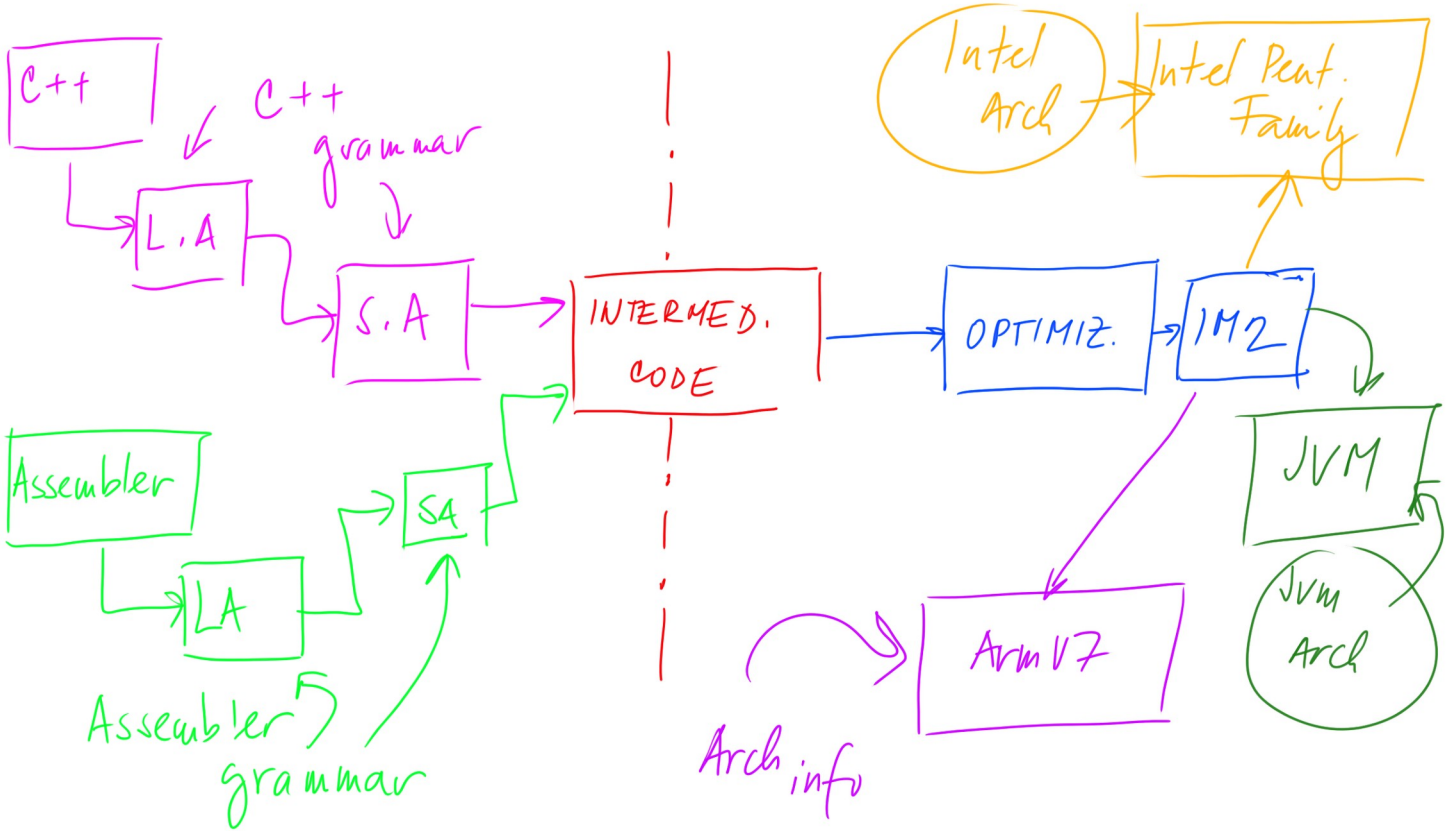
# NTI/PRK
# Intermediate languages and gcc as a use-case

Lenka Kosková Třísková, lenka.koskova.triskova@tul.cz,
LS 2024

# Intermediate code



# Type validation

- Attribute grammar with type identification
  - A special form of a grammar with type rules
- In the derivation tree, each node has associated type

https://www.cs.csub.edu/~melissa/cs350-f15/notes/notes04.html

# Structured type equivalence

```
struct person {
        char *name;
        int     yearOfBirth;
}
```

```
struct individual {
        char * IndName;
        int BirthYear;
}
```

Structure is equivalent.

