Projeto Ouroboros

Um sistema de integração automatizada entre C++ e linguagens de *script*

Wilson Kazuo Mizutani e Fernando Omar Aluani

11 de novembro de 2013

Orientador: Prof. Dr. Marco Dimas Gubitoso

1. Introdução: Motivações e Objetivos

Sistema de *scripts* da UGDK (2011)

Sistema de *scripts* da UGDK (2011)

```
23  entrance = {
24    neighborhood = { "opencorridor" },
25    width = roomsize,
26    height = roomsize,
27    matrix = [[
28  %%%%%%%%%%%%%%%%
29  %..............%
30  %..............%
31  %..............%
32  %..............%
33  %..%..%..%..%..%
34  %...............
35  %...............
36  %...............
37  %...............
38  %..%..%..%..%..#
39  %..............#
40  %..............#
41  %..............#
42  %..............#
43  ################
44  ]],
45    objects = {},
46    recipes = {
47      ["urn"] = { property = "urn" },
48    },
49    collision_classes = {
50      { "Switch", "Wall" }
51    },
52    setup = function(self)
53      for i = 1,12 do
54        local x,y = 1+i, 2
55        for i = 1,math.random(3,4) do
56          self:MakeRecipe("urn", ugdk_math.Vector2D(x+math.random()*2-1, y+math.random()*2-1))
57        end
58        y = 13
59        for i = 1,math.random(3,4) do
60          self:MakeRecipe("urn", ugdk_math.Vector2D(x+math.random()*2-1, y+math.random()*2-1))
61        end
62      end
63    end
64  }
```

Sistema de *scripts* da UGDK (2011)

```
entrance = {
  neighborhood = { "opencorridor" },
  width = roomsize,
  height = roomsize,
  matrix = [[
%%%%%%%%%%%%%%%%
%..............%
%..............%
%..............%
%..............%
%..%..%..%..%..%
%..............
%..............
%..............
%..............
%..%..%..%..%..#
%..............#
%..............#
%..............#
%..............#
################
]],
  objects = {},
  recipes = {
    ["urn"] = { property = "urn" },
  },
  collision_classes = {
    { "Switch", "Wall" }
  },
  setup = function(self)
    for i = 1,12 do
      local x,y = 1+i, 2
      for i = 1,math.random(3,4) do
        self:MakeRecipe("urn", ugdk_math.Vector2D(x+math.random()*2-1, y+math.random()*2-1))
      end
      y = 13
      for i = 1,math.random(3,4) do
        self:MakeRecipe("urn", ugdk_math.Vector2D(x+math.random()*2-1, y+math.random()*2-1))
      end
    end
  end
}
```

- Interface única para diferentes linguagens de *script*

Sistema de *scripts* da UGDK (2011)

```
entrance = {
  neighborhood = { "opencorridor" },
  width = roomsize,
  height = roomsize,
  matrix = [[
%%%%%%%%%%%%%%%%
%..............%
%..............%
%..............%
%..............%
%..%..%..%..%..%
%..............
%..............
%..............
%..............
%..%..%..%..%..#
%..............#
%..............#
%..............#
%..............#
################
]],
  objects = {},
  recipes = {
    ["urn"] = { property = "urn" },
  },
  collision_classes = {
    { "Switch", "Wall" }
  },
  setup = function(self)
    for i = 1,12 do
      local x,y = 1+i, 2
      for i = 1,math.random(3,4) do
        self:MakeRecipe("urn", ugdk_math.Vector2D(x+math.random()*2-1, y+math.random()*2-1))
      end
      y = 13
      for i = 1,math.random(3,4) do
        self:MakeRecipe("urn", ugdk_math.Vector2D(x+math.random()*2-1, y+math.random()*2-1))
      end
    end
  end
}
```

- Interface única para diferentes linguagens de *script*
- Exportação automatizada das classes em C++ (via SWIG).

