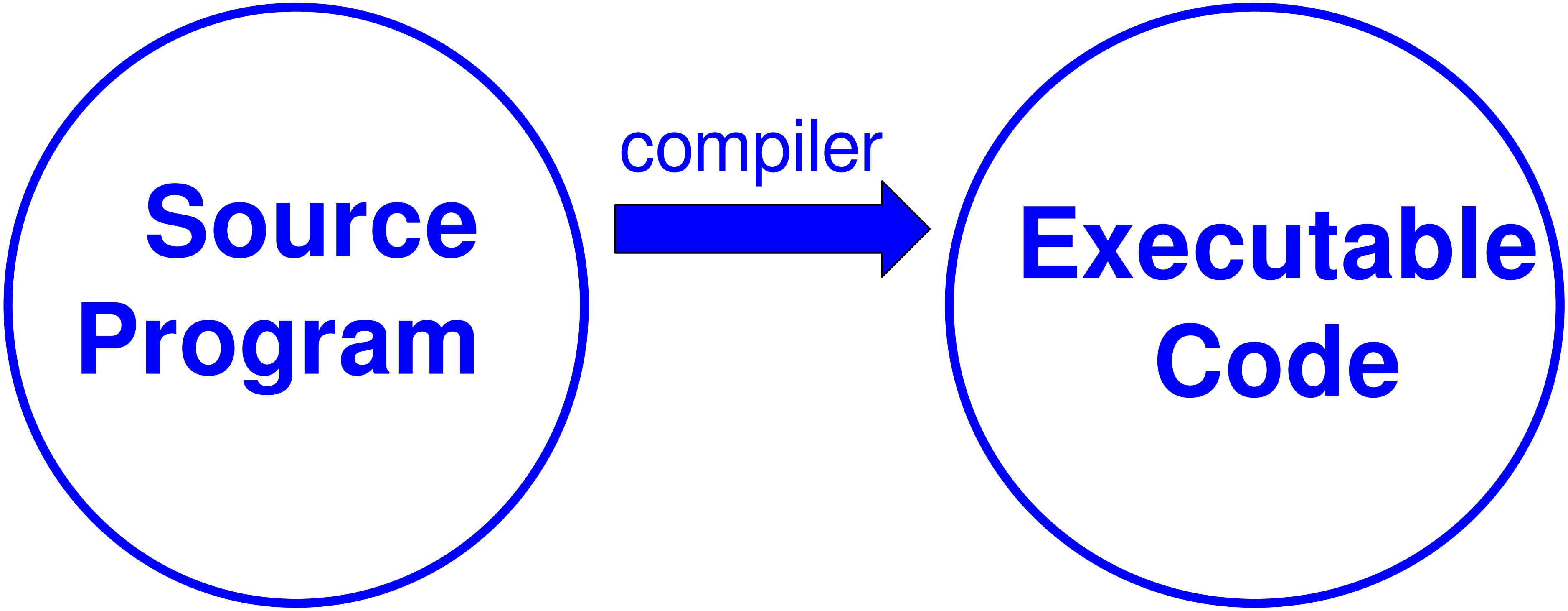
FORMAL LANGUAGES AND COMPILERS

Paola Quaglia

Goals & Expected Learning

- Concepts, techniques, and algorithms for the implementation of compilers
- Formal languages
- Recognition of formal languages