Naming is different.

# Control flow diagramm

A basic block sequence of consecutive statements, without branching or loops.

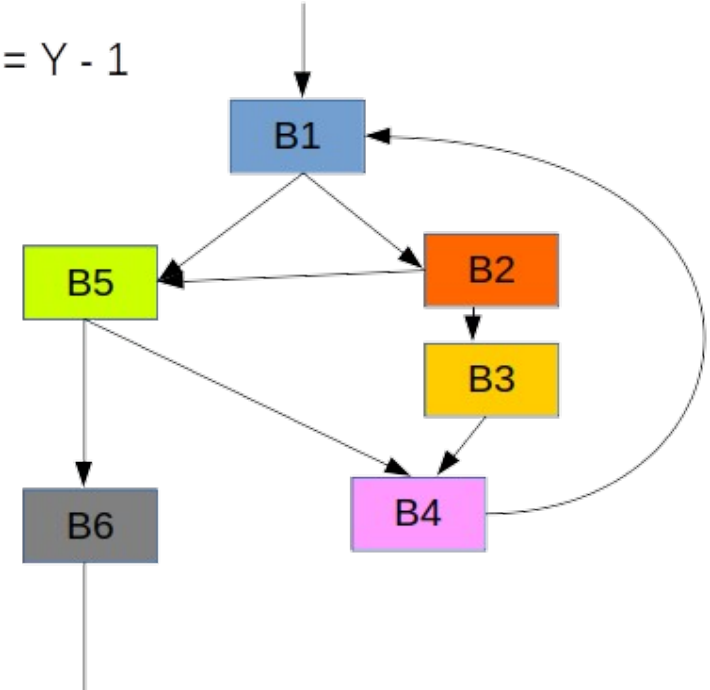
A program control flow graph is an oriented graph (B, H, s), where: B is the set of blocks, H is the set of edges, and s is a mapping BxB → H such that: s(h)={Bi,Bj} just if Bi is a potential successor of Bj in terms of program control flow.

```
while (((A<B)&(C<D)) or (X==Y))
```

# Příklad

Originální kód:

```
while (A<B) and (C<D) or (X=Y):
    A = A +1
X = Y - 1
```




Tříadresní kód

```
1: if (A >B) goto (4)
2: if (C >D) goto (4)
3: goto 5
4: if X<>Y goto 8
5: T1 = A + 1
6: A = T1
7: goto 1
8: T2 = Y – 1
9: X = T2
```

# Intermediate code - design goal

Preservation of structure (validity ranges, flow control, expressions, etc.) and semantic meaning.

Shading of the original syntax - conversion to a "universal language". Syntax does not have to be "human readable".

Preparation for generating machine code or other output.

There may even be several different ones.

Approach:

Tree structure vs. Three-address code

# Tree structure

The information is stored in a tree structure.

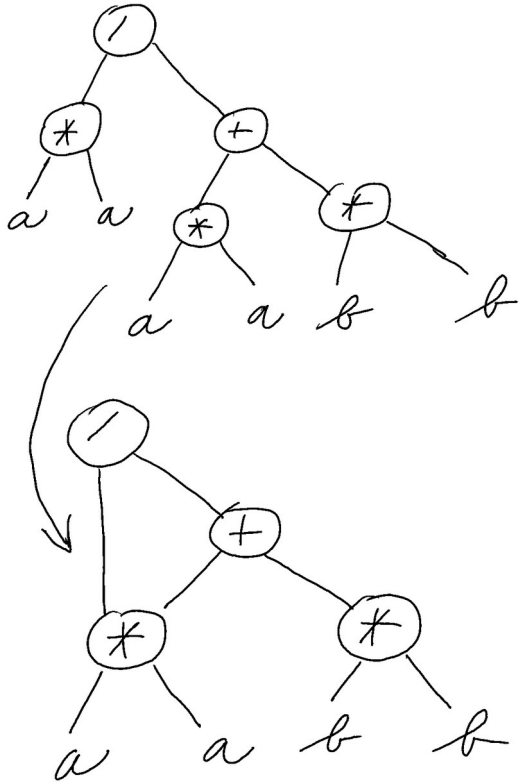
It is often used for code optimization.

Example: AST: abstract syntax tree - described standard for GCC.

