Subject: Initialization for Buffer<T>
Posted by Mindtraveller on Sat, 29 Aug 2015 16:41:29 GMT
View Forum Message <> Reply to Message

Buffer<T> is known as replacement for C-style arrays. So I guess it needs some way to fill it as easy as possible (like we initialize arrays in C). So here is my proposal with little additions:

```
template <class T>
class Buffer : Moveable< Buffer<T> > {
 size_t size;
 mutable T *ptr;
 int insertI;

public:
 operator T*()                    { return ptr; }
 operator const T*() const          { return ptr; }
 T *operator~()                    { return ptr; }
 const T *operator~() const          { return ptr; }

 Buffer<T> & Alloc(size_t _size)            { Clear(); ptr = new T[_size]; size = _size; return *this; }
 Buffer<T> & Alloc(size_t _size, const T& in) { Clear(); ptr = new T[_size]; size = _size; Fill(ptr, ptr
+ size, in); return *this; }

 void Clear()                      { if(ptr) delete[] ptr; ptr = NULL; insertI = 0; size = 0; }
 size_t GetCount()                  { return size; }

 Buffer()                          { Init(); ptr = NULL; }
 Buffer(size_t size)                { Init(); ptr = new T[size]; }
 Buffer(size_t size, const T& init)   { Init(); ptr = new T[size]; Fill(ptr, ptr + size, init); }
 ~Buffer()                         { if(ptr) delete[] ptr; }

 void Init()                       { size = 0; insertI = 0; }
 void operator=(Buffer rval_ v)      { if(ptr) delete[] ptr; ptr = v.ptr; v.ptr = NULL; }
 Buffer<T> & operator<< (const T &in){ ptr[insertI++] = in; return *this; }
 Buffer(Buffer rval_ v)             { ptr = v.ptr; v.ptr = NULL; size = v.size; v.size=0; insertI = 0; }
};

// Usage:
// buffer.Alloc(3) << v1 << v2 << v3;
```

Subject: Re: Initialization for Buffer<T>
Posted by mirek on Sun, 30 Aug 2015 05:38:19 GMT
View Forum Message <> Reply to Message

Mindtraveller wrote on Sat, 29 August 2015 18:41Buffer<T> is known as replacement for C-style arrays. So I guess it needs some way to fill it as easy as possible (like we initialize arrays in C). So here is my proposal with little additions:

```
template <class T>
class Buffer : Moveable< Buffer<T> > {
 size_t size;
 mutable T *ptr;
 int insertI;

public:
 operator T*()                       { return ptr; }
 operator const T*() const           { return ptr; }
 T *operator~()                      { return ptr; }
 const T *operator~() const          { return ptr; }

 Buffer<T> & Alloc(size_t _size)             { Clear(); ptr = new T[_size]; size = _size; return *this; }
 Buffer<T> & Alloc(size_t _size, const T& in) { Clear(); ptr = new T[_size]; size = _size; Fill(ptr, ptr
+ size, in); return *this; }

 void Clear()                        { if(ptr) delete[] ptr; ptr = NULL; insertI = 0; size = 0; }
 size_t GetCount()                   { return size; }

 Buffer()                            { Init(); ptr = NULL; }
 Buffer(size_t size)                 { Init(); ptr = new T[size]; }
 Buffer(size_t size, const T& init)  { Init(); ptr = new T[size]; Fill(ptr, ptr + size, init); }
 ~Buffer()                           { if(ptr) delete[] ptr; }

 void Init()                         { size = 0; insertI = 0; }
 void operator=(Buffer rval_ v)      { if(ptr) delete[] ptr; ptr = v.ptr; v.ptr = NULL; }
 Buffer<T> & operator<< (const T &in){ ptr[insertI++] = in; return *this; }
 Buffer(Buffer rval_ v)              { ptr = v.ptr; v.ptr = NULL; size = v.size; v.size=0; insertI = 0; }
};

// Usage:
// buffer.Alloc(3) << v1 << v2 << v3;
```

Adding 2 new member variable seems quite wasteful.

