Jancy

LLVM-based scripting language for IO and UI programming

Vladimir Gladkov
Tibbo Technology Inc

http://tibbo.com/jancy

Why?! Do we need more?

Not logged in Talk Contributions Create account Log in

Article Talk | Read Edit View history | Search

WIKIPEDIA
The Free Encyclopedia

Main page
Contents
Featured content
Current events
Random article
Donate to Wikipedia
Wikipedia store

nteraction
Help
About Wikipedia
Community portal
Recent changes
Contact page

Tools
What links here
Related changes
Upload file
Special pages
Permanent link
Page information
Wikidata item
Cite this page

Print/export

List of programming languages

From Wikipedia, the free encyclopedia

The aim of this **list of programming languages** is to include all notable programming languages in existence, both those in current use and historical ones, in alphabetical order, except for dialects of BASIC and esoteric programming languages.

***Note**: Dialects of BASIC have been moved to the separate List of BASIC dialects.*

***Note**: This page does not list esoteric programming languages.*

Programming language lists

- **Alphabetical**
- Categorical
- Chronological
- Generational

V·T·E

A · B · C · D · E · F · G · H · I · J · K · L · M · N · O · P · Q · R · S · T · U · V · W · X · Y · Z

Contents : See also

A [edit]

- A# .NET
- A# (Axiom)
- A-0 System
- A+
- A++
- ABAP
- ABC
- Ada
- Adenine
- Agda
- Agilent VEE
- Agora
- AIMMS
- Alef
- Apex (Salesforce.com)
- API
- ARexx

~700 already!!

Wanted! (for IO Ninja)

- IO
  - Safe pointer arithmetic
  - High level of source compatibility with C
  - Built-in incremental lexer generator

Wanted! (for IO Ninja)

- IO
  - Safe pointer arithmetic
  - High level of source compatibility with C
  - Built-in incremental lexer generator
- UI
  - Properties
  - Events
  - Excel-like “reactive“ evaluation

Jancy Design Goals

- Embedded scripting language
- Statically typed
- C-family language syntax
- ABI-compatible with C
- Garbage collected (accurate GC)
- LLVM as back-end

Other interesting features

- Const-correctness
- Multiple inheritance
- Partial application
- Schedulers
- Exception-style syntax over error code checks
- Dual type modifiers
- Bigendian integers
- Bitflag enums
- Break-n/Continue-n
- Hex literals

Handling binary data (wrong)

```
public class IPv4Packet {
    private static final int IP_TOS_POS = 1;  // type of service
    private static final int IP_LEN_POS = 2;  // total packet length
    private static final int IP_ID_POS = 4;   // the packet id
    private static final int IP_FRAG_POS = 6; // the frag flags and offset
    // ...
    public int getTypeOfService() {
        if (_isReadTOS == false) {
            myTOS = myPacket[myIPHdrOffset + IP_TOS_POS] & 0x0f;
            _isReadTOS = true;
        }
        return myTOS;
    }
    public int getFragmentFlags() {
        if (_isReadFragFlags == false) {
            _isReadFragFlags = true;
            myFragmentFlags = ByteUtils.getByteNetOrderTo_uint16(
                myPacket, myIPHdrOffset + IP_FRAG_POS) >> 13;
        }
        return myFragmentFlags;
    }
    // ...
}
```

Handling binary data (wrong)

```
public class IPv4Packet {
    private static final int IP_TOS_POS = 1;  // type of service
    private static final int IP_LEN_POS = 2;  // total packet length
    private static final int IP_ID_POS = 4;   // the packet id
    private static final int IP_FRAG_POS = 6; // the frag flags and offset
    // ...
    public int getTypeOfService() {
        if (_isReadTOS == false) {
            myTOS = myPacket[myIPHdrOffset + IP_TOS_POS] & 0x0f;
            _isReadTOS = true;
        }
        return myTOS;
    }
    public int getFragmentFlags() {
        if (_isReadFragFlags == false) {
            _isReadFragFlags = true;
            myFragmentFlags = ByteUtils.getByteNetOrderTo_uint16(
                myPacket, myIPHdrOffset + IP_FRAG_POS) >> 13;
        }
        return myFragmentFlags;
    }
    // ...
}
```

Handling binary data (wrong)

```
public class IPv4Packet {
    private static final int IP_TOS_POS = 1;  // type of service
    private static final int IP_LEN_POS = 2;  // total packet length
    private static final int IP_ID_POS = 4;   // the packet id
    private static final int IP_FRAG_POS = 6; // the frag flags and offset
    // ...
    public int getTypeOfService() {
        if (_isReadTOS == false) {
            myTOS = myPacket[myIPHdrOffset + IP_TOS_POS] & 0x0f;
            _isReadTOS = true;
        }
        return myTOS;
    }
    public int getFragmentFlags() {
        if (_isReadFragFlags == false) {
            _isReadFragFlags = true;
            myFragmentFlags = ByteUtils.getByteNetOrderTo_uint16(
                myPacket, myIPHdrOffset + IP_FRAG_POS) >> 13;
        }
        return myFragmentFlags;
    }
    // ...
}
```

Handling binary data (right)

Step #1 – Define data layout

```
struct IpHdr
{
    uint8_t m_headerLength : 4;
    uint8_t m_version      : 4;
    uint8_t m_typeOfService;
    // ...
}

struct IcmpHdr
{
    uint8_t m_type;
    uint8_t m_code;
    bigendian uint16_t m_checksum;
    // ...
}
```

Handling binary data (right)