# Tree edits

a *x* a / (a *x* a + b *x* b)

Subtree replacement



# Three Address Code (Linear Representation)

The code is very close to assembly - it can be viewed as virtual machine code.
Mathematical operations rewritten for registers => expressions only combinations of two operators.
Always longer, always more variables.

```
Original:

x = y*3 + z;
```

```
TAC:

t1 = y * 3;

t2 = t1 + z;
```

# Static single assignment - helpful for control flow analysis

```
Orignal:

x = a + b

y = x + 1

x = b + c

z = x + y
```

```
SSA edit:

x1 = a + b

y1 = x1 + 1

x2 = b + c

z = x2 + y1
```

# Internal gcc structure

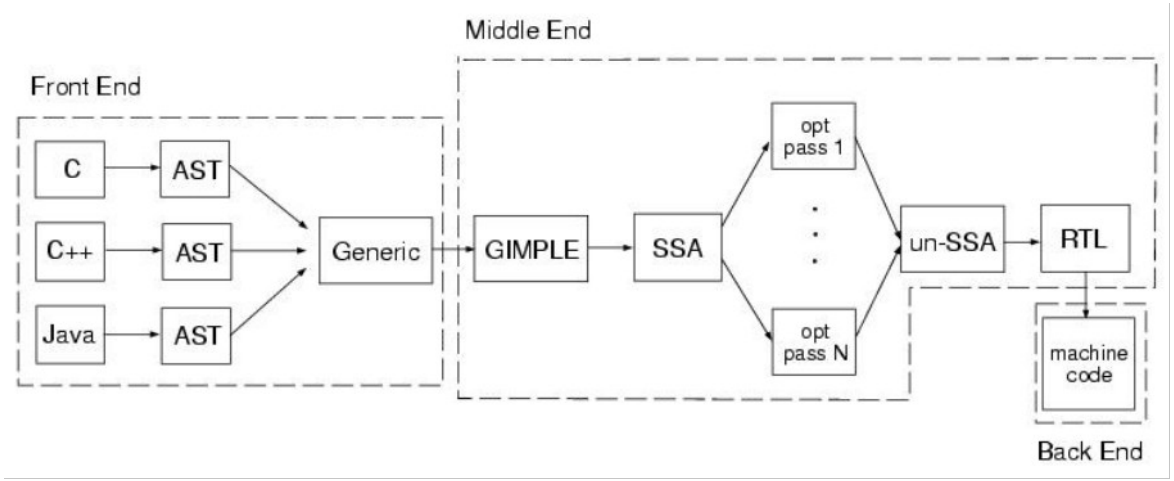
AST – abstract syntax tree

GENERIC – syntaktický strom

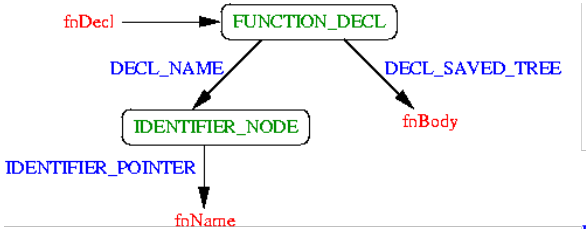
GIMPLE – zjednodušená podmnožina GENERIC užitá k optimalizaci

