Memory Management – II

**Question:** How to deallocate dynamically allocate memory without using “free()” function.

**Solution:** Standard library function [realloc()](http://www.cplusplus.com/reference/clibrary/cstdlib/realloc/) can be used to deallocate previously allocated memory. Below is function declaration of “realloc()” from “stdlib.h”

|  |
| --- |
| void \*realloc(void \*ptr, size\_t size); |

If “size” is zero, then call to realloc is equivalent to “free(ptr)”. And if “ptr” is NULL and size is non-zero then call to realloc is equivalent to “malloc(size)”.

Let us check with simple example.

|  |
| --- |
| /\* code with memory leak \*/  #include <stdio.h>  #include <stdlib.h>    int main(void)  {      int \*ptr = (int\*)malloc(10);        return 0;  } |

Check the leak summary with valgrind tool. It shows memory leak of 10 bytes, which is highlighed in red colour.

[narendra@ubuntu]$ valgrind –leak-check=full ./free

==1238== LEAK SUMMARY:

==1238== definitely lost: 10 bytes in 1 blocks.

==1238== possibly lost: 0 bytes in 0 blocks.

==1238== still reachable: 0 bytes in 0 blocks.

==1238== suppressed: 0 bytes in 0 blocks.

[narendra@ubuntu]$

Let us modify the above code.

|  |
| --- |
| #include <stdio.h>  #include <stdlib.h>    int main(void)  {      int \*ptr = (int\*) malloc(10);        /\* we are calling realloc with size = 0 \*/      realloc(ptr, 0);          return 0;  } |

Check the valgrind’s output. It shows no memory leaks are possible, highlighted in red color.

[narendra@ubuntu]$ valgrind –leak-check=full ./a.out

==1435== ERROR SUMMARY: 0 errors from 0 contexts (suppressed: 11 from 1)

==1435== malloc/free: in use at exit: 0 bytes in 0 blocks.

==1435== malloc/free: 1 allocs, 1 frees, 10 bytes allocated.

==1435== For counts of detected errors, rerun with: -v

==1435== All heap blocks were freed — no leaks are possible.

# Use of realloc()

Size of dynamically allocated memory can be changed by using realloc().

As per the C99 standard:

|  |
| --- |
| void \*realloc(void \*ptr, size\_t size); |

*realloc deallocates the old object pointed to by ptr and returns a pointer to a new object that has the size specified by size. The contents of the new object is identical to that of the old object prior to deallocation, up to the lesser of the new and old sizes. Any bytes in the new object beyond the size of the old object have indeterminate values.*

The point to note is that **realloc() should only be used for dynamically allocated memory**. If the memory is not dynamically allocated, then behavior is undefined.

For example, program 1 demonstrates incorrect use of realloc() and program 2 demonstrates correct use of realloc().

**Program 1:**

|  |
| --- |
| #include <stdio.h>  #include <stdlib.h>  int main()  {     int arr[2], i;     int \*ptr = arr;     int \*ptr\_new;       arr[0] = 10;     arr[1] = 20;       // incorrect use of new\_ptr: undefined behaviour     ptr\_new = (int \*)realloc(ptr, sizeof(int)\*3);     \*(ptr\_new + 2) = 30;       for(i = 0; i < 3; i++)       printf("%d ", \*(ptr\_new + i));       getchar();     return 0;  } |

Output:  
Undefined Behavior

**Program 2:**

|  |
| --- |
| #include <stdio.h>  #include <stdlib.h>  int main()  {     int \*ptr = (int \*)malloc(sizeof(int)\*2);     int i;     int \*ptr\_new;       \*ptr = 10;     \*(ptr + 1) = 20;    **ptr\_new = (int \*)realloc(ptr, sizeof(int)\*3);**     \*(ptr\_new + 2) = 30;     for(i = 0; i < 3; i++)         printf("%d ", \*(ptr\_new + i));       getchar();     return 0;  } |

Output:  
*10 20 30*

Whether realloc deallocates the old pointer and and allocates a completely new memory or it expands the current memory size depends on how much of memory is available. If there is enough memory after the previously allocated one, it will just expand it, that might be happening in your case. If there is no sufficient space, then it will deallocate the old memory and will allocate new.

Another example using **realloc():**

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

int main(int argc, const char \* argv[])

{

/\* Define required variables \*/

char \*ptr1, \*ptr2;

size\_t length1, length2;

/\* Define the amount of memory required \*/

length1 = 10;

length2 = 30;

/\* Allocate memory for our string \*/

ptr1 = (char \*)malloc(length1);

/\* Check to see if we were successful \*/

if (ptr1 == NULL)

{

/\* We were not so display a message \*/

printf("Could not allocate required memory\n");

/\* And exit \*/

exit(1);

}

/\* Copy a string into the allocated memory \*/

strcpy(ptr1, "C malloc");

/\* Oops, we wanted to say more but now do not

have enough memory to store the message! \*/

/\* Expand the available memory with realloc \*/

ptr2 = (char \*)realloc(ptr1, length2);

/\* Check to see if we were successful \*/

if (ptr2 == NULL)

{

/\* We were not so display a message \*/

printf("Could not re-allocate required memory\n");

/\* And exit \*/

exit(1);

}

/\* Add the rest of the message to the string \*/

strcat(ptr2, " at TechOnTheNet.com");

/\* Display the complete string \*/

printf("%s\n", ptr2);

/\* Free the memory we allocated \*/

free(ptr2);

return 0;

}

<http://www.ibm.com/developerworks/linux/library/l-memory/>