Subject: Recent Ubuntu8.04 troubles confirmed to be the compiler bug
Posted by mirek on Wed, 14 May 2008 11:14:47 GMT
View Forum Message <> Reply to Message

I was hunting this one for about 12 hours total, tracked the problem down, seen the broken assembly and prepared the isolated test.

This test should print "32", which is the sizeof(Item). But if you compile it with -O3 flag under

gcc (GCC) 4.2.3 (Ubuntu 4.2.3-2ubuntu7)

it prints 0.

Please check, maybe there is still something wrong with code, some undefined behaviour:

CGGBug.h

```
#include <stdio.h>
#include <string.h>
#include <stdlib.h>
#include <new>

template <class T> inline const T& my_max(const T& a, const T& b) { return a > b ? a : b; }
template <class T> inline const T& my_min(const T& a, const T& b) { return a < b ? a : b; }

template <class T>
inline T minmax(T x, T _min, T _max) { return my_min(my_max(x, _min), _max); }

void *MemoryAllocSz(size_t& sz);
void  MemoryFree(void *);

template <class T>
class Vector {
 T       *vector;
 int      items;
 int      alloc;

 static void RawFree(T *ptr)           { if(ptr) MemoryFree(ptr); }
 static T   *RawAlloc(int& n);

 void RawInsert(int q, int count);


public:
 void  InsertN(int q, int count);
 const T& First() { return vector[0]; }
 int   GetCount() const { return items; }
```

```
 Vector() { vector = NULL; items = alloc = 0; }
};

template <class T>
T * Vector<T>::RawAlloc(int& n)
{
 size_t sz0 = n * sizeof(T);
 size_t sz = sz0;
 void *q = MemoryAllocSz(sz);
 n += (int)((sz - sz0) / sizeof(T));
 return (T *)q;
}

template <class T>
void Vector<T>::RawInsert(int q, int count)
{
 if(!count) return;
 if(items + count > alloc) {
  T *newvector = RawAlloc(alloc = alloc + my_max(alloc, count));
  if(vector) {
   memcpy(newvector, vector, q * sizeof(T));
   memcpy(newvector + q + count, vector + q, (items - q) * sizeof(T));
   RawFree(vector);
  }
  vector = newvector;
 }
 else {
  memmove(vector + q + count, vector + q, (items - q) * sizeof(T));
 }
 items += count;
}

template <class T>
void Vector<T>::InsertN(int q, int count)
{
 RawInsert(q, count);
}

void *MemoryAllocSz(size_t& sz);
void  MemoryFree(void *);

struct Item {
 char h[32];
};

struct Bar {
 Vector<Item> li;
```

```
 void DoTest(int i, int count);
};
```

GCCBug1.cpp

```
#include "GCCBug.h"

char array[256];

void *MemoryAllocSz(size_t& sz)
{
 printf("%d\n", sz);
 return array;
}

void MemoryFree(void *) {}
```

GCCBug2.cpp

```
#include "GCCBug.h"

void Bar::DoTest(int i, int count)
{
 li.InsertN(minmax(i, 0, li.GetCount()), my_max(count, 0));
}
```

GCCBug3.cpp

```
#include "GCCBug.h"

int main(int argc, char argv[])
{
 Bar b;
 b.DoTest(0, 1);
 return 0;
}
```

(Note, it is probably important to keep all this in 4 specific files to avoid specific inlining variants).

Compiled with flags:

-ggdb -g2 -fexceptions -O3 -x c++

Mirek