incalang – an overview

a few interesting remarks about the
GPL language incalang

overview

incalang at a glance – the language (1)

- Syntax resembles **C++**

```
float g( float y, const float *z ) {
    float a = 0;
    for( int x = 0; x < 10; x++ )
        a += f( y * z[ x ], x );
    return a;
}
```

- Supports **complex data types**

```
void f() {
    String a = “Hello “;
    String b = “World”;
    g( a + left( b, 3 ) ); // called with “Hello Wor”
}
```

- **Stack-bound instances** of objects

```
class A {
public:
    void create() { /* constructor */ }
    void destroy() { /* destructor */ }
}

void f() {
    A a; // stack bound instance of A
}
```

- **Multiple inheritance, virtual functions, references**

```
class A extends B, C {
public:
    void m( int i ); // overload
}

void f( B &b, int i ) {
    b.m( i ); // calls overloaded m if b is of type A
}
```

- Compiler with **intelligent look-ahead**

```
void f() {
    MyObject a; // okay, since defined below
    g( a );
}

class MyObject {
    int x, y, z;
}
```

incalang at a glance – the language (2)

- **Function overloading** by **parameter types**

```
void f( int x, int y ) { // f1
}

void f( float x, float y ) { // f2
}

void f( double x, double y = 1.25 ) { // f3
}

void g() {
    f( 1, 2 ); // will call f1
    f( 1.5, 3.5 ); // will call f2
    f( (double)0.3 ); // will call f3
}
```

- **Operator overloading** for all binary and unary operators

```
class A {
public:
    A& operator*( A &a ) { ...  }
}

void f() {
    A a, b;
    f( a * b ); // calls operator* function
}
```

- **Templates** for functions and classes with **explicit** and **implicit** creation of the type instance

```
template<T> class A {
    T x; // a member variable of type T
}

void f() {
    A<String> a; // explicit instancing
    g( a );
}

template<T> swap( T &a, T &b ) {
    T c = a;
    a = b;
    b = c;
}

void h( int x ) {
    int y = 2;
    swap( x, y ); // implicit instancing
}
```

incalang at a glance – the editor

- **Automatic formatting** of **source code** while writing

```
void f(int*x, int y) {    □   void f( int *x, int y ) {
g(3*y,5+*x);                      g( 3 * y, 5 + *x );
}                             }
if((a.x(&y)+7)&3)         □   if( ( a.x( &y ) + 7 ) & 3 )
class A {                     class A {
public:                   □   public:
void f(int x);                    void f( int x );
}                             }
```

- **Syntax coloring**

- **Collapsing** of **blocks** with a simple key command (shift-tab)

```
void f( int x ) {         □   void f( int *x, int y ) { ... }
    int y = 0;                n( 1, 2 );
    y += g( y, 3 );
    y += g( y, 10 );
    return y;
}
n( 1, 2 );
if( a == 3 ) {            □   if( a == 3 ) { ... }
    b += 1;                   a += 7;
    m( a, b );
}
a += 7;
```

- **Copy & paste**

- **Find & replace**

- **Built-in debugger with breakpoints, views on variables and stack**

incalang at a glance – the libraries

A few examples of functions, that are available in incalang by default:

- **String** data type, providing the functions known from **BASIC**:

  ```
  bool f( String a, String b ) {
      return left( a, 3 ) >= mid( b, 5 );
  }
  ```

- Functions for **input and output** of text via the **console**

- **Arrays** for several data types for **inserting** and **deleting** elements, for doing binary und linear **searches** and fast **sorting**

- Extensive mathematical functions, among others **trigonometric functions**, bit manipulation, minimum and maximum, **interpolation** (linear, kubisch, u.a.)

- **Point**, **Color**, und **Vector** data types, which allow for expressions that resemble **renderman** syntax:

  ```
  float f( Point p, Vector v ) {
      p += normalize( v ) * 0.7;
      return xcomp( p );
  }
  ```

- Functions to get **perlin noise** (noise, snoise)

- **BigInt** data type to do calculations with arbitrarily large integer numbers. Allows for multiplication, division and comparison of numbers, as well as calculation of square roots, powers, und **powers in modulo groups**

- **General stream architecture** (InputStream, OutputStream). access on data streams in files or in memory using the **File** und **MemoryBuffer** classes. compression and decompression of data using the classes **InflaterStream** and **DeflaterStream**

- Ability to call open **OpenGL** functions. Furthermore, additional **SGL library** (Simple Graphics Layer) that allows for **simple drawing** of twodimensional content and especially **text** on top of OpenGL