Proposta de TCC (2013)

Proposta de TCC (2013)

Projeto Ouroboros

Proposta de TCC (2013)

Projeto Ouroboros

- Resolver os problemas do SWIG

Proposta de TCC (2013)

Projeto Ouroboros

- Resolver os problemas do SWIG
- Independência da UGDK

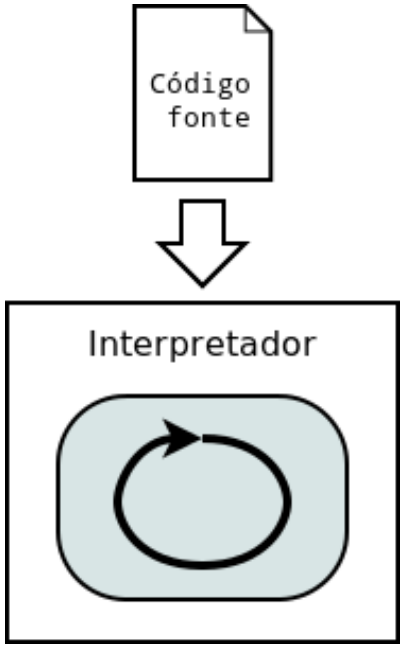
Proposta de TCC (2013)

Projeto Ouroboros

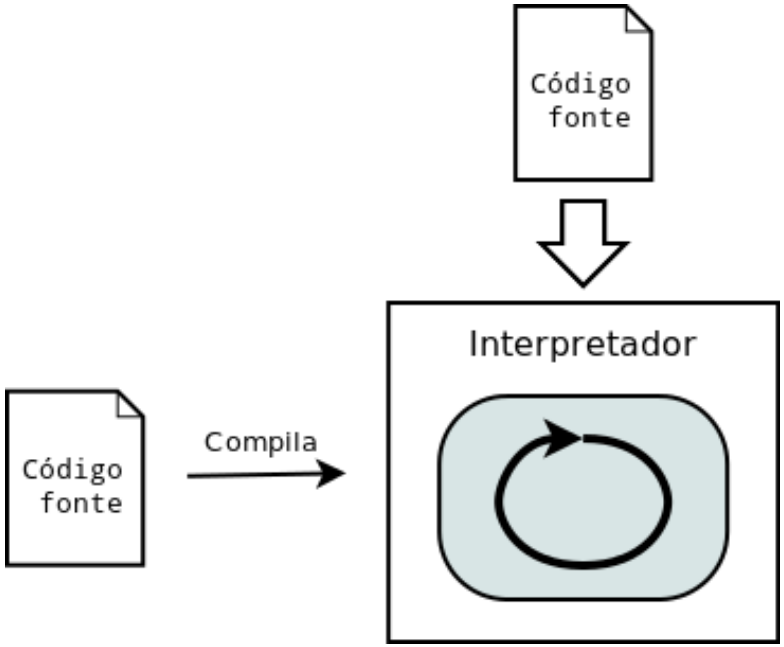
- Resolver os problemas do SWIG
- Independência da UGDK
- Software Livre