Step #2 – Access buffer

```
printIpHdr (void const* buffer)
{
    IpHdr const* ipHdr = (IpHdr const*) buffer;

    print ($"IP version = $(ipHdr.m_version)\n");
    // ...

    if (ipHdr.m_protocol == IPPROTO_ICMP)
    {
        buffer += ipHdr.m_headerLength * 4;
        IcmpHdr const* icmpHdr = (IcmpHdr const*) buffer;

        print ($"ICMP type = $(icmpHdr.m_type)\n");
        // ...
    }
}
```

Handling binary data (right)

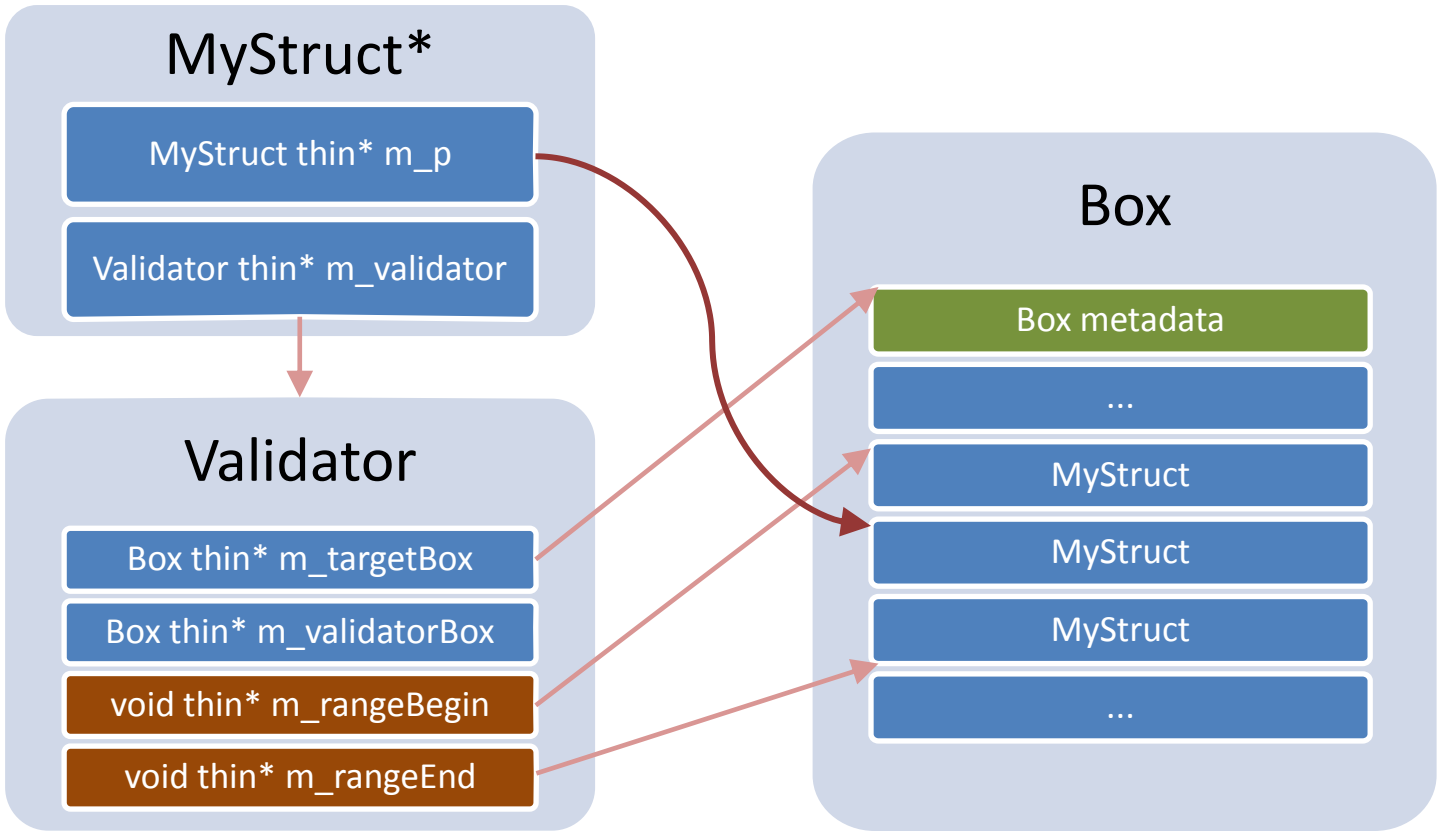
Step #2 – Access buffer

```
printIpHdr (void const* buffer)
{
    IpHdr const* ipHdr = (IpHdr const*) buffer;

    print ($"IP version = $(ipHdr.m_version)\n");
    // ...

    if (ipHdr.m_protocol == IPPROTO_ICMP)
    {
        buffer += ipHdr.m_headerLength * 4;
        IcmpHdr const* icmpHdr = (IcmpHdr const*) buffer;

        print ($"ICMP type = $(icmpHdr.m_type)\n");
        // ...
    }
}
```

How is pointer arithmetic safe?