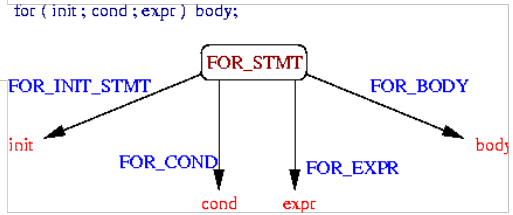
SSA – static simple assignment

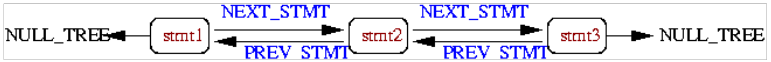
RTL – register transfer language



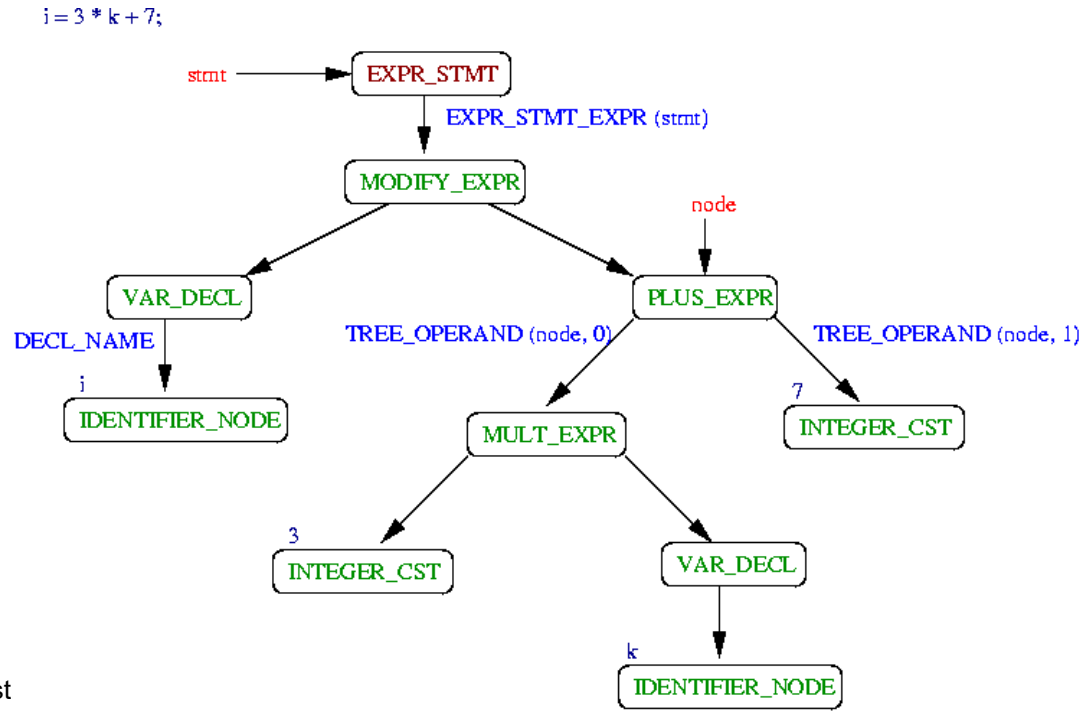
# AST - Example


fnDecl
FUNCTION_DECL
DECL_NAME
DECL_SAVED_TREE
IDENTIFIER_NODE
fnBody
IDENTIFIER_POINTER
fnName


for ( init ; cond ; expr ) body;
FOR_STMT
FOR_INIT_STMT
FOR_BODY
init
FOR_COND
FOR_EXPR
body
cond
expr


NULL_TREE
stmt1
NEXT_STMT
PREV_STMT
stmt2
NEXT_STMT
PREV_STMT
stmt3
NULL_TREE

# AST – Examples




Obrázky z: http://icps.u-strasbg.fr/~pop/gcc-ast

# Gimple - Example

```
struct A { A(); ~A(); };
     int i;
     int g();
     void f()
     {
       A a;
       int j = (--i, i ? 0 : 1);

       for (int x = 42; x > 0; --x)
         {
           i += g()*4 + 32;
         }
     }
```

```
void f()
     {
       int i.0;
       int T.1;
       int iftmp.2;
       int T.3;
       int T.4;
       int T.5;
       int T.6;

       {
         struct A a;
         int j;

         __comp_ctor (&a);
         try
           {
             i.0 = i;
             T.1 = i.0 - 1;
             i = T.1;
             i.0 = i;
```

# RTL

```
#include <stdio.h>

int
main (void)
{
  printf ("Hello, world!\n");
  return 0;
}
```

```
(note 1 0 3 NOTE_INSN_DELETED)
(note 3 1 18 2 [bb 2] NOTE_INSN_BASIC_BLOCK)
(insn/f 18 3 19 2 (set (mem:DI (pre_dec:DI (reg/f:DI 7
sp)) [0  S8 A8])
        (reg/f:DI 6 bp)) hello.c:5 65 {*pushdi2_rex64}
     (nil))
(insn/f 19 18 20 2 (set (reg/f:DI 6 bp)
        (reg/f:DI 7 sp)) hello.c:5 89 {*movdi_internal}
     (nil))
(note 20 19 2 2 NOTE_INSN_PROLOGUE_END)
(note 2 20 5 2 NOTE_INSN_FUNCTION_BEG)
(insn 5 2 6 2 (set (reg:DI 5 di)
        (symbol_ref/f:DI ("*.LC0") [flags 0x2]
<var_decl 0x7fc588b8db40 *.LC0>)) hello.c:6 89
{*movdi_internal}
     (nil))
(call_insn 6 5 7 2 (set (reg:SI 0 ax)
        (call (mem:QI (symbol_ref:DI ("puts") [flags
0x41]  <function_decl 0x7fc588ae2360 __builtin_puts>)
[0 __builtin_puts S1 A8])
            (const_int 0 [0]))) hello.c:6 669
{*call_value}
```