A API das linguagens

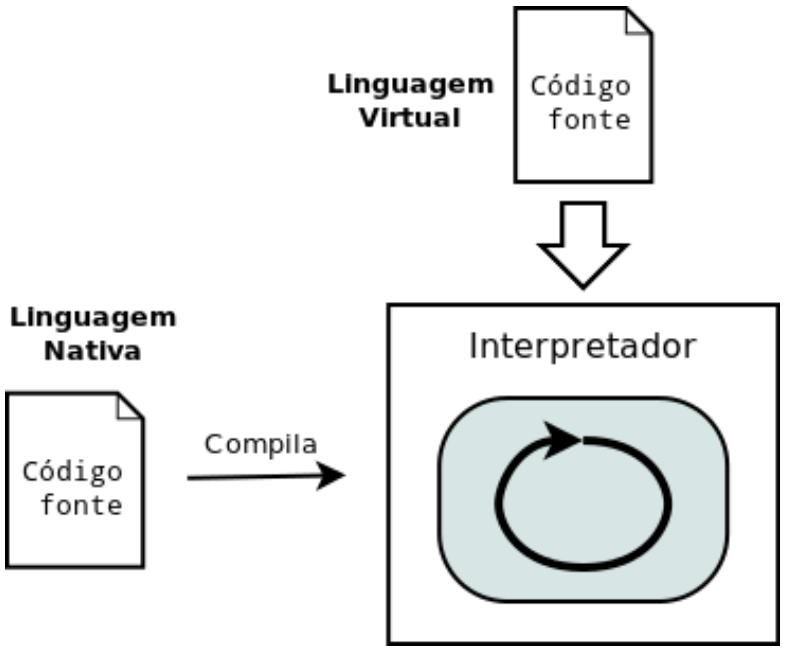
O interpretador



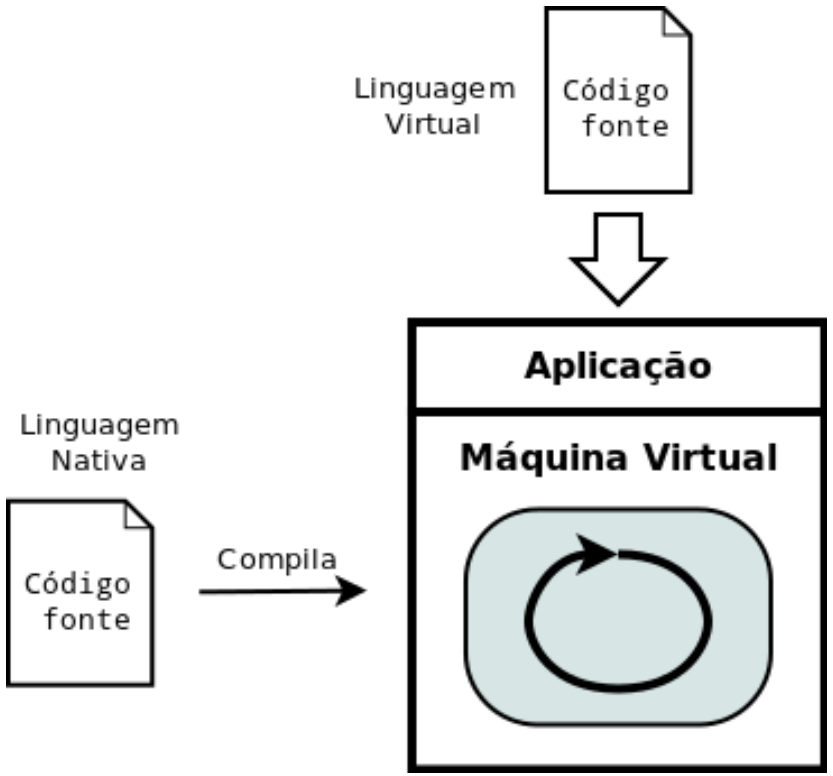
Como ele funciona?



