Memory Management

We want to create our own memory management system, which actually is some kind of container of memory pages. Each page is just a chunk of bytes in memory.
User of this memory doesn’t know that when he writes/reads data through the Memory Pool and it’s possible that the data is even divided between different pages (see example below);

To create memory management system which is built from three components:

- base class memManager_t
- derived class memPage_t
- derived class memPool_t

functionality of memManager_t

- object of type memManager_t can 't be constructed
- possibility to get and set current position in memory
- to get following information about memory status:

  memory empty ?
  actual size of the memory (how many bytes really written in memory)
- to read data from memory – 2 functions:

  a) if position is not given, then from current position
  b) else from position given by user
- to write data into memory – 2 functions:

  a) if position is not given, then from current position
  b) else from position given by user

functionality of memPage_t

memPage_t has to hold any data as a stream of bytes

- object of type memPage_t can be constructed from:

  some default size (for example 1024 bytes) with the size provided by user
- copy of object of type memPage_t is forbidden
- possibility to get and set current position in memory buffer
- to get following information about page status:

  is page empty
  is page full
  actual size of the page (how many bytes really written in page)
  capacity of the page (length)
- to read data from page – 2 functions :

  a) if position is not given, then from current position
  b) else from position given by user
- to write data into page

  a) if position is not given, then from current position
  b) else from position given by user

functionality of memPool_t:

The role of memPool_t is to control placement of data in vector of memory pages and to provide user the following functionality:

- object of type memPool_t has to be constructed with one empty page
- copy of objects of type memPool_t is forbidden
- possibility to get and set current position in memPool_t
  (take in consideration how many bytes are really written in memPool_t)
- to get following information about Memory Pool status:
  empty ?
  actual size of the object memPool_t (how many bytes really written in pool)
- to read data from Memory Pool – 2 functions :
  c) if position is not given, then from current position
  d) else from position given by user
- to write data into Memory Pool
  c) if position is not given, then from current position
  d) else from position given by user
- to provide possibility to set and to get default size of memory page (**one for all pages**)

Example:

Object of type memPool _t contains 3 memPage_t pages.

Page 1
1024 bytes length (capacity)
1024 bytes actual size
Page 2
1024 bytes length (capacity)
1024 bytes actual size
Page 3
1024 bytes length (capacity)
200 bytes actual size

First 2 buffers are full, and in the last one only 200 bytes are filled.

So , actual size of memPool _t is 2248 bytes.
Actual size of first 2 buffers is 1024 bytes, and actual size of last buffer is 200 bytes.

If now user wants to write into memPool _t object of 1000 bytes length, then memPool _t has to:

- add 824 bytes to last buffer (till 1024 bytes),
- to create a new buffer of size 1024 bytes
- to write into it 176 bytes.

So , it's possible that when we write into Memory Page integer number then the first 3 bytes of it are in one page and the last byte in another page !!!

"Holes" in the pages are forbidden – nothing can't be written over in a byte which address in Page is larger than Actual Size

Notes:

For sure ANY data can be written and read to/from Memory Pool, for example:

```
memPool_t mp;
int k;
mp.write(&k , sizeof(int));             // from Current position
mp.write(&k , sizeof(int), int pos);    // from Given position
```

^^

STL (Standard Template Library) Vector example

```
#include <iostream>
#include <vector>
using namespace std;

int main() {

        vector<Page_t*> v;              // CTOR of vector of pointers to page

        Page_t* pg = new Page_t;
        v.insert(v.end(), pg);          // add new pointer to page to the end of vector ?
}
```