- **Thread library** that meets **soft real time criteria**, consisting of the classes **Thread**, **Mutex** and **Queue**

incalang at a glance – the implementation

- **Extremely fast compiler** and linker by using several mechanisms of memory management and optimization

- Forward resolution of class names that are defined late by using a complex, eight-level resolution algorithm in a dependency graph

- Transformation of the program code into a **virtual machine code** called **IMC** (Incalang Machine Code):

```
# function generateRay;f;f;Ray::&
[ locals size 32 ]
0000  ldi.q #52
0003  pushself
0004  add.q
0005  xload 16
0007  ldi.q #20
000a  pushlocal -28
000c  load.q
000d  add.q
000e  xstore 16
0010  pop 16
0012  ldi.q #4
0015  pushself
0016  add.q
0017  ldi.q #20
001a  pushself
001b  add.q
001c  pushlocal -24
001e  fload.s
001f  csf root::operator*;f;.Point::&
0022  pushstack -16
0024  xload 16
```

- **IMC** is a bytecode for a stack machine

- The **IMC** code is executed by a **separate interpreter process**, crashing your program will not crash the IDE

- Incalang implements tail call optimizations on the **IMC** level

- Incalang combines a number of approaches and ideas that not have been combined that way before

incalang at a glance – sample programs

- **Lenstra.Inca** implementes **Lenstra's algorithm using elliptic curves to factor large integer numbers** into their prime factors. This sample uses the BigInt data type to do calculations with large numbers and in modulo groups.

- **Raytracer.Inca** implements a very simple raytracer that generated the image of the earth. It uses a shader by **Ken Musgrave.** The program shows how to use the point, vector and color data types offered in incalang, as well as the other renderman-like structures.

- **ERModeller.Inca** is a simple entity-relationship-modeller that has been written for a real small database project. It is capable of handling rather large diagrams and has a nice interface.

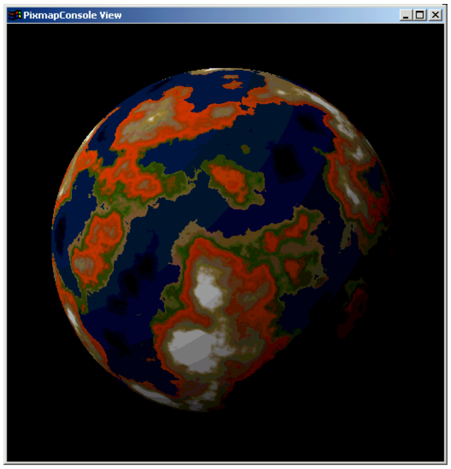
- **Water.Inca** shows how to use **OpenGL** in incalang. It shows a pool of water in which drops fall. **Level.Inca** is also an OpenGL demonstration that let's you roam through a three-dimensional labyrinth.

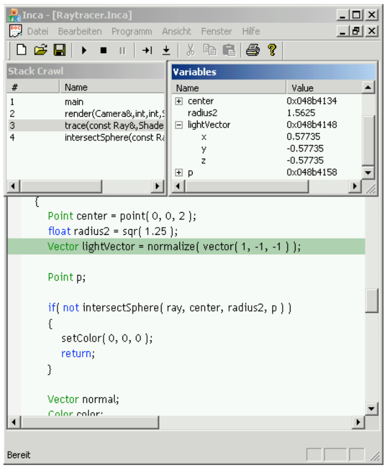
- **CallFromDll.Inca** shows how to call C-functions from incalang. Basically, every function that is provided via a DLL can be called from incalang.

- **StreamSample.Inca** shows how to use streams, how to compress and decompress data, how to read and write from files, and how to use automatically growing memory allocations.

- **SimpleThreads.Inca** shows how to use the incalang thread library which meets soft real time requirements and let's you control scheduling times up to a level of microseconds. incalang's thread library does not use Windows threads.

incalang at a glance – screenshots (1)