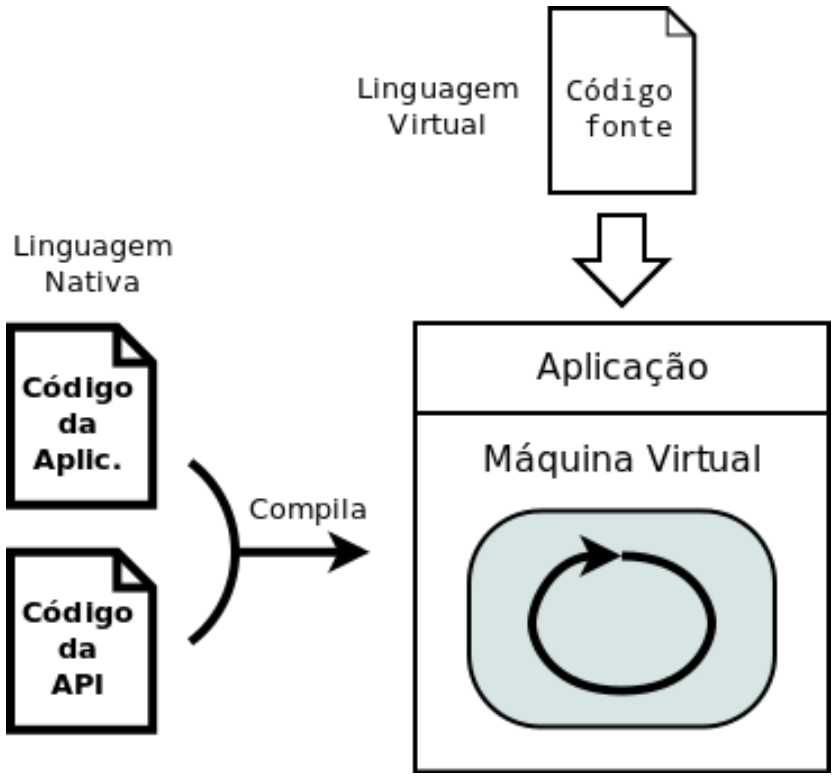
Linguagem Nativa e Linguagem Virtual



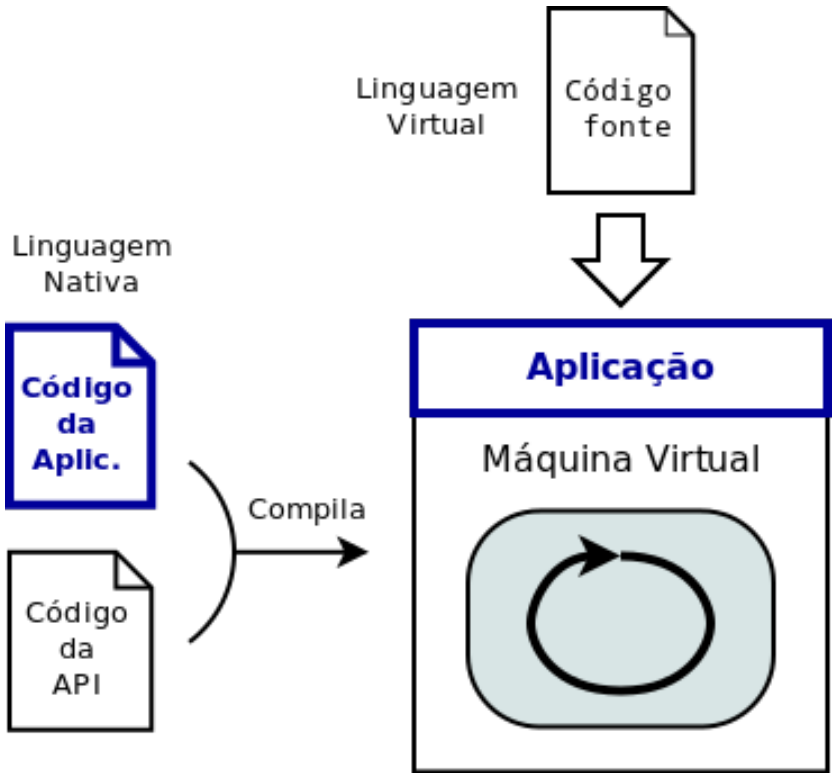
A Máquina Virtual



A API Nativa


Linguagem
Virtual
Código
fonte
Linguagem
Nativa
Código
da
Aplic.
Código
da
API
Compila
Aplicação
Máquina Virtual

Onde queremos mexer



3. O Sistema

3.1. Visão Geral

O Problema

O Problema

Integrar, de maneira automatizada, uma linguagem nativa (C++) com mais de uma linguagem de *script* (Lua, Python, ...).

O Problema

Integrar, de maneira automatizada, uma linguagem nativa (C++) com mais de uma linguagem de *script* (Lua, Python, ...).

Os casos

Os casos

A integração ocorre em dois sentidos:

Os casos

A integração ocorre em dois sentidos:

- Máquina Virtual $\Rightarrow$ Aplicação: **Incorporação**.

Os casos

A integração ocorre em dois sentidos:

- Máquina Virtual $\Rightarrow$ Aplicação: **Incorporação**.
- Aplicação $\Rightarrow$ Máquina Virtual: **Exportação**.

3.2. Incorporação
(a.k.a. *embedding*)

O que é incorporação?

O que é incorporação?

myscript.lua

```
1 variable = 42
2 function foo (x, y)
3   return x+y
4 end
```

O que é incorporação?

myscript.lua

```lua
variable = 42
function foo (x, y)
  return x+y
end
```

myscript.py

```python
variable = 42
def foo(x, y):
  return x+y
```

O que é incorporação?

myscript.lua

```lua
variable = 42
function foo (x, y)
  return x+y
end
```

myscript.py

```python
variable = 42
def foo(x, y):
  return x+y
```

myprog.cxx

```cpp
ScriptManager *manager = SCRIPT_MANAGER();
// Load the script
VirtualObj myscript = manager->LoadModule("myscript");
// Gets the value 42
int script_integer = myscript["variable"].value<int>();
// Results in 9001
int result = myscript["foo"](9000, 1).value<int>();
```

Como funciona? [1] [2]

Como funciona? [1] [2]

Incorporando com Lua

```
1 // L is a lua_State*
2 lua_getfield(L, LUA_GLOBALSINDEX, "variable");
3 // The value is put on the top of the stack
4 int value = lua_tointeger(L, -1);
```

Como funciona? [1] [2]

Incorporando com Lua

```
1 // L is a lua_State*
2 lua_getfield(L, LUA_GLOBALSINDEX, "variable");
3 // The value is put on the top of the stack
4 int value = lua_tointeger(L, -1);
```

Incorporando com Python

```
1 // module is a PyObject*
2 PyObject *var = PyObject_GetAttrString(module, "variable");
3 // var contains the value we want
4 int value = PyInt_AsLong(var);
```

Interface Unificada

Interface Unificada

Ouroboros Project API
(OPA)

Interface Unificada

Ouroboros Project API
(OPA)

Principais classes:

Interface Unificada

Ouroboros Project API (OPA)

Principais classes:

- `VirtualObj`: abstrai quaisquer objetos de *script*.

