Subject: Re: Quick bi-array
Posted by Mindtraveller on Wed, 21 Jan 2009 17:55:36 GMT
View Forum Message <> Reply to Message

I`ve tried to test BiVector<Element *> and it looks like it was not the thing I wanted.
What did I want? I wanted a deque-like container which is extremely fast on adding and removing it`s records. This means no allocation/deallocation is accepted (just calling Element::operator= only). The idea of such a container is as quick as possible container access while program runs. Contrary, possible allocation/deallocation on program start/finish is acceptable.
So the thing I wanted is such code

```
 struct Element
 {
  Element()       :a(0)  {Cout()<<"*";}
  Element(int _a) :a(_a) {Cout()<<"+";}
  ~Element()             {Cout()<<".";}
  Element & operator= (const Element&op) {a=op.a; Cout()<<"="; return *this;}
  int a;
 };
 QuickDeque<Element> vec;
 static Element elm(0);
 for (int i=0;i<10;++i)
 {
  elm.a = i;
  vec.AddTail(elm);
 }
```

should give output:
Quote:==========
without any constructors/desctructors after program started.
And it looks like I managed to do something like it (just a second quick try after BiVector test):

```
template<class T> class QuickDeque
{
public:
 QuickDeque(int cap = 10) :capacity(0), data(NULL), start(0), count(0) {ASSERT(cap > 0);
AddAlloc(cap);}
 ~QuickDeque() {DeAlloc(); }

 void AddTail(const T&t) {if (count >= capacity) AddAlloc(count); int offs=start+count; if (offs >=
capacity) offs-=capacity; *((T *) &data[offs*sizeof(T)]) = t; ++count;}
 void DropHead(T &t) {ASSERT(count > 0); t = *((T *) &data[start*sizeof(T)]); if (++start ==
capacity) start=0; --count;}
 T &operator[] (int n) {ASSERT(n>=0 && n<count); int offs=start+n; if (offs >= capacity)
offs-=capacity; return *((T *) &data[offs*sizeof(T)]);}
 int GetCount() {return count;}

private:
 void AddAlloc(int capacityAdd)
 {
  ASSERT(capacityAdd >= 0);
```

```
  int capacityNew = capacity+capacityAdd;
  byte *newData = new byte[capacityNew*sizeof(T)];
  memcpy(&newData[0], &data[start*sizeof(T)], (capacity-start)*sizeof(T));
  memcpy(&newData[(capacity-start)*sizeof(T)], &data[0], start*sizeof(T));
  for (int i=count; i<capacityNew; ++i)
   new (&newData[i*sizeof(T)]) T();

  if (data)
   delete[] data;
  start    = 0;
  data     = newData;
  capacity = capacityNew;
 }

 void DeAlloc()
 {
  for (int i=0; i<capacity; ++i)
   ((T *) &data[i*sizeof(T)])->T::~T();
  if (data)
   delete[] data;
 }

 int   capacity;
 byte *data;
 int   start,
       count;
};
```

I wonder if it works with polymorphic elements...