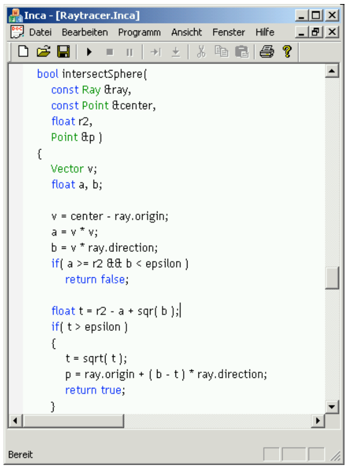




#	Name
1	main
2	render(Camera&,int,int,S
3	trace(const Ray&,Shade
4	intersectSphere(const Ra

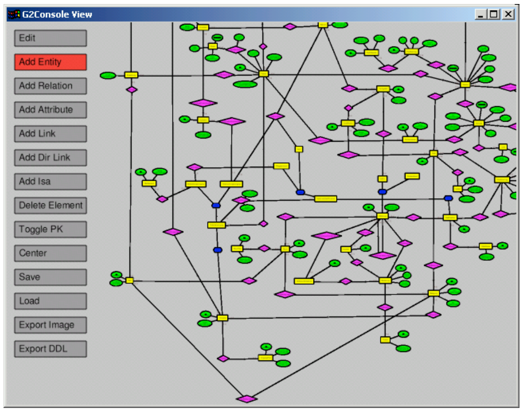
Name	Value
⊞ center	0x048b4134
radius2	1.5625
⊟ lightVector	0x048b4148
x	0.57735
y	-0.57735
z	-0.57735
⊞ p	0x048b4158

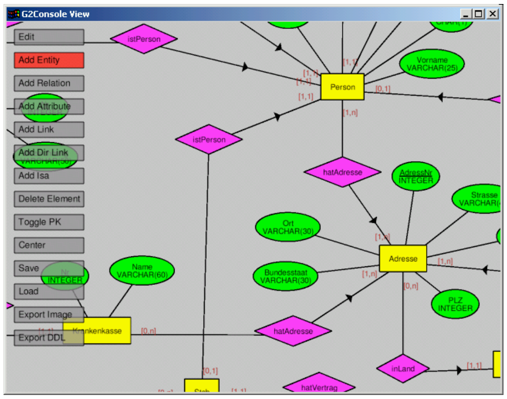
```
{
   Point center = point( 0, 0, 2 );
   float radius2 = sqr( 1.25 );
   Vector lightVector = normalize( vector( 1, -1, -1 ) );

   Point p;

   if( not intersectSphere( ray, center, radius2, p ) )
   {
      setColor( 0, 0, 0 );
      return;
   }

   Vector normal;
   Color color;
```

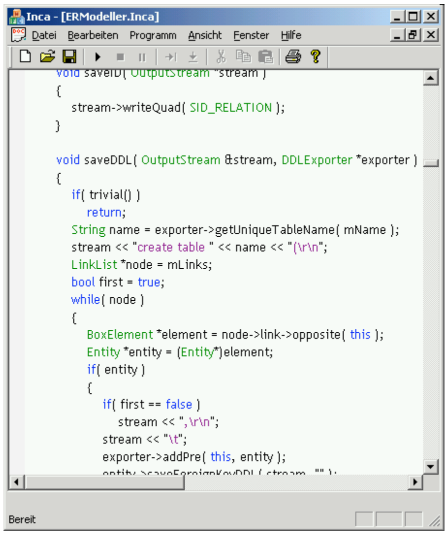


```
bool intersectSphere(
   const Ray &ray,
   const Point &center,
   float r2,
   Point &p )
{
   Vector v;
   float a, b;

   v = center - ray.origin;
   a = v * v;
   b = v * ray.direction;
   if( a >= r2 && b < epsilon )
      return false;

   float t = r2 - a + sqr( b );
   if( t > epsilon )
   {
      t = sqrt( t );
      p = ray.origin + ( b - t ) * ray.direction;
      return true;
   }
```

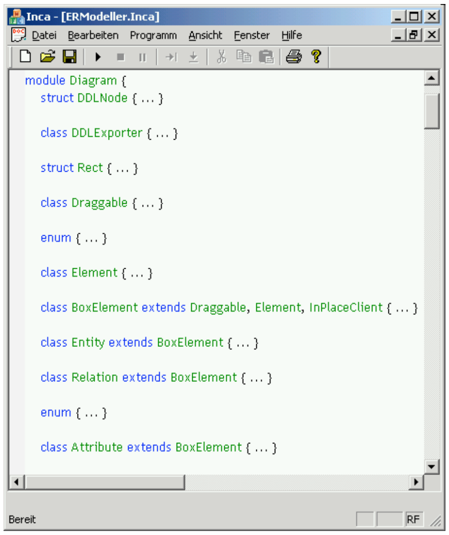
Screenshots of the sample raytracer: rendered image (left), debug view (middle), source code view (right)







```
void saveID( OutputStream *stream )
{
   stream->writeQuad( SID_RELATION );
}

void saveDDL( OutputStream &stream, DDLExporter *exporter )
{
   if( trivial() )
      return;
   String name = exporter->getUniqueTableName( mName );
   stream << "create table " << name << "(\r\n";
   LinkList *node = mLinks;
   bool first = true;
   while( node )
   {
      BoxElement *element = node->link->opposite( this );
      Entity *entity = (Entity*)element;
      if( entity )
      {
         if( first == false )
            stream << ",\r\n";
         stream << "\t";
         exporter->addPre( this, entity );
         entity->saveForeignKeyDDL( stream, "" );
```