Interface Unificada

Ouroboros Project API (OPA)

Principais classes:

- `VirtualObj`: abstrai quaisquer objetos de *script*.
- `VirtualMachine`: abstrai cada linguagem virtual.

Interface Unificada

Ouroboros Project API (OPA)

Principais classes:

- `VirtualObj`: abstrai quaisquer objetos de *script*.
- `VirtualMachine`: abstrai cada linguagem virtual.
- `ScriptManager`: gerencia as máquinas virtuais.

Interface Unificada

Ouroboros Project API
(OPA)

Principais classes:

- `VirtualObj`: abstrai quaisquer objetos de *script*.
- `VirtualMachine`: abstrai cada linguagem virtual.
- `ScriptManager`: gerencia as máquinas virtuais.

Como?

Interface Unificada

Ouroboros Project API
(OPA)

Principais classes:

- `VirtualObj`: abstrai quaisquer objetos de *script*.
- `VirtualMachine`: abstrai cada linguagem virtual.
- `ScriptManager`: gerencia as máquinas virtuais.

Como? **Polimorfismo!**

Tipos definidos pelo Usuário

Tipos definidos pelo Usuário

E se eu quiser fazer isso?

Tipos definidos pelo Usuário

E se eu quiser fazer isso?

Usando tipo do usuário

```
1 // Load the script
2 VirtualObj myscript = manager->LoadModule("myscript");
3 // Some object
4 MyClass *object = myscript["object"].value<MyClass*>();
```

Tipos definidos pelo Usuário

E se eu quiser fazer isso?

Usando tipo do usuário

```
1 // Load the script
2 VirtualObj myscript = manager->LoadModule("myscript");
3 // Some object
4 MyClass *object = myscript["object"].value<MyClass*>();
```

- A máquina virtual tem que conhecer `MyClass`.

Tipos definidos pelo Usuário

E se eu quiser fazer isso?

Usando tipo do usuário

```
1 // Load the script
2 VirtualObj myscript = manager->LoadModule("myscript");
3 // Some object
4 MyClass *object = myscript["object"].value<MyClass*>();
```

- A máquina virtual tem que conhecer `MyClass`.
- Precisa tratar conversões de tipo implícitas.

Tipos definidos pelo Usuário

E se eu quiser fazer isso?

Usando tipo do usuário

```
1 // Load the script
2 VirtualObj myscript = manager->LoadModule("myscript");
3 // Some object
4 MyClass *object = myscript["object"].value<MyClass*>();
```

- A máquina virtual tem que conhecer `MyClass`.
- Precisa tratar conversões de tipo implícitas.
- Tem que gerenciar a memória.

3.3. Exportação

O que é exportação?

O que é exportação?

myprog.h

```
1 class MyClass { /* ... */ };
2
3 int bar (int x) { return x+x; }
```

O que é exportação?

myprog.h

```
1 class MyClass { /* ... */ };
2
3 int bar (int x) { return x+x; }
```

myscript.lua

```
1 require 'myprog'
2
3 obj = myprog.MyClass()
4
5 print(myprog.bar(1337))
```

O que é exportação?

myprog.h

```
1 class MyClass { /* ... */ };
2
3 int bar (int x) { return x+x; }
```

myscript.lua

```
1 require 'myprog'
2
3 obj = myprog.MyClass()
4
5 print(myprog.bar(1337))
```

myscript.py

```
1 from myprog import MyClass
2 from myprog import bar
3
4 obj = MyClass()
5
6 print(bar(1337))
```