Fat pointers, obviously

```
MyStruct* p;
```



Loads/stores are bounds checked

Pointer dereference

```
foo (char* p, size_t i)
{
    p += i;
    *p = 10; // <-- range is checked
}
```

Array indexing

```
bar (size_t i)
{
    static int a [] = { 10, 20, 30 };
    int x = a [i]; // <-- range is checked
}
```

Dynamic sizeof/countof

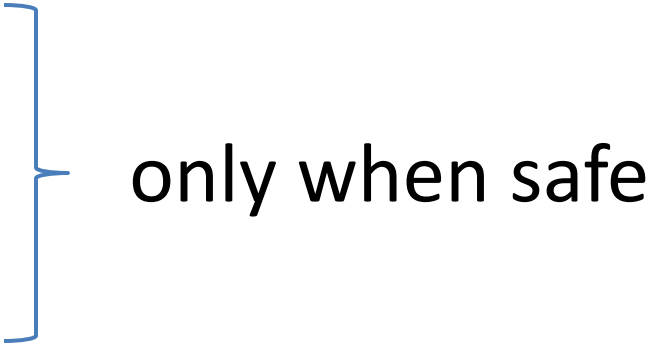
```
foo (int* a)
{
    size_t count = dynamic countof (a);
    for (size_t i = 0; i < count; i++)
    {
        // do something with a [i]
    }
}
```

Are bounds checks enough?

- Dangling pointers?
- Unions?
- Reinterpret casts?
- Pointer-to-fields increments?
- Downcasts?

Are bounds checks enough?

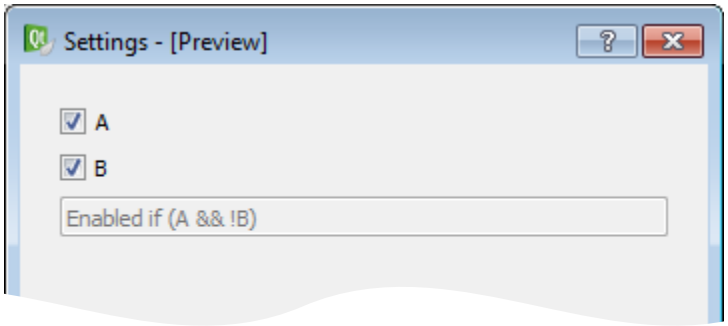
- Dangling pointers – impossible
- Unions
- Reinterpret casts
- Pointer-to-fields increments
- Downcasts – dynamic casts




```
foo (Parent* a)
{
    Child* c = dynamic (Child*) a;
    // ...
}
```

Reactive Programming for UI

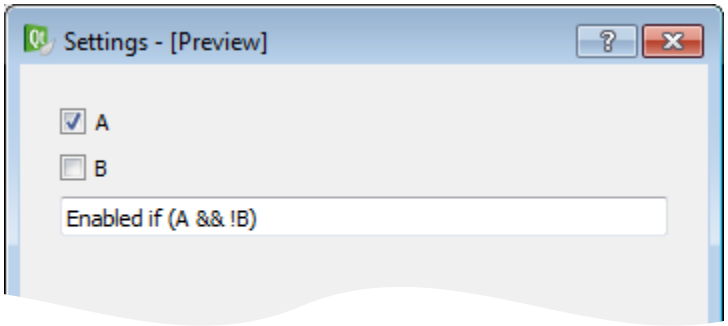
- Automatic propagation of changes
- Observable pattern
- Our goal: Excel-like re-evaluation for UI
- Our workhorses:
  - Multicasts & events
  - Properties




```
m_editBox.m_isEnabled =
    m_checkBoxA.m_isChecked &&
    !m_checkBoxB.m_isChecked;
```

Reactive Programming for UI

- Automatic propagation of changes
- Observable pattern
- Our goal: Excel-like re-evaluation for UI
- Our workhorses:
  - Multicasts & events
  - Properties




```
m_editBox.m_isEnabled =
    m_checkBoxA.m_isChecked &&
    !m_checkBoxB.m_isChecked;
```

Multicasts & events

```
class C1
{
    event m_onComplete ();

    work ()
    {
        // ...
        m_onComplete (); // OK, 'call' is accessible from C1
    }
}

foo (C1* c)
{
    multicast m (int);
    m += bar;
    m += baz;
    m (100);  // <-- foo (100); bar (100);

    c.m_onComplete (); // <-- error, 'call' is inaccessible
}
```

Bindable properties

```
int bindable property g_bindableProp;
```

```
g_bindableProp.set (int x)
{
    if (x == m_value)
        return;

    m_value = x;
    m_onChanged (); // compiler-generated event is 'm_onChanged'
}
```

```
onPropChanged ()
{
    // ...
}

foo ()
{
    bindingof (g_bindableProp) += onPropChanged;
    g_bindableProp = 100; // onPropChanged will be called
}
```

