
```

```
    /* initialize the array using array notation */
```

```
    for(i = 0; i < END; i++)
```

```
    {
```

```
        ptr1[i] = i+i;
```

```
    }
```

```
    /******
```

```
    /* getting memory for an array using calloc() - 2 parameters */
```

```
    ptr2 = (int *) calloc(END, sizeof(int));
```

```
    /* if memory allocation fails... */
```

```
    if(ptr2 == NULL)
```

```
    {
```

```
        fprintf(stderr, "calloc() failed!\n");
```

```
        /* exit with an error message */
```

```
        exit(1);
```

```
    }
```

```
    /* initialize the array using pointer arithmetic */
```

```
    ptr3 = ptr2;
```

```
    for(i = 0; i < END; i++)
```

```
    {
```

```
        *(ptr3++) = i+i;
```

```
    }
```

```
    /* print array contents */
```

```
    printf("---Using malloc()---\n");
```

```
    printf("Array pointed by ptr1:\n");
```

```
    for(i = 0; i < END; i++)
```

```
    {
```

```
        printf("%3d ", ptr1[i]);
```

```
    }
```

```
    printf("\n\n");
```

```
    printf("---Using calloc()---\n");
```

```
    printf("Array pointed by ptr2:\n");
```

```

for(i = 0; i < END; i++)
{
    printf("%3d ", ptr2[i]);
}
printf("\n\n");
return 0;
}

```

Output:

```

C:\> "d:\testprog\Debug\testprog.exe"
---Using malloc()---
Array pointed by ptr1:
 0  2  4  6  8 10 12 14 16 18

---Using calloc()---
Array pointed by ptr2:
 0  2  4  6  8 10 12 14 16 18

Press any key to continue

```

- More calloc() and malloc() program example.

```

/* calloc() and malloc() example */
#include <stdlib.h>
#include <stdio.h>
#define n 10

/* a struct */
typedef struct book_type
{
    int id;
    char name[20];
    float price;
}book;

int main(void)
{
    int *aPtr = NULL, *bPtr = NULL, m = 0;
    char *str = NULL;
    book *bookPtr = NULL;

    /* create an int array of size 10 */
    aPtr = (int *)calloc(n, sizeof(int));
    /* do some verification */
    if(aPtr == NULL)
    {
        printf("calloc for integer fails lol!\n");
        exit (0);
    }
}

```

```

else
    printf("memory allocation for int through calloc() is OK\n");

/* create a char array of size 10 */
str = (char *)calloc(n, sizeof(char));
if(str == NULL)
{
    printf("calloc for char fails lol!\n");
    exit (0);
}
else
    printf("memory allocation for char through calloc() is OK\n");

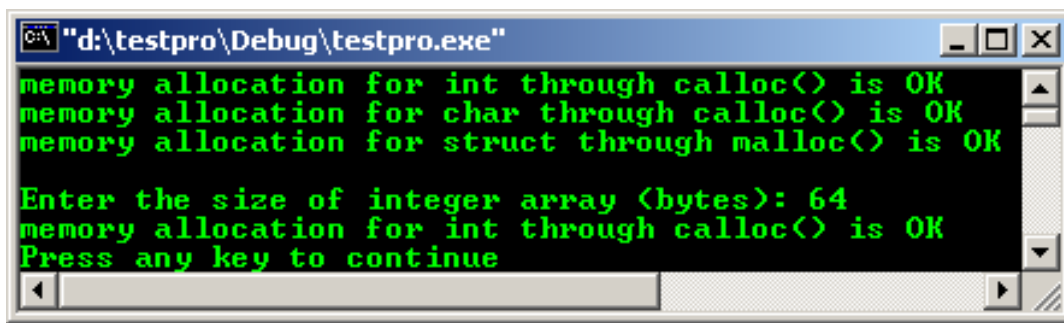
/* create a structure of book */
bookPtr = (book *)malloc(sizeof(book));
if(bookPtr == NULL)
{
    printf("malloc for struct fails lol!\n");
    exit (0);
}
else
    printf("memory allocation for struct through malloc() is OK\n");