Como funciona: Lua

Como funciona: Lua

Lua_myprog_wrap.cxx

```
1  int wrap_bar (lua_State* L) {
2    int arg0 = lua_tointeger(L, 1);
3    lua_settop(L, 0);
4    int result = bar(arg0);
5    lua_pushinteger(L, result);
6    return 1;
7  }
8
9  luaL_Reg functions[] = { { "bar", wrap_bar }, ... };
10
11 int luaopen_myprog (lua_State* L) {
12   lua_newtable(L);
13   luaL_register(L, nullptr, functions);
14   return 1;
15 }
```

Como funciona: Python

Como funciona: Python

Python_myprog_wrap.cxx

```
1  PyObject* wrap_bar (PyObject* self, PyObject* args) {
2    PyObject *arg0 = PyTuple_GetItem(args, 0);
3    int arg0_value = PyInt_AsLong(arg0);
4    int result_value = bar(arg0_value);
5    PyObject *result = PyInt_FromLong(result_value);
6    return result;
7  }
8
9  static PyMethodDef myprogmethods[] = {
10   {"bar", wrap_bar, ... }, ...
11 };
12
13 PyMODINIT_FUNC initmyprog (void) {
14   Py_InitModule("myprog", myprogmethods);
15 }
```

Como vocês podem ver...

Como vocês podem ver... É muito chato!

Como vocês podem ver... É muito chato!

Precisamos gerar esse código automaticamente!

Primeira solução

Primeira solução

Simple Wrapper and Interface Generator (SWIG)

Primeira solução

Simple Wrapper and Interface Generator (SWIG)

Faz o "trabalho sujo" para nós, mas tem problemas:

Primeira solução

Simple Wrapper and Interface Generator (SWIG)

Faz o "trabalho sujo" para nós, mas tem problemas:

- Não permite herança de classes exportadas.

Primeira solução

Simple Wrapper and Interface Generator (SWIG)

Faz o "trabalho sujo" para nós, mas tem problemas:

- Não permite herança de classes exportadas.
- Não reconhece certas estruturas, como classes aninhadas.

Primeira solução

Simple Wrapper and Interface Generator (SWIG)

Faz o "trabalho sujo" para nós, mas tem problemas:

- Não permite herança de classes exportadas.
- Não reconhece certas estruturas, como classes aninhadas.
- Complicações para gerenciar a memória.

Primeira solução

Simple Wrapper and Interface Generator (SWIG)

Faz o "trabalho sujo" para nós, mas tem problemas:

- Não permite herança de classes exportadas.
- Não reconhece certas estruturas, como classes aninhadas.
- Complicações para gerenciar a memória.
- Uma interface não muito agradável.

Segunda solução

Segunda solução

Ouroboros Project Wrapper and Interface Generator (OPWIG)

Segunda solução

Ouroboros Project Wrapper and Interface Generator (OPWIG)

Principais partes:

Segunda solução

Ouroboros Project Wrapper and Interface Generator (OPWIG)

Principais partes:

- *Parser*: analisa as declarações em C++ do usuário.

Segunda solução

Ouroboros Project Wrapper and Interface Generator (OPWIG)

Principais partes:

- *Parser*: analisa as declarações em C++ do usuário.
- Metadados: abstraem as declarações analisadas.

Segunda solução

Ouroboros Project Wrapper and Interface Generator (OPWIG)

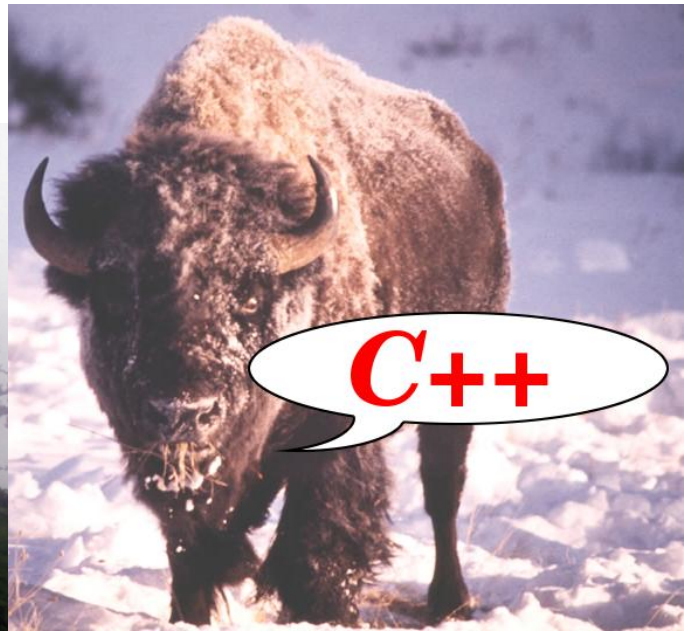
Principais partes:

- *Parser*: analisa as declarações em C++ do usuário.
- Metadados: abstraem as declarações analisadas.
- Gerador: gera o código dos *wrappers* com base nos metadados.

