# CoCoALib - Design #934

## MachineInt: change semantics?

30 Sep 2016 11:28 - John Abbott

| | | | |
|---|---|---|---|
| **Status:** | In Progress | **Start date:** | 30 Sep 2016 |
| **Priority:** | Normal | **Due date:** | |
| **Assignee:** | | **% Done:** | 30% |
| **Category:** | Safety | **Estimated time:** | 3.00 hours |
| **Target version:** | CoCoALib-0.99880 | **Spent time:** | 3.35 hours |

**Description**

The implementation of MachineInt is a bit messy as it tries to guarantee that no information will be lost (*e.g.* all values from the most negative long int to the most positive unsigned long int can be represented).

Would it be reasonable to change MachineInt so that it is just a "safe" version of long int?

This would simplify the implementation (since an extra bit is not needed).

The idea is that a value of type unsigned long int which is too big to fit into (signed) long int would automatically throw an exception.

Discuss!

**Related issues:**

| | | |
|---|---|---|
| Related to CoCoALib - Design #925: MachineInt or long for args which are indi... | **In Progress** | **20 Sep 2016** |
| Related to CoCoALib - Design #581: C++14: MachineInt | **Closed** | **04 Jul 2014** |
| Related to CoCoALib - Support #1666: MachineInt: chase through ULL changes | **In Progress** | **16 Feb 2022** |

### History

#### #1 - 30 Sep 2016 11:28 - John Abbott

*- Related to Design #925: MachineInt or long for args which are indices (yet again) added*

#### #2 - 30 Sep 2016 11:38 - John Abbott

Advantages:

- One hope is that the simpler implementation may allow the compiler to be clever (and optimise away almost all conversions).
- As already stated the impl should become simpler.

Disadvantages:

- changing the semantics is not backward-compatible (but who would notice?).
- the only case I can think of where a user might notice "reduced capability" is automatically converting an unsigned long int (whose value is too large to fit into a signed long int) into a RingElem.

I recall that Scott Meyers advised against using unsigned integral types (because they can produce unexpected behaviour), and that we have decided to adopt this viewpoint. In view of this, I think that the disadvantage is minor (and would "punish" only those who disregard to advice not to use unsigned types).

#### #3 - 30 Sep 2016 13:46 - John Abbott

I implemented a prototype, and ran a speed test. The results are mixed: usefully faster than current MachineInt, but disappointingly slow compared to

using long directly.

The test used the new type for indices into an ad-hoc class IdMat (for identity matrix). There was just a double loop to compute the sum of the entries of the matrix (in ZZ/(101)).

#### #4 - 30 Sep 2016 14:12 - John Abbott

For the record, here are the timings I obtained (on the Linux tower in my office):
built-in IdentityMat 1.66s
IdMat with long indices 0.84s
IdMat with MachineInt indices 0.95s
IdMat with new-style MachineInt indices 0.86

Why is built-in IdentityMat so much slower? Presumably because of a virt-mem-fn dispatch.

**NOTE** very mysterious: for the built-in matrix I used operator() which does an arg check then calls mem fn myEntry; out of curiosity I tried calling the mem-fn myEntry directly and the loop time **increased** to 2.27s. How is that possible?

**NOTE2** repeated runs exhibit considerable variability in the timings (but the case of using long gives stable times). The machine should be unloaded, and dedicated to me. Very odd!