However, I am now in process of adding C++11 std::initializer_list to all containers. Without your post, I would probably forgot about Buffer. With C++11, we can now have

```
Buffer<int> x{ v1, v2, v3 };

x = { v4, v5, v6 };
```

Subject: Re: Initialization for Buffer<T>
Posted by Mindtraveller on Sun, 30 Aug 2015 09:10:46 GMT
View Forum Message <> Reply to Message

This is of corse more convenient than "classic" U++ approach with operator<<(), but what should one do if he's developing i.e. for embedded system with no C++11 support in compiler?

Subject: Re: Initialization for Buffer<T>
Posted by mirek on Sun, 30 Aug 2015 09:20:38 GMT
View Forum Message <> Reply to Message

Mindtraveller wrote on Sun, 30 August 2015 11:10This is of corse more convenient than "classic" U++ approach with operator<<(), but what should one do if he's developing i.e. for embedded system with no C++11 support in compiler?

Actually, the only problem for C++11 was windows. It is now solved (with VS2015).

Second problem can be some old linux distro - still can be solved by compiling new GCC on it.

Embedded systems would worry me least - all of them are using GCC, so they are C++11 for quite a long time now...

For upcoming release, C++11 will not be required, but supported. I think this issue is exactly what should be resolved with C++11 features intended for it... :)

Mirek

Subject: Re: Initialization for Buffer<T>
Posted by Mindtraveller on Tue, 06 Oct 2015 10:30:01 GMT
View Forum Message <> Reply to Message

After some experience with Buffer, I still propose adding GetCount() member. It would be convenient.

Subject: Re: Initialization for Buffer<T>
Posted by mirek on Thu, 08 Oct 2015 07:19:52 GMT
View Forum Message <> Reply to Message

If you need GetCount, why not use Vector?

Subject: Re: Initialization for Buffer<T>
Posted by Mindtraveller on Thu, 08 Oct 2015 12:01:00 GMT

View Forum Message <> Reply to Message

I can not guarantee that elements are situated in Vector just like in c-style array. Event if it current;y implemented this way, noone will guarantee it will be implemented this way in the future. So I thought you implemented Buffer specially for cases we need c-style array.

---

Subject: Re: Initialization for Buffer<T>
Posted by mirek on Fri, 09 Oct 2015 15:34:33 GMT
View Forum Message <> Reply to Message

Actually, it is guaranteed.

typedef T *Iterator

Iterator type. Iterator is guaranteed to be of T* type.

operator T*()
Returns non-constant pointer to elements.

operator const T*() const
Returns constant pointer to elements.

I agree that more clearly worded guarantee should be given here (this is sort of implicit). Updating docs now...

Buffer indeed is replacement of situation

byte *buffer = new byte[N];
...
delete[] buffer;

but adding GetCount() would mean it is less efficient that this raw code. In the situation when you need GetCount, overhead of Vector is so small that you should perhaps use it instead (it only stores 'alloc' in addition to 'count').

---

Subject: Re: Initialization for Buffer<T>
Posted by Mindtraveller on Sat, 10 Oct 2015 20:26:25 GMT
View Forum Message <> Reply to Message

Thank you for detailed answer. Still after these years of using U++, I suppose I have to start making sime kind of guide or manual for U++. Because there are so many features and factors which are almsot undocumented. It would be good to have them documented. I see it as more structured version of your Help Tutorials (Core, GUI, etc.)

Subject: Re: Initialization for Buffer<T>
Posted by mirek on Sun, 11 Oct 2015 10:56:55 GMT
View Forum Message <> Reply to Message

Mindtraveller wrote on Sat, 10 October 2015 22:26Thank you for detailed answer. Still after these years of using U++, I suppose I have to start making sime kind of guide or manual for U++. Because there are so many features and factors which are almsot undocumented. It would be good to have them documented. I see it as more structured version of your Help Tutorials (Core, GUI, etc.)

Well, my fault. While I think reference docs is quite complete these days, some basic ideas are hidden in details. Plus, some issues are evolving over time, so some of docs can be misleading these days...