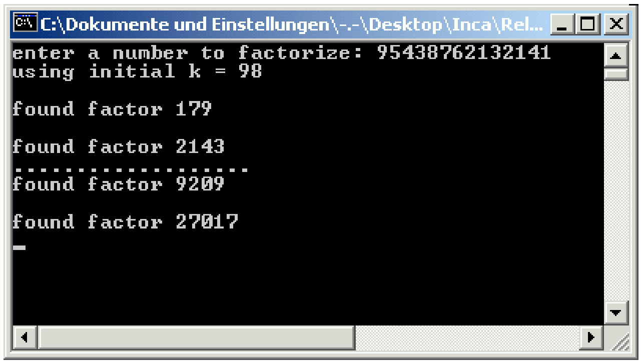


```
module Diagram {
   struct DDLNode { ... }

   class DDLExporter { ... }

   struct Rect { ... }

   class Draggable { ... }

   enum { ... }

   class Element { ... }

   class BoxElement extends Draggable, Element, InPlaceClient { ... }

   class Entity extends BoxElement { ... }

   class Relation extends BoxElement { ... }

   enum { ... }

   class Attribute extends BoxElement { ... }
```

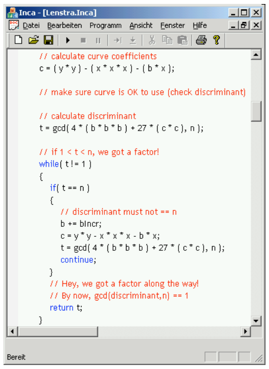
Screenshots of the sample ER modeler: editor view (top left), editor view detail (top right), source code view (bottom left and right)

incalang at a glance – screenshots (2)

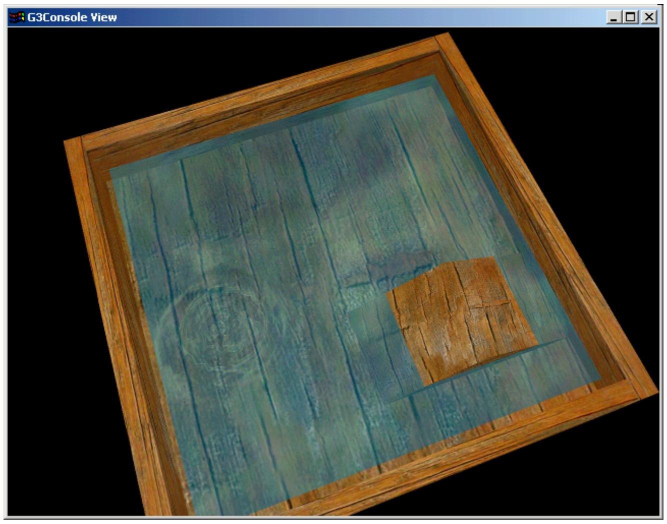
Screenshots of Inca’s sample program “Lenstra” that uses Lenstra’s elliptic curve method to factor numbers: program input and output (left) and source code view (right)

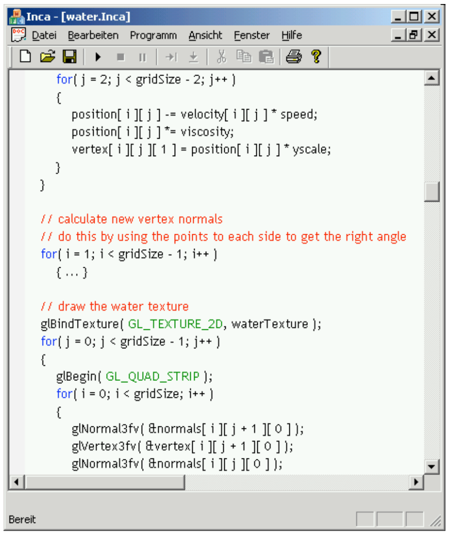


```
enter a number to factorize: 95438762132141
using initial k = 98

found factor 179

found factor 2143
.................
found factor 9209

found factor 27017
_
```