/* clean up the memory allocated */
free(aPtr);
free(str);
free(bookPtr);

/* other way */
/* get the number of elements from the user and then allocate */
printf("\nEnter the size of integer array (bytes): ");
scanf("%d", &m);
bPtr = (int *)calloc(m, sizeof(int));
if(bPtr == NULL)
{
    printf("calloc for int fails lol!\n");
    exit (0);
}
else
    printf("memory allocation for int through calloc() is OK\n");
free(bPtr);
return 0;
}

```

Output:



```
C:\d:\testpro\Debug\testpro.exe
memory allocation for int through calloc() is OK
memory allocation for char through calloc() is OK
memory allocation for struct through malloc() is OK

Enter the size of integer array (bytes): 64
memory allocation for int through calloc() is OK
Press any key to continue
```

- Program example for `realloc()`.

```
/* playing with realloc(). Store user input in an array */
#include <stdio.h>
#include <stdlib.h>

#define INITIAL_SIZE 5;

int main()
{
    int *Arr, *temp;
    int limit, input, n = 0, r, i;

    /* initially, allocate some space for A */
    limit = INITIAL_SIZE;
    Arr = (int *) malloc (limit * sizeof(int));
    /* do some verification, if fail */
    if (Arr == NULL)
    {
        /* display the error message */
        fprintf(stderr, "malloc() failed!\n");
        /* exit with the error code */
        exit(1);
    }

    /* array loop */
    printf("Enter numbers, 1 per line. End with ctrl-D\n");
    while(1)
    {
        printf("Next number: ");
        r = scanf("%d", &input);
        fflush(stdin);

        /* verify the input */
        if(r < 1)
            break;
        /* get more space for Arr using realloc() */
        if(n >= limit)
        {
            printf("More than 5 elements per loop, reallocating the storage... \n");
            limit = 2 * limit;
            temp = (int *)realloc(Arr, limit * sizeof(int));
            /* verify again... */
        }
    }
}
```



```

    if(temp == NULL)
    {
        fprintf(stderr, "realloc() failed!\n");
        exit(1);
    }
    else
        printf("realloc is OK lol, proceed your input...\n");
    Arr = temp;
}

Arr[n] = input;
n++;
}

/* trim Arr down to size */
temp = (int *)realloc(Arr, n*sizeof(int));
/* verify... */
if(temp == NULL)
{
    fprintf(stderr, "realloc() fails lol!\n");
    exit(1);
}

Arr = temp;

printf("\nContents of the array Arr:\n");
/* print the array */
for(i = 0; i < n; i++)
{
    printf("%2d ", Arr[i]);
}
printf("\n");
return 0;
}

```

Output:

```

C:\E:\TEST\testproga\Debug\testproga.exe
Enter numbers, 1 per line. End with ctrl-D
Next number: 3
Next number: 5
Next number: 8
Next number: 90
Next number: 23
Next number: 14
More than 5 elements per loop, reallocating the storage...
realloc is OK lol, proceed your input...
Next number: 56
Next number: 12
Next number: 63
Next number: 67
Next number: 9
More than 5 elements per loop, reallocating the storage...
realloc is OK lol, proceed your input...
Next number: 3
Next number: 9
Next number: 12
Next number: ^D

Contents of the array Arr:
3 5 8 90 23 14 56 12 63 67 9 3 9 12
Press any key to continue

```

- Program examples compiled using [gcc](#).

