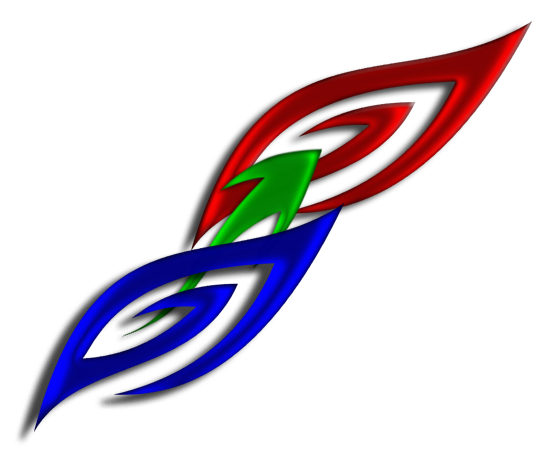
Wind

Software library for C++



Subhajit Sahu

Function Names

The function naming strategy is important for easy usability and understandability of the library as an API. It should be tried to remove unnecessary features, and map requisite feature to proper function names. Since functions perform an action, their names should be verbs, and as much as possible fall into the following names:

|  |  |
| --- | --- |
| Name | Use |
| Begin | Prepare a module before first use |
| End | End the use of a module |
| Create | Alternative to constructor, where it might need to be avoided |
| Destroy | Alternative to destructor where it might need to be avoided |
| Optimize | Take time to perform certain optimization |
|  |  |

# Configuring the Library

## Heap

Memory management is one of the most important challenges faced in the process of developing any dynamic program whose input data varies during runtime. Improper memory management can lead to reduced performance and may even cause the program to crash abruptly. Dynamic memory allocation is performed from a memory region called heap, and in Windows, each process can have multiple heaps. Each process also has a default heap of its own. It was tested and observed that memory allocation of large blocks a memory takes a lot of time. However, following are the objectives of dynamic memory allocation schemes:

* Heap allocation functions should support both C and C++ style functions (with C style functions containing the actual code, and C++ functions simply calling them).
* Care must be taken with constructors which otherwise might lead to unexpected results.
* Heap functions must use basic verbs for function names, if available.
* Since memory allocation is a costly operation, functions should constrain the programmer to put less stress on dynamic allocation.
* Heap functions must return both pointer to memory and size actually allocated, as return value.

|  |  |
| --- | --- |
| Flag | Meaning |
| HEAP\_FILL\_ZERO | Fills the allocated block of memory with zeroes. |
| HEAP\_GENERATES\_EXCEPTIONS | Exception is generated if a heap function fails. |
| HEAP\_NOT\_SERIALIZED | Access to heap functions is not controlled. |

Table : Flags taken as input to Heap functions

|  |  |
| --- | --- |
| Function | Use |
| heap\_Begin() | Prepares the Heap module for first use. |
| heap\_End() | Ends the use of the Heap module. |
| heap\_Create() | Creates a new Heap and returns its object. |
| heap\_Destroy() | Destroys a created heap. |
| heap\_Alloc() | Allocates memory from heap |
| heap\_ReAlloc() | Reallocates memory from heap |
| heap\_Free() | Free allocated memory from heap |
| heap\_Optimize() | Optimizes a heap so that further operations are faster. |

# Memory Management

Memory management is one of the most important challenges faced in the process of developing any dynamic program whose input data varies during runtime. Improper memory management can lead to reduced performance and may even cause the program to crash abruptly.

## Heap

Memory management is one of the most important challenges faced in the process of developing any dynamic program whose input data varies during runtime. Improper memory management can lead to reduced performance and may even cause the program to crash abruptly. Dynamic memory allocation is performed from a memory region called heap, and in Windows, each process can have multiple heaps. Each process also has a default heap of its own. It was tested and observed that memory allocation of large blocks a memory takes a lot of time. However, following are the objectives of dynamic memory allocation schemes:

## Heap

Memory management is one of the most important challenges faced in the process of developing any dynamic program whose input data varies during runtime. Improper memory management can lead to reduced performance and may even cause the program to crash abruptly. Dynamic memory allocation is performed from a memory region called heap, and in Windows, each process can have multiple heaps. Each process also has a default heap of its own. It was tested and observed that memory allocation of large blocks a memory takes a lot of time. However, following are the objectives of dynamic memory allocation schemes:

* Heap allocation functions should support both C and C++ style functions (with C style functions containing the actual code, and C++ functions simply calling them).
* Care must be taken with constructors which otherwise might lead to unexpected results.
* Heap functions must use basic verbs for function names, if available.
* Since memory allocation is a costly operation, functions should constrain the programmer to put less stress on dynamic allocation.
* Heap functions must return both pointer to memory and size actually allocated, as return value.

|  |  |
| --- | --- |
| Flag | Meaning |
| HEAP\_FILL\_ZERO | Fills the allocated block of memory with zeroes. |
| HEAP\_GENERATES\_EXCEPTIONS | Exception is generated if a heap function fails. |
| HEAP\_NOT\_SERIALIZED | Access to heap functions is not controlled. |

Table : Flags taken as input to Heap functions

|  |  |
| --- | --- |
| Function | Use |
| heap\_Begin() | Prepares the Heap module for first use. |
| heap\_End() | Ends the use of the Heap module. |
| heap\_Create() | Creates a new Heap and returns its object. |
| heap\_Destroy() | Destroys a created heap. |
| heap\_Alloc() | Allocates memory from heap |
| heap\_ReAlloc() | Reallocates memory from heap |
| heap\_Free() | Free allocated memory from heap |
| heap\_Optimize() | Optimizes a heap so that further operations are faster. |