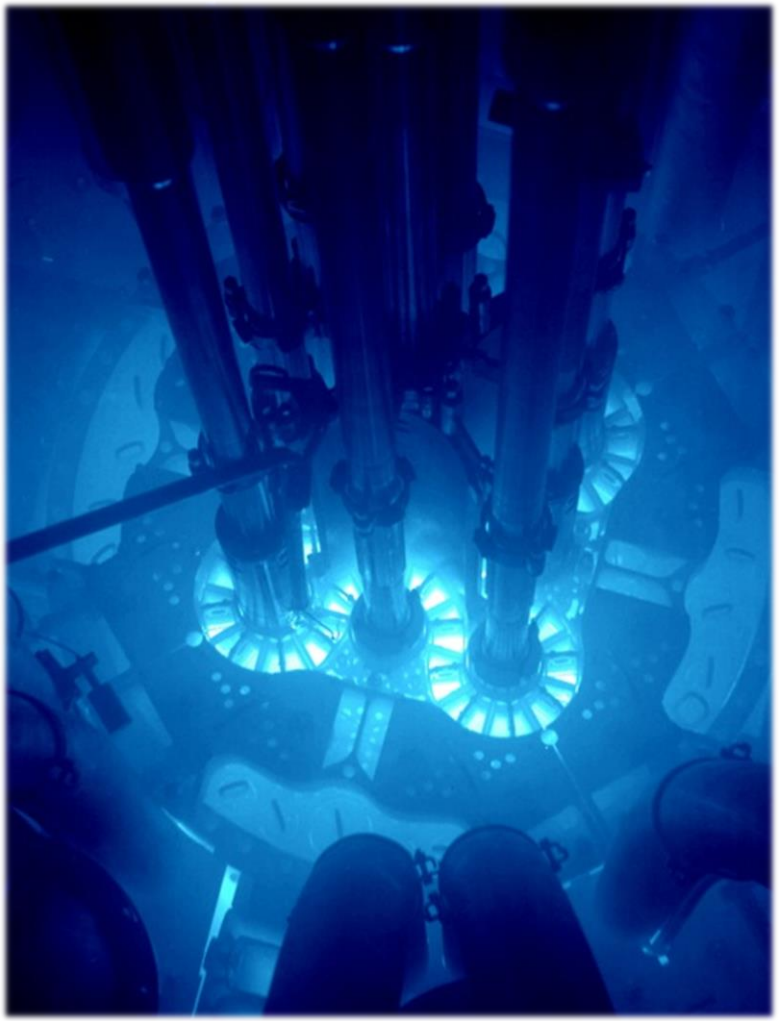
Dilemma

- We want Excel-like re-evaluation

Dilemma

- We want Excel-like re-evaluation
- Implicit subscriptions are hard to control

Solution – reactors!



Solution – reactors!

```
reactor TcpConnectionSession.m_uiReactor ()
{
    m_title = $"TCP $(m_addressCombo.m_editText)";
    m_isTransmitEnabled = m_state == State.Connected;
    m_actionTable [ActionId.Disconnect].m_isEnabled = m_state != State.Closed;
    m_adapterProp.m_isEnabled = m_useLocalAddressProp.m_value;
    m_localPortProp.m_isEnabled = m_useLocalAddressProp.m_value;
}
```

Solution – reactors!

```
reactor TcpConnectionSession.m_uiReactor ()
{
    m_title = $"TCP $(m_addressCombo.m_editText)";
    m_isTransmitEnabled = m_state == State.Connected;
    m_actionTable [ActionId.Disconnect].m_isEnabled = m_state != State.Closed;
    m_adapterProp.m_isEnabled = m_useLocalAddressProp.m_value;
    m_localPortProp.m_isEnabled = m_useLocalAddressProp.m_value;
}
```

Automated, but controlled

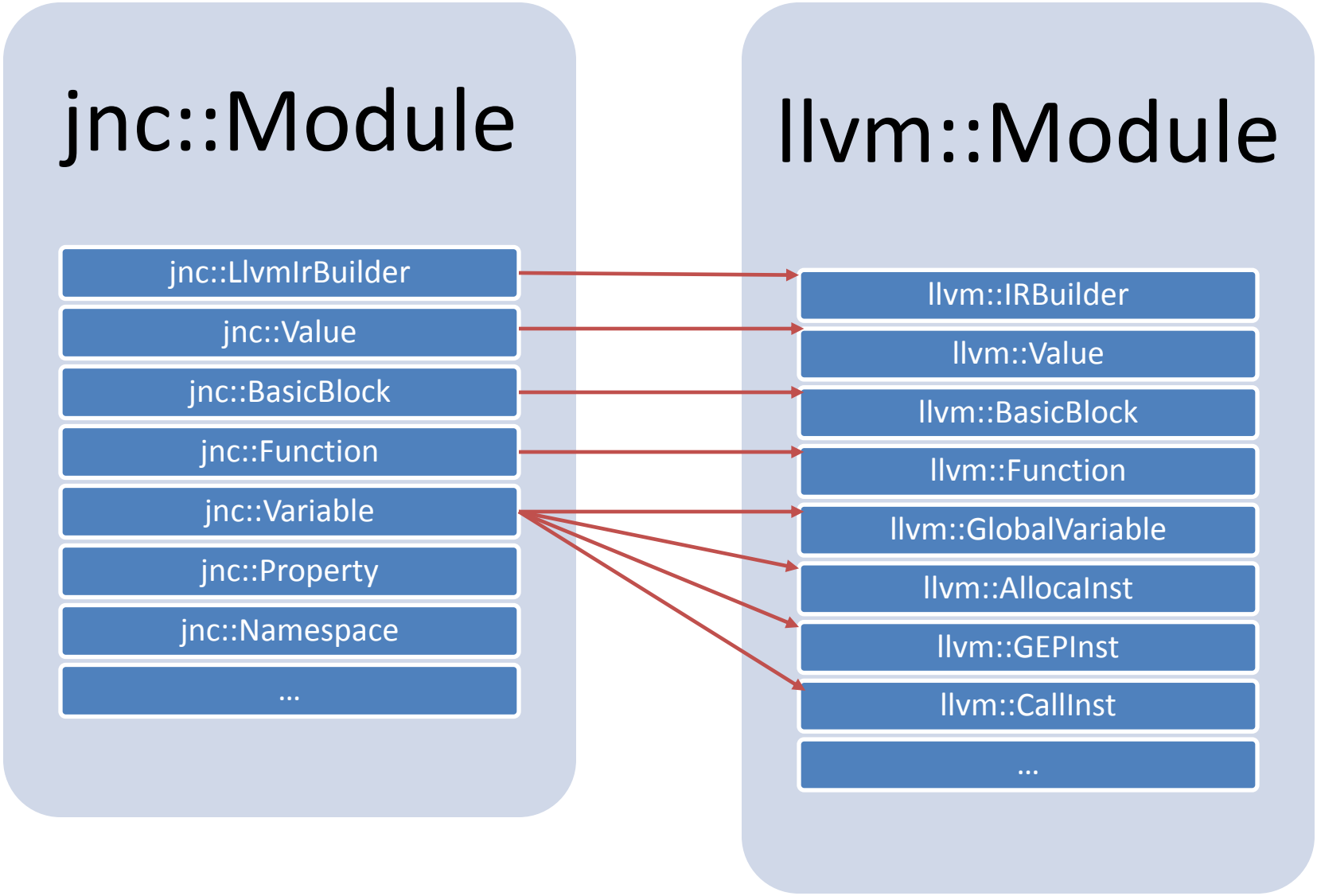
```
reactor m_uiReactor ()
{
    m_title = $"TCP $(m_addressCombo.m_editText)";
    m_isTransmitEnabled = m_state == State.Connected;
    // ...

    onevent m_transmitButton.m_onClicked ()
    {
        // handle start button click...
    }

    onevent (m_userEdit.m_onChanged, m_passwordEdit.m_onChanged) ()
    {
        // handle login change...
    }
}
```

```
m_uiReactor.start ();
// ...
m_uiReactor.stop ();
```