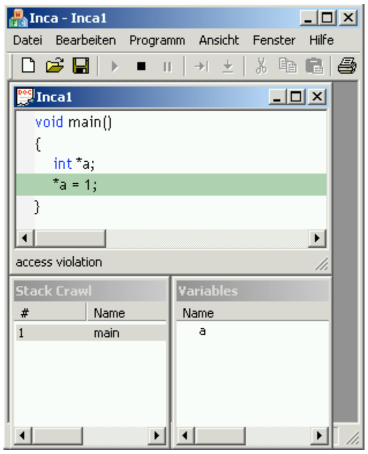
```
// calculate curve coefficients
c = ( y * y ) - ( x * x * x ) - ( b * x );

// make sure curve is OK to use (check discriminant)

// calculate discriminant
t = gcd( 4 * ( b * b * b ) + 27 * ( c * c ), n );

// if 1 < t < n, we got a factor!
while( t != 1 )
{
   if( t == n )
   {
      // discriminant must not == n
      b += bIncr;
      c = y * y - x * x * x - b * x;
      t = gcd( 4 * ( b * b * b ) + 27 * ( c * c ), n );
      continue;
   }
   // Hey, we got a factor along the way!
   // By now, gcd(discriminant,n) == 1
   return t;
}
```





```
      for( j = 2; j < gridSize - 2; j++ )
      {
         position[ i ][ j ] -= velocity[ i ][ j ] * speed;
         position[ i ][ j ] *= viscosity;
         vertex[ i ][ j ][ 1 ] = position[ i ][ j ] * yscale;
      }
   }

   // calculate new vertex normals
   // do this by using the points to each side to get the right angle
   for( i = 1; i < gridSize - 1; i++ )
      { ... }

   // draw the water texture
   glBindTexture( GL_TEXTURE_2D, waterTexture );
   for( j = 0; j < gridSize - 1; j++ )
   {
      glBegin( GL_QUAD_STRIP );
      for( i = 0; i < gridSize; i++ )
      {
         glNormal3fv( &normals[ i ][ j + 1 ][ 0 ] );
         glVertex3fv( &vertex[ i ][ j + 1 ][ 0 ] );
         glNormal3fv( &normals[ i ][ j ][ 0 ] );
```

Screenshots of the “Water” OpenGL sample program: display (left) and source code view (right)

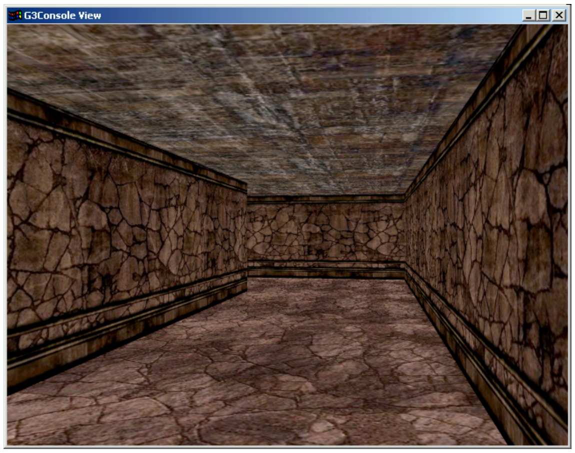


```
void main()
{
   int *a;
   *a = 1;
}
```

access violation

#	Name
1	main

Name
a



Screenshots of Inca catching a bus error in a compiled application (left) and of the display of the “Level” OpenGL sample program (right)

The given screenshots were made on a standard installation of Inca on Windows 2000.

incalang at a glance – applications

incalang has been released under the GPL, its source code is freely available. The following ways of using it might be possible:

1. Adapt it for special usage – incalang as **module**

    - incalang can be used as scripting language
    - incalang might be used as language to interface to complex systems or to write plug-ins for complex systems
    - incalang as programming language and the libraries can be changed in any needed way

2. Platform for academic experiments – incalang as **platform**

    - incalang is a working fully functioning compiler that doesn't much of the capabilities that C++ offers
    - incalang is much smaller than, say, gcc
    - changes to the compiler core of incalang might be easy to do

3. Application as a programming language – incalang as **tool**

    - incalang is suited for developing tools or doing rapid prototyping due to its fast compiler
    - incalang has all needed facilities to write even complex applications
    - the IDE and editor might be useful especially for someone learning to program. The setup is very easy.