```

/* calloc() and malloc() example */
#include <stdlib.h>
#include <stdio.h>
#define n 10

/* a struct */
typedef struct book_type
{
    int id;
    char name[20];
    float price;
}book;

int main(void)
{
    int *aPtr = NULL, *bPtr = NULL, m = 0;
    char *str = NULL;
    book *bookPtr = NULL;

    /* create an int array of size 10 */
    aPtr = (int *)calloc(n, sizeof(int));
    /* do some verification */
    if(aPtr == NULL)
    {
        printf("calloc for integer fails lol!\n");
        exit (0);
    }
    else

```

```

printf("memory allocation for int through calloc() is OK\n");

/* create a char array of size 10 */
str = (char *)calloc(n, sizeof(char));
if(str == NULL)
{
    printf("calloc for char fails lol!\n");
    exit (0);
}
else
    printf("memory allocation for char through calloc() is OK\n");

/* create a structure of book */
bookPtr = (book *)malloc(sizeof(book));
if(bookPtr == NULL)
{
    printf("malloc for struct fails lol!\n");
    exit (0);
}
else
    printf("memory allocation for struct through malloc() is OK\n");

/* clean up the memory allocated */
free(aPtr);
free(str);
free(bookPtr);

/* other way */
/* get the number of elements from the user and then allocate */
printf("\nEnter the size of integer array (bytes): ");
scanf("%d", &m);
bPtr = (int *)calloc(m, sizeof(int));
if(bPtr == NULL)
{
    printf("calloc for int fails lol!\n");
    exit (0);
}
else
    printf("memory allocation for int through calloc() is OK\n");
free(bPtr);
return 0;
}

```

- Another program example.

```

[bodo@bakawali ~]$ gcc memalloc.c -o memalloc
[bodo@bakawali ~]$ ./memalloc

```

```

memory allocation for int through calloc() is OK
memory allocation for char through calloc() is OK
memory allocation for struct through malloc() is OK

```

Enter the size of integer array (bytes): 37
memory allocation for int through calloc() is OK

```
/****** malalloc.c *****/
/******run on FeDora 3 Machine***** */
/* playing with malloc() and free(), memory on the heap */
#include <stdio.h>
#include <stdlib.h>

int main()
{
    int x;
    int *y;
    int *buffer = NULL;
    /* do 100 times iteration, 100 blocks */
    for(x=0; x<100; x++)
    {
        /* for every iteration/block, allocate 16K,
        system will truncate to the nearest value */
        y = (int *)malloc(16384);
        /* if there is a problem */
        if(y == NULL)
        {
            puts("No more memory for allocation lol!");
            /* exit peacefully */
            exit(0);
        }
        else
        {
            /* allocate the memory block, print the block and the address */
            printf("Allocating-->block: %i address: %p\n", x, y);
            free((void *)buffer);
            printf("---->Freeing the memory block: %i address: %p\n", x, y);
        }
    }
    return 0;
}
```

```
[bodo@bakawali ~]$ gcc malalloc.c -o malalloc
[bodo@bakawali ~]$ ./malalloc
```

```
Allocating-->block: 0 address: 0x804a008
---->Freeing the memory block: 0 address: 0x804a008
Allocating-->block: 1 address: 0x804e010
---->Freeing the memory block: 1 address: 0x804e010
Allocating-->block: 2 address: 0x8052018
---->Freeing the memory block: 2 address: 0x8052018
Allocating-->block: 3 address: 0x8056020
---->Freeing the memory block: 3 address: 0x8056020
Allocating-->block: 4 address: 0x805a028
```

---->Freeing the memory block: 4 address: 0x805a028
Allocating-->block: 5 address: 0x805e030
---->Freeing the memory block: 5 address: 0x805e030
...

-----o0o-----

Further reading and digging:

1. [Check the best selling C and C++ books at Amazon.com.](#)
2. [C and buffer overflow tutorial: stack frame activity.](#)
3. [C and Assembler, Compiler and Linker.](#)

[| C Storage Class & Memory 1](#) | [Main](#) | [Microsoft & C/Win32](#) | [C++/OOP](#) | [Site Index](#) |
[Download](#) |

C Storage Class & Memory: [Part 1](#) | [Part 2](#) |

To:
Tenouk 2003-2005 © Tenouk. All rights reserved.