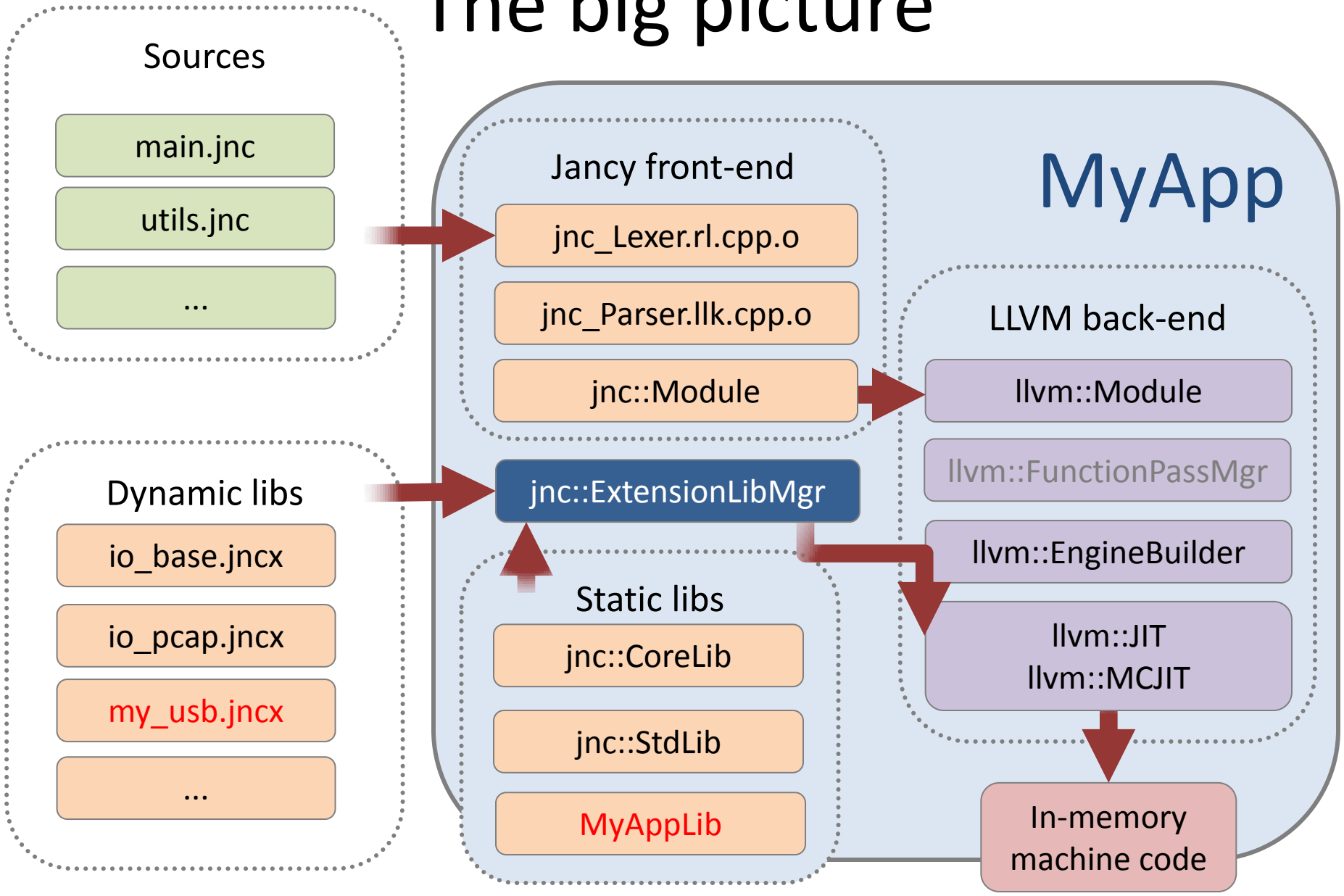
Implementation

- Main goal: embedded scripting
- Ragel-generated lexer as a front-end
- Table-driven generated top-down parser
- LLVM API to generate in-memory IR
- LLVM MCJIT to machine code
- Plugins for NetBeans IDE

