Dynamic allocation in C/C++

Graham (15) Feb 19, 2008 at 4:48pm

WHAT IT IS

Dynamic allocation is the automatic allocation of memory in C/C++, Unlike declarations, which load data onto the programs data segment, dynamic allocation creates new usable space on the programs STACK (an area of RAM specifically allocated to that program).

It is accomplished by two functions (in C) and two operators (in C++):
malloc and free, new and delete.

WHY YOU USE IT

Dynamic allocation can be quite useful. For example, a dynamically sized array:

Assume you have variable size as a variable (it is NOT defined as constant). The compiler will give you an error if you try this:
BOTH VERSIONS

```
int array[size];
```

What you can do is allocate this memory dynamically. For example:
C VERSION:

```
1 int *array;
2 array=(int *) malloc(size*sizeof(int));
```

C++ VERSION:

```
1 int *array;
2 array=new int[size]l
```

NOTICE:
Since this data is pushed on the stack, not the segment, a possibility of MEMORY LEAKS occurs. Memory leaks are where allocated memory is not freed before the program returns. For example, to safely deallocate the array just created (and normally return from the program):
C VERSION:

```
free(array);
```

C++ VERSION:

```
delete [] array
```

Another special notice: the brackets immediately following delete are only necessary if you allocated an array originally.

Dynamic allocation is NECESSARY to there use of linked lists. Assuming self-referential structure Node has been declared with nextptr being the link:
C VERSION:

```
1 struct Node *newptr;
2 newptr=(struct Node *) malloc(sizeof(Node));
3 /*We'll insert this into this linked list just
4  at the second item, assuming topptr is the name
5  of the top of this list.*/
6 newptr->nextptr=topptr->nextptr;
7 topptr->nextptr=newptr;
```

C++ VERSION:

```
1 Node *newptr;
2 newptr=new Node;
3 //We'll insert this as the second item, using topptr.
4 newptr->nextptr=topptr->nextptr;
5 topptr->nextptr=newptr;
```

Of course, in both of these examples, no other members within structure Node are used, which defeats the purpose of this linked list.
To free a node:
C VERSION:

```
1 struct Node *tempptr;
2 /*The node we'll free is, again, the second.*/
3 tempptr=topptr->nextptr;
4 /*We'll unbind this node first, so as not to lose all of the rest of the list with it.*/
5 topptr->nextptr=tempptr->nextptr;
6 /*It's free, get rid of it.*/
7 free(tempptr);
```

C++ VERSION:

```
1 Node *tempptr;
2 //We'll get the second node again.
3 tempptr=topptr->nextptr;
```

```
4 //Unbind this node.
5 topptr->nextptr=tempptr->nextptr;
6 //Get rid of it.
7 delete tempptr;
```

HOW TO USE IT

The syntax, for the C functions, is:

```
1 #include <stdlib.h>
2
3 void *malloc(size_t size)
```

The casting in the previous examples is necessary because malloc returns a void pointer.

```
1 #include <stdlib.h>
2
3 void free(void *);
```

This doesn't need casting; anything passed is implicitly demoted.

The operators in C++ are:

```
NewAlloc=new Type;
```

C++ will automatically cast for you.

```
1       //If this is an array
2 delete [] NewAlloc;
3       //Otherwise
4 delete NewAlloc;
```

GOING FARTHER

Several classes, like vector and list, use this. Implement your own!

Understanding this article requires a good understanding of pointers and arrays (and their relationships). I know, I should put this at the top.

Define a class (I named mine SafeSpot) that will automatically handle the deallocation of memory defined within it when the object is destroyed. This will ELIMINATE all of those unforseen memory leaks. You may want to try different scopes of this class for different scopes of allocation (don't let a temp function object exist for the whole program).

Please reply, and tell me what you think of this article!
Thank you,
Graham Northup

Last edited on Feb 19, 2008 at 4:50pm

Mayflower (25) Feb 23, 2008 at 10:59am

I would have been doomed with memory leaks if it wasn't for this.

```
1 typedef unsigned char byte;
2 struct Color {
3     byte R;
4     byte G;
5     byte B;
6 }
7
8 Color * BitmapArray;
```

Made a dispose method for my Bitmap class... THANKS!

satm2008 (148) Mar 13, 2008 at 1:52am

Graham:
>>dynamic allocation creates new usable space on the programs STACK (an area of RAM specifically allocated to that program).

Correction: Dynamic allcoation does not create/allocate space in STACK segment where as it actually allocates in HEAP segment which is different from STACK.
Every program has its own three memory segments, ie, code segment in which the compiled (object/machine) code resideds, stack segment in which the local/temporary variables are stored and heap segment which is used for dynamic memory allocation.

Stack is very volatile as it changes every function (or a code-block) enters and exits. Meaning a variable created during a function call is created and destroyed as the call exits. When the segment is not initialized, it tends to have a garbage data.

Where as the heap is for dynamic allocation (by malloc() or new) and keeps the data until the allocation is cleared (by free() or delete).
The allocated memory belongs to the pointer pointed-by until it is freed/destroyned or the main program ends.
If the allocated memory is abandoned by its pointer then it causes a memory leak, meaning the allocated memory is abandoned (left ignored) and not allowing the same to be reused by other parts of the program. That is called a memory leak.
Please note that, malloc() or new do not guarantee that allocated space is blank and fresh hence you better use memset() or a constructor to make it fresh before using the allocated.

Hope it is clear now.

Good luck :)

Topic archived. No new replies allowed.