#### #5 - 30 Sep 2016 14:21 - John Abbott

Here is the test code I have been using: (functional rather than elegant)

```
class INDEX
{
public:
  INDEX(long n);
//  operator long() { return value; }
//private:
  long value;
  friend long CheckIndexRange(const INDEX& i, long n, const char* const FnName);
};

INDEX::INDEX(long n):
    value(n)
{
  if (n < 0) CoCoA_ERROR(ERR::NotNonNegative, "INDEX ctor");
}

long CheckIndexRange(const INDEX& i, long n, const char* const FnName)
{
  if (i.value >= n) CoCoA_ERROR(ERR::BadIndex, FnName);
  return i.value;
}

long CheckIndexRange(const MachineInt& i, long n, const char* const FnName)
{
  if (IsNegative(i) || !IsSignedLong(i) || AsSignedLong(i) >= n) CoCoA_ERROR(ERR::BadIndex, FnName);
  return AsSignedLong(i);
}

long CheckIndexRange(long i, long n, const char* const FnName)
{
  if (i >= n) CoCoA_ERROR(ERR::BadIndex, FnName);
  return i;
}
```

```
class IdMat
{
public:
  IdMat(const ring& R, INDEX n): myN(n.value), myR(R) {}
  ConstRefRingElem myEntry(const INDEX& i, const INDEX& j) { CheckIndexRange(i, myN, "IdMat::myEntry");CheckIn
dexRange(j, myN, "IdMat::myEntry"); if (i.value==j.value) return one(myR); else return zero(myR); }
  ConstRefRingElem myEntry1(const MachineInt& i, const MachineInt& j) { CheckIndexRange(i, myN, "IdMat::myEntr
y");CheckIndexRange(j, myN, "IdMat::myEntry"); if (AsSignedLong(i)==AsSignedLong(j)) return one(myR); else ret
urn zero(myR); }
  ConstRefRingElem myEntry2(long i, long j) { if (i < 0 || j < 0) CoCoA_ERROR(ERR::BadIndex, "myEntry2"); Chec
kIndexRange(i, myN, "IdMat::myEntry");CheckIndexRange(j, myN, "IdMat::myEntry"); if (i==j) return one(myR); el
se return zero(myR); }
private:
  long myN;
  const ring& myR;
};

void program()
{
  GlobalManager CoCoAFoundations;

  ring R = NewZZmod(101);
  const long N = 10000;
  ConstMatrix I = IdentityMat(R,N);

  const int r = N;
  const int c = N;
  RingElem sum(R);
  double t0 = CpuTime();
  for (int i=0; i < r; ++i)
    for (int j=0; j < c; ++j)
    {
      sum += I(i,j);
//      sum += I->myEntry(i,j);
    }
  double t1 = CpuTime();
  cout << "using CoCoALib's IdentityMat:" << endl;
  cout << "sum = " << sum << endl;
  cout << "Loop time " << t1-t0 << endl << endl;

  IdMat J(R,10000);
  RingElem sum2(R);
  double t2 = CpuTime();
  for (int i=0; i < r; ++i)
    for (int j=0; j < c; ++j)
    {
      sum2 += J.myEntry(i,j);
    }
  double t3 = CpuTime();
  cout << "Using ad hoc class INDEX:" << endl;
  cout << "sum2 = " << sum2 << endl;
  cout << "Loop time " << t3-t2 << endl << endl;

  RingElem sum4(R);
  double t6 = CpuTime();
  for (int i=0; i < r; ++i)
    for (int j=0; j < c; ++j)
    {
      sum4 += J.myEntry1(i,j);
    }
  double t7 = CpuTime();
  cout << "Using MachineInt:" << endl;
  cout << "sum4 = " << sum4 << endl;
  cout << "Loop time " << t7-t6 << endl << endl;

  RingElem sum3(R);
  double t4 = CpuTime();
  for (int i=0; i < r; ++i)
    for (int j=0; j < c; ++j)
    {
      sum3 += J.myEntry2(i,j);
    }
  double t5 = CpuTime();
  cout << "using long:" << endl;
```

```
  cout << "sum3 = " << sum3 << endl;
  cout << "Loop time " << t5-t4 << endl;
}
```

#### #6 - 30 Sep 2016 15:01 - John Abbott

In the example code above, initially I used the name index for the class (rather than INDEX) but that produced strange compilation problems. A WWW search revealed that index is a special extension in gcc (which can be disabled by setting some compilation flags).

#### #7 - 30 Sep 2016 22:11 - John Abbott

I tried on my old MacBook with g++-4.2, and the results were disappointing. The new class INDEX was significantly slower than MachineInt; I have absolutely no idea why. Anyway, since it is such an old platform, it's probably best just to ignore the anomaly.

On the linux VM in the MacBook, results were more sane: INDEX and MachineInt were about the same speed, and not too much slower than long. On one run INDEX was slightly faster than long!?!

#### #8 - 02 Mar 2017 14:50 - Anna Maria Bigatti

*- % Done changed from 0 to 20*

*- Estimated time set to 3.00 h*

After personal discussion in Kassel: I agree.

#### #9 - 05 Apr 2019 16:06 - John Abbott

*- Related to Design #581: C++14: MachineInt added*

#### #10 - 02 Feb 2021 17:04 - John Abbott

*- Status changed from New to In Progress*

*- Target version changed from CoCoALib-1.0 to CoCoALib-0.99850*

*- % Done changed from 20 to 30*

There is a second impl of MachineInt in the current sources. The impl was not complete (but it is now).

However many tests failed: it seems that we do use unsigned long in several places.
One particular failure was in test-convert. With the version of MachineInt which accepts only signed long values,
there is a problem (I think) with testing ul == N where ul is a large value of type unsigned long.

Not sure what to do :-/

#### #11 - 16 Mar 2024 21:25 - John Abbott

*- Target version changed from CoCoALib-0.99850 to CoCoALib-0.99880*

#### #12 - 15 Apr 2024 10:06 - John Abbott

*- Related to Support #1666: MachineInt: chase through ULL changes added*