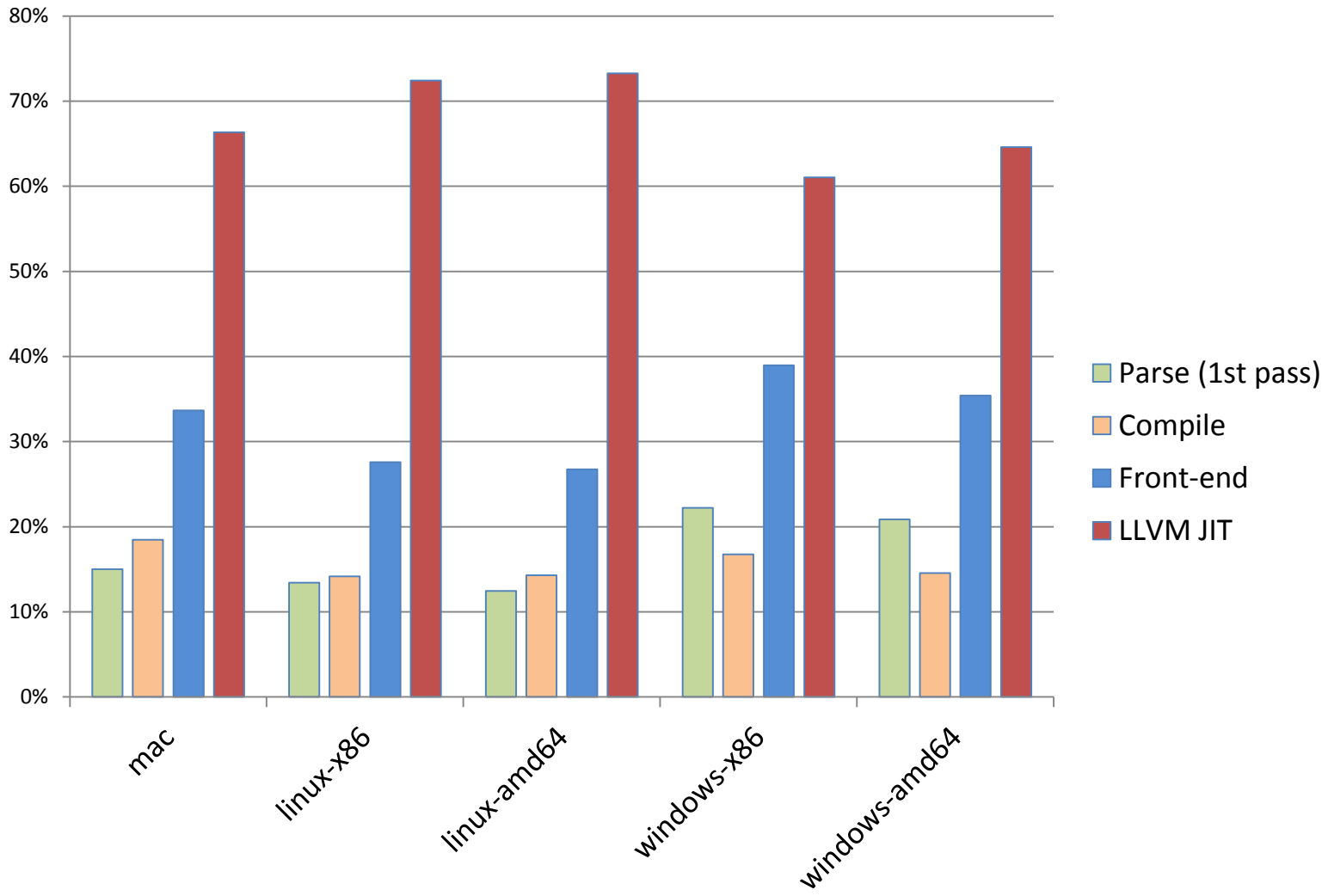
jnc::Module vs llvm::Module



The big picture



Where is time spent?



Summary

- Open source LLVM-based scripting language
- Offers unique features
- Used in a real-life product IO Ninja
- Comes with NetBeans-based IDE
- Live demo on the website
- Play, contribute, use in your projects

http://tibbo.com/jancy

vovkos@tibbo.com

Overview

- Why?
- 2 main Jancy features
- Compiler design and how we use LLVM
- Questions

Why not recursive descent?

- Easy to experiment with syntax
- EBNF grammar as a permanently relevant syntax reference
- Natural constraints to not let you get too crazy with the syntax

Partial application

```
class C1
{
    foo (int x, int y, int z);

    // ...
}

bar ()
{
    C1 c;

    function* f (int, int) = c.foo ~(,, 300);
    f (100, 200); // => c.foo (100, 200, 300);
}
```

Autoget properties

```
int autoget property g_simpleProp;
```

```
g_simpleProp.set (int x)
{
    m_value = x; // the name of a compiler-generated field is 'm_value'
}
```

Indexed properties

```
int indexed property g_simpleProp (size_t i);
```

```
property g_prop
{
    int get (size_t i, size_t j);

    set (size_t i, size_t j, int x);
    set (size_t i, size_t j, double x);
}
```

```
foo ()
{
    int x = g_simpleProp [10];
    g_prop [x] [20] = 100;
}
```

Handling binary data (right)

Step #1 – Define structures

```
class IpHdr (BigEndianStructure):
    _fields_ = [
        ("version",         c_ubyte, 4),
        ("headerLength",    c_ubyte, 4),
        ("typeOfService",   c_ubyte),
        # ...
    ]

class IcmpHdr (BigEndianStructure):
    _fields_ = [
        ("type",            c_ubyte),
        ("code",            c_ubyte),
        ("headerChecksum", c_ushort),
        # ...
    ]
```