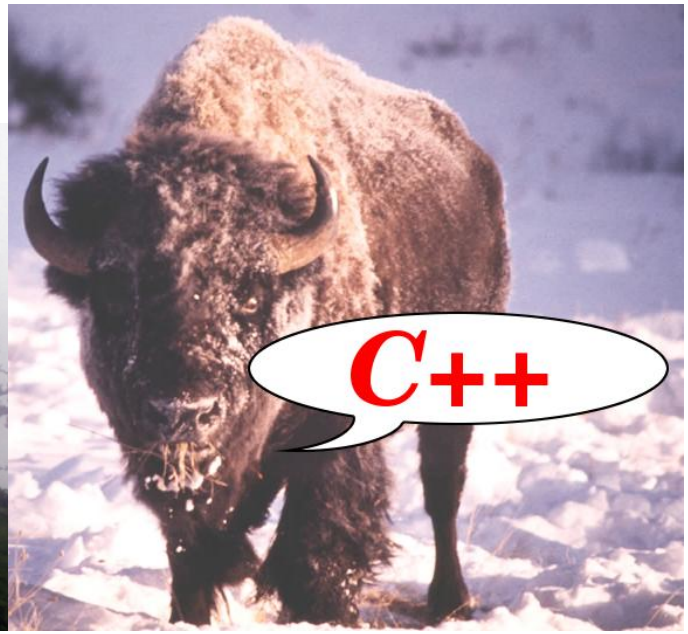
Implementação do *Parser*

Implementação do *Parser*

Implementação do *Parser*



Implementação do *Parser*



`Flexc++` e `Bisonc++` por Frank B. Brokken [3][4]

4. Conclusões

Funciona?

Funciona?

Testes!

Funciona?

Testes!

- Sistema antigo: usado pelo Horus Eye e pelo Asteroids Wars.

Funciona?

Testes!

- Sistema antigo: usado pelo Horus Eye e pelo Asteroids Wars.
- *Parser* e metadados: testes de unidade e de funcionalidade, usando `gooogletest` [5] e Travis [6].

Funciona?

Testes!

- Sistema antigo: usado pelo Horus Eye e pelo Asteroids Wars.
- *Parser* e metadados: testes de unidade e de funcionalidade, usando `gooogletest` [5] e Travis [6].
- OPA e OPWIG: testes de aceitação na forma de *milestones*.

Dificuldades

Dificuldades

- Compatibilidade com `C++11` em Windows.

Dificuldades

- Compatibilidade com C++11 em Windows.
- C++ não tem uma gramática livre de contexto.

Dificuldades

- Compatibilidade com C++11 em Windows.
- C++ não tem uma gramática livre de contexto.
- Falta de planejamento no começo.

Próximos passos

Próximos passos

- Lançamento oficial (organizar repositórios, criar página oficial...)

Próximos passos

- Lançamento oficial (organizar repositórios, criar página oficial...)
- Incrementar o *parser* e o gerador para ele reconhecer C++ completamente (incluindo C++11)

Próximos passos

- Lançamento oficial (organizar repositórios, criar página oficial...)
- Incrementar o *parser* e o gerador para ele reconhecer C++ completamente (incluindo C++11)
- Dar suporte a mais linguagens!

Próximos passos

- Lançamento oficial (organizar repositórios, criar página oficial...)
- Incrementar o *parser* e o gerador para ele reconhecer C++ completamente (incluindo C++11)
- Dar suporte a mais linguagens!
- Explorar usos alternativos com programação reflexiva em C++

Obrigado!

Bibliografia

1 Lua C API. http://www.lua.org/manual/5.1/manual.html#5, Novembro 2013.

2 Python C API. http://docs.python.org/2/c-api/, Novembro 2013.

3 Frank B. Brokken. http://flexcpp.sourceforge.net/, Novembro 2013.

4 Frank B. Brokken. http://bisoncpp.sourceforge.net/, Novembro 2013.

5 Google. https://code.google.com/p/googletest/, Novembro 2013.

6 Travis CI. https://travis-ci.org/, Novembro 2013.