Handling binary data (right)

Step #2 – Access buffer

```
def packStruct (dst, src):
    size = min (len (src), sizeof (dst))
    memmove (addressof (dst), src, size)

def printIp (buffer):
    ipHdr = IpHdr ()
    packStruct (ipHdr, buffer)

    print "IP version = ", ipHdr.version
    # ...

    if ipHdr.protocol == 1:
        icmpHdr = IcmpHdr ()
        packStruct (icmpHdr, buffer [ipHdr.headerLength * 4:])

        print "ICMP type     = ", icmpHdr.type
        #...
```

Handling binary data (right)

Step #2 – Access buffer

```
def packStruct (dst, src):
    size = min (len (src), sizeof (dst))
    memmove (addressof (dst), src, size)

def printIp (buffer):
    ipHdr = IpHdr ()
    packStruct (ipHdr, buffer)

    print "IP version = ", ipHdr.version
    # ...

    if ipHdr.protocol == 1:
        icmpHdr = IcmpHdr ()
        packStruct (icmpHdr, buffer [ipHdr.headerLength * 4:])

        print "ICMP type     = ", icmpHdr.type
        #...
```

Schedulers

```
class Scheduler
{
    abstract schedule (function* f ());
}
```

```
class WorkerThread: jnc.Scheduler
{
    override schedule (function* f ())
    {
        // enqueue f and signal worker thread event
    }

    workerThread ()
    {
        for (;;)
        {
            // wait for worker thread event
            function* f () = getNextRequest ();
            f ();
        }
    }
}
```

Using schedulers

```
startSomeAsyncStuff (function* onComplete (int));

foo (int result);

bar ()
{
    WorkerThread workerThread;
    workerThread.start ();

    // create a scheduled pointer and pass it as completion routine
    doSomeAsyncStuff  (foo @ workerThread);

    (foo @ workerThread) (-1); // ...or schedule immediatly
}
```

Real-life example from IO Ninja

```
TcpListenerSession.construct (doc.PluginHost* pluginHost)
{
    // ...

    m_listenerSocket = new io.Socket ();
    m_listenerSocket.m_onSocketEvent +=
        onListenerSocketEvent @ pluginHost.m_mainThreadScheduler;

    // ...
}

TcpListenerSession.onListenerSocketEvent (
    io.SocketEventParams const* params)
{
    // ...
}
```

POD vs non-POD

- POD: no meta-data
  - Primitive types
  - Aggregates of PODs
  - Can be arbitrary modified byte-by-byte
- Non-POD: have meta-data
  - Classes
  - Pointers
  - Aggregates of non-PODs
  - Cannot be arbitrary modified byte-by-byte

Only cast when it’s safe

Sample types:

```
struct PodParent
{
    int m_a;
}

struct NonPodChild: PodParent
{
    char const* m_s;
}
```

Only cast when it's safe

Upcasts:

```
foo (NonPodChild* b)
{
    PodParent* a = b;
}
```

Reinterpret casts:

- POD-to-POD
- non-POD-to-const-POD

```
foo (PodParent* a, NonPodChild* b)
{
    char* p = (char*) a;
    char* p2 = (char*) b; // <-- error
    char const* p3 = (char const*) b; // <-- OK
}
```

Const-correctness is forced

```
foo (int const* src)
{
    int a = *src;

    *src = 0; // <-- error

    int* p = (int*) src; // <-- error
    *p = 0;
}
```

Otherwise, dynamic cast

```
foo (PodParent* a)
{
    NonPodChild* c = (NonPodChild*) a; // <-- error
    NonPodChild const* c = (NonPodChild const*) a; // <-- error

    NonPodChild* c = dynamic (NonPodChild*) a; // OK
}
```

Is it enough to be safe?

Method #1 – Reinterpret casts

```
foo (char* a)
{
    intptr* p = (intptr*) &a; // get a pointer to our pointer
    p [1] = 0x1234; // replace validator with garbage
    *a = 0; // <-- bang?
}

bar ()
{
    intptr a [] = { 0x1234, 0x5678 }; // fat pointer buffer
    char* p = *(char**) a; // construct a bogus fat pointer
    *p = 0; // <-- bang?
}
```

Is it enough to be safe?

Method #2 – Downcasts

```
struct Parent
{
    // ...
}

struct Child: Parent
{
    char const* m_p;
}

foo (Parent* a)
{
    Child* c = (Child*) a; // assume a is Child
    char c = c.m_p [0]; // <-- bang?
}
```

Is it enough to be safe?

Method #2 – Field member pointers

```
struct MyStruct
{
    intptr m_i;
    char const* m_p;
}

foo (MyStruct* a)
{
    intptr* p = &a.m_i; // get a pointer to a field
    p [2] = 0x1234; // replace validator of m_p with garbage
    *a.m_p = 0; // <-- bang?
}
```

Properties

- Two forms of declarations
  - Simple
  - Full
- Read-only properties
- Indexed properties
- Autoget properties
- Bindable properties
- Property pointers!

Multicasts

```
multicast m (int);
```

```
void clear ();
intptr setup (function* (int)); // returns cookie
intptr add (function* (int)); // returns cookie
function* remove (intptr cookie) (int);
function* getSnapshot () (int);
void call (int);
```

```
m = foo;      // same as m.setup (foo);
m += bar;     // same as m.add (bar);
m -= cookie; // same as m.remove (cookie);
m = null;     // same as m.clear ();
m (10);       // same as m.call (10);
```

Simple property declaration

```
int property g_simpleProp;

int const property g_simpleConstProp;
```

```
int property g_simpleProp;

int g_simpleProp.get ()
{
    // ...
}

g_simpleProp.set (int x)
{
    // ...
}
```

Why you should NOT write a new programming language?

- Time- and effort-consuming
- New syntax and concepts for others to learn
- Immature and buggy toolset
- Lack of a good standard library
- Lack of community and 3rd party libraries
- Lack of good documentation
- …

Why you should NOT write a new programming language?

- Time- and effort-consuming
- New syntax and concepts for others to learn
- Immature and buggy toolset
- Lack of a good standard library
- Lack of community and 3rd party libraries
- Lack of good documentation
- …
- Chances are you will be the only user

Why you should STILL write a new programming language?

Selfish

- It’s really fun!
- Learn a lot
- Apply creativity
- We have LLVM now!
- Challenging task
  - Ego boost
  - Looks good in resume

Why you should STILL write a new programming language?

Selfish

- It’s really fun!
- Learn a lot
- Apply creativity
- We have LLVM now!
- Challenging task
  - Ego boost
  - Looks good in resume

Not as selfish

- Fill a gap
- Combine features
- Change annoying things
- Field-test new concepts
- Be a part of The Progress

Lexer generator

```
jnc.AutomatonResult automaton scanRx (jnc.Recognizer* recognizer)
{
    %% "get"
        createToken (Token.Get);

    %% "set"
        createToken (Token.Set);

    %% "quit"
        createToken (Token.Quit);

    %% [_\w][_\w\d]*
        createToken (Token.Identifier, recognizer.m_lexeme);

    // ...
}
```

Running automatons

```
jnc.Recognizer recognizer;
recognizer.m_automatonFunc = scanRx;
```

```
bool result = try recognizer.recognize ("getOption myOption");
if (!result)
{
    // handle recognition error
}
```

```
try
{
    recognizer.write ("ge");
    recognizer.write ("tOp");
    recognizer.write ("tion");
    recognizer.eof (); // notify recognizer about eof
                       // this can trigger actions or errors
catch:
    // handle recognition error
}
```

Switching languages

```
jnc.AutomatonResult automaton scanGlobal (jnc.Recognizer* recognizer)
{
    %% '#'
        recognizer.m_automatonFunc = scanPreprocessor; // switch to pp

    // ...
}
```

```
jnc.AutomatonResult automaton scanPreprocessor (jnc.Recognizer* recognizer)
{
    %% "if"
        createToken (Token.If);

    %% "ifdef"
        createToken (Token.Ifdef);

    // ...

    %% '\n'
        recognizer.m_automatonFunc = scanGlobal; // switch back
}
```

Properties

```
property g_prop
{
    int m_x = 5; // member field with in-place initializer

    int get ()
    {
        return m_x;
    }

    set (int x)
    {
        m_x = x;
        update ();
    }

    set (double x); // overloaded setter
    update ();      // helper method
}
```

Binding to bindable properties

```
int bindable property g_bindableProp;
```

```
onPropChanged ()
{
    // ...
}

foo ()
{
    bindingof (g_bindableProp) += onPropChanged;
    g_bindableProp = 100; // onPropChanged will be called
}
```

Under the hood

class ReactorClass

methods:

- start (...)
- stop ()

fields:

- int m_state
- ReactorBindSite m_bindSite [0]
- ReactorBindSite m_bindSite [1]
- ReactorBindSite m_bindSite [2]
- ...
- int m_reactionState [0]
- int m_reactionState [1]
- int m_reactionState [2]
- ...

```
reactor m_uiReactor ()
{
    m_title = $"Title: $(m_name.m_editText)";
    m_a.m_isEnabled = m_state != 0;
    m_b.m_isEnabled = m_useB.m_isChecked;
    m_c.m_isEnabled = m_useC.m_isChecked;
    // ...
    onevent m_button.m_onClicked ()
    {
        // ...
    }
}
```

struct ReactorBindSite

- event* m_event
- intptr m_cookie

Under the hood

class ReactorClass

methods:

- start (…)
- stop ()

fields:

- int m_state
- ReactorBindSite m_bindSite [0]
- ReactorBindSite m_bindSite [1]
- ReactorBindSite m_bindSite [2]
- …
- int m_reactionState [0]
- int m_reactionState [1]
- int m_reactionState [2]
- …

```
reactor m_uiReactor ()
{
 #1 m_title = $"Title: $(m_name.m_editText)";
 #2 m_a.m_isEnabled = m_state != 0;
 #3 m_b.m_isEnabled = m_useB.m_isChecked;
 #4 m_c.m_isEnabled = m_useC.m_isChecked;
    // ...
 #5 onevent m_button.m_onClicked ()
    {
        // ...
    }
}
```

struct ReactorBindSite

- event* m_event
- intptr m_cookie

reactions >= # bind sites

OK to return addresses of locals

```
int* foo ()
{
    int a = 10;

    // ...

    return &a; // no problem
}
```

Multicasts

```
foo (int x);
bar (int x);

baz ()
{
    multicast m (int);

    m += foo;
    m += bar;
    m (100);  // <-- foo (100); bar (100);
}
```

Properties

```
int property g_simpleProp;
```

```
int g_simpleProp.get ()
{
    // ...
}

g_simpleProp.set (int x)
{
    // ...
}
```

```
foo ()
{
    g_simpleProp = 10;     // g_simpleProp.set is called
    int x = g_simpleProp; // g_simpleProp.get is called
}
```