Source Program

compiler

Executable Code

A long way to go

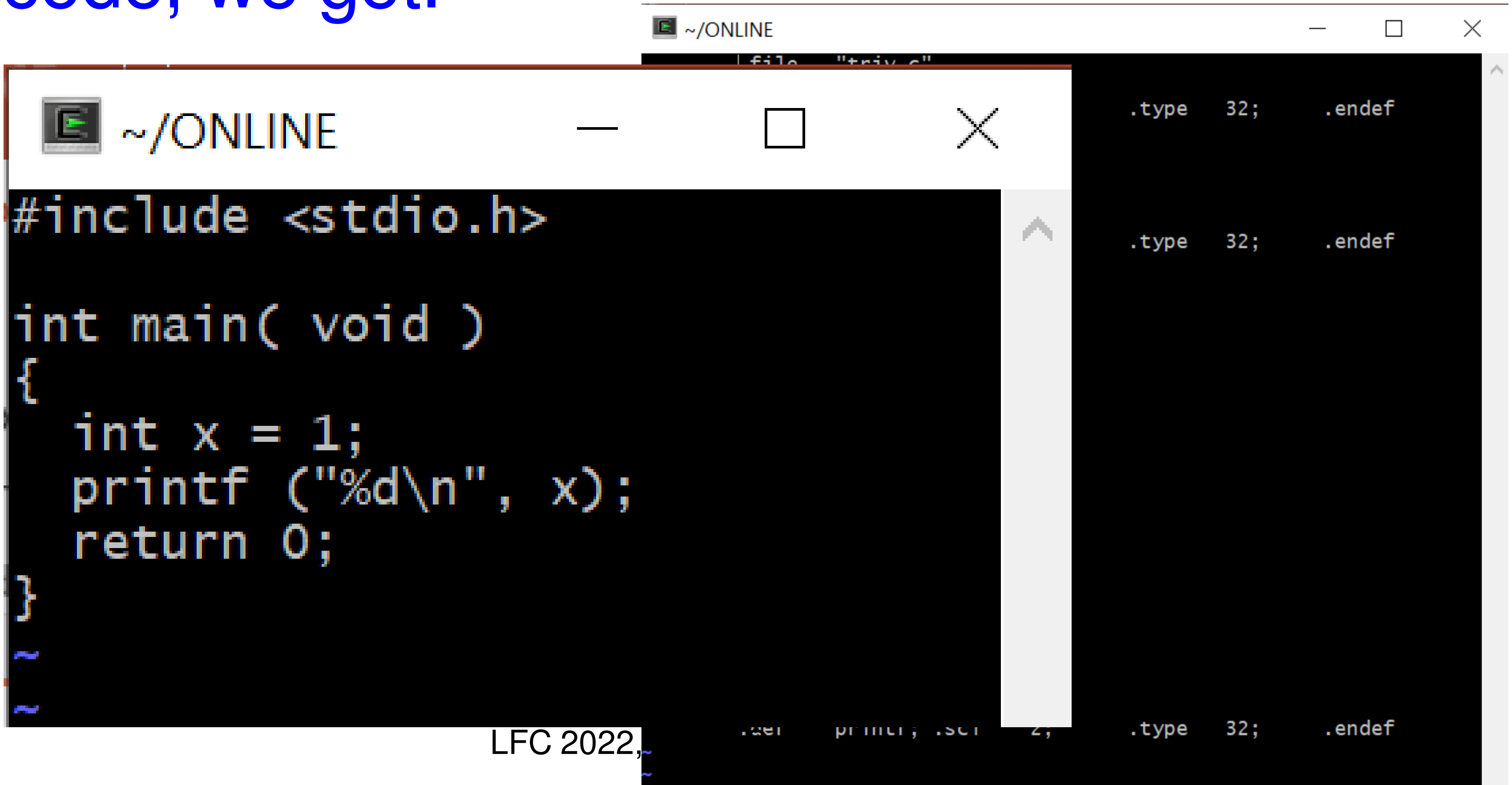
A long way to go

- E.g., the development of gcc has been going on for more than 20 years by now

- Some years ago it already consisted of 2++ million lines of code

A long way to go

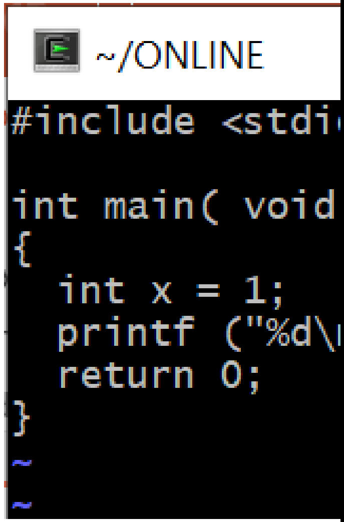
E.g., from source program to assembly code, we get:



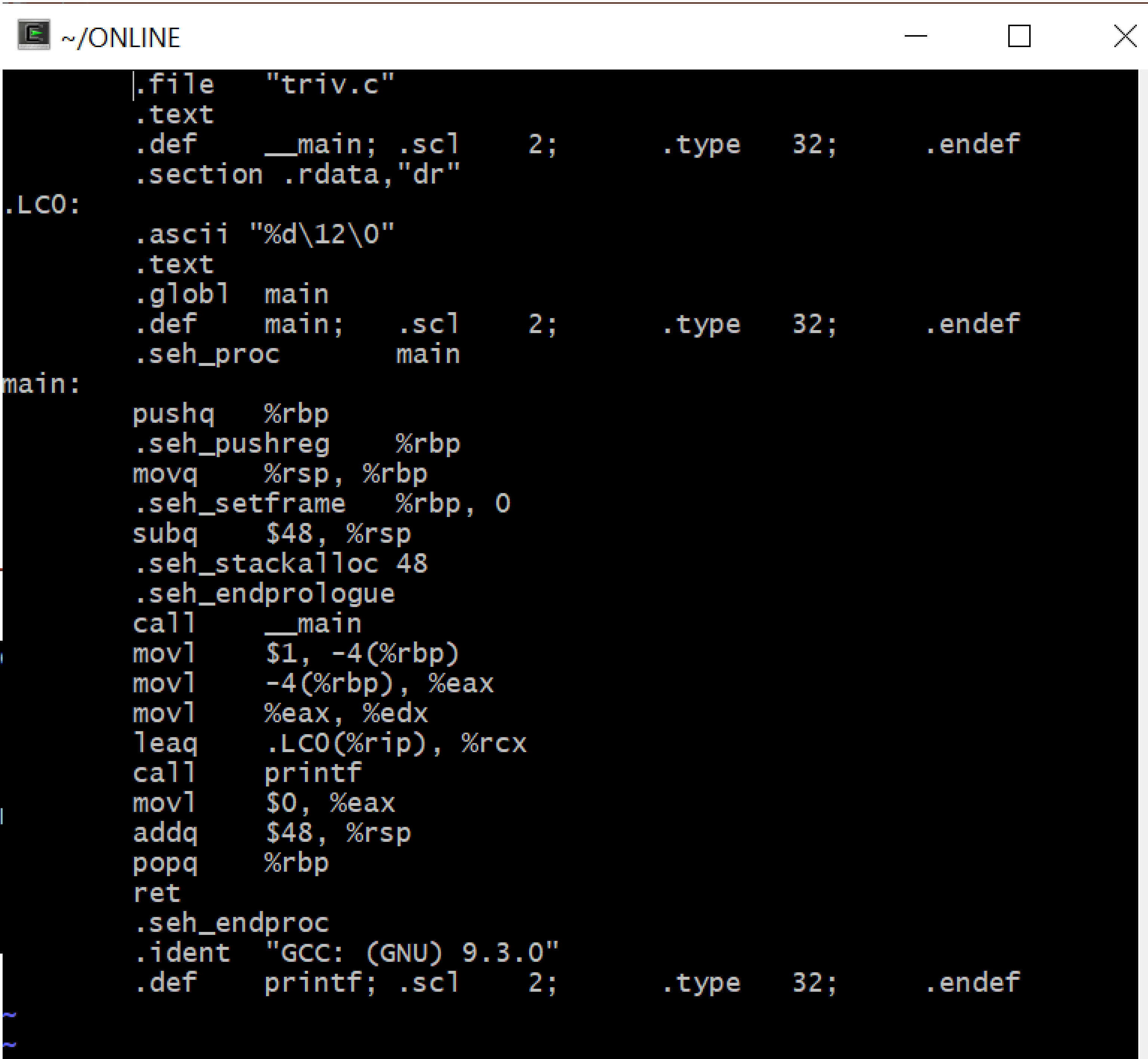
```
#include <stdio.h>

int main( void )
{
  int x = 1;
  printf ("%d\n", x);
  return 0;
}
~
~
```

```
        .file   "triv.c"
                                .type   32;     .endef

                                .type   32;     .endef

        .def    printf; .scl    2;      .type   32;     .endef
~
~
```

E.g.,



```
#include <stdi

int main( void
{
  int x = 1;
  printf ("%d\
  return 0;
}
~
~
```



```
        .file   "triv.c"
        .text
        .def    __main; .scl    2;      .type   32;     .endef
        .section .rdata,"dr"
.LC0:
        .ascii "%d\12\0"
        .text
        .globl  main
        .def    main;   .scl    2;      .type   32;     .endef
        .seh_proc       main
main:
        pushq   %rbp
        .seh_pushreg    %rbp
        movq    %rsp, %rbp
        .seh_setframe   %rbp, 0
        subq    $48, %rsp
        .seh_stackalloc 48
        .seh_endprologue
        call    __main
        movl    $1, -4(%rbp)
        movl    -4(%rbp), %eax
        movl    %eax, %edx
        leaq    .LC0(%rip), %rcx
        call    printf
        movl    $0, %eax
        addq    $48, %rsp
        popq    %rbp
        ret
        .seh_endproc
        .ident  "GCC: (GNU) 9.3.0"
        .def    printf; .scl    2;      .type   32;     .endef
~
~
```

A long way to go

- And assembly code it not yet executable

- Still needed:
  - Conversion of textual instructions to binary instructions
  - Linking to include libraries

Various phases in pipe line

- What we do have:
  - The program is written in a language which “adheres” to a specific grammar
  - Some programs are syntactically legal, some are not
  - The grammar lets us decide on this

Various phases in pipe line

```
pippo = 1;
pluto = 2;
```

```
while = 1
pluto = 2;
```

In most languages a keyword, cannot be used as an identifier

OK

NOK

The grammar

- States the form of an assignment
  - **IDENTIFIER ASSIGN EXPRESSION SEMICOLON**

- Cannot know “pippo”, “pluto”, nor the infinitely many strings we may use as variables

Lexical analysis

- Translates a stream of characters into a stream of **tokens**
- E.g. translates
- **`pippo = 2*3`**;
- Into
- **`<ID,pippo> ASS <NUM,2> MUL <NUM,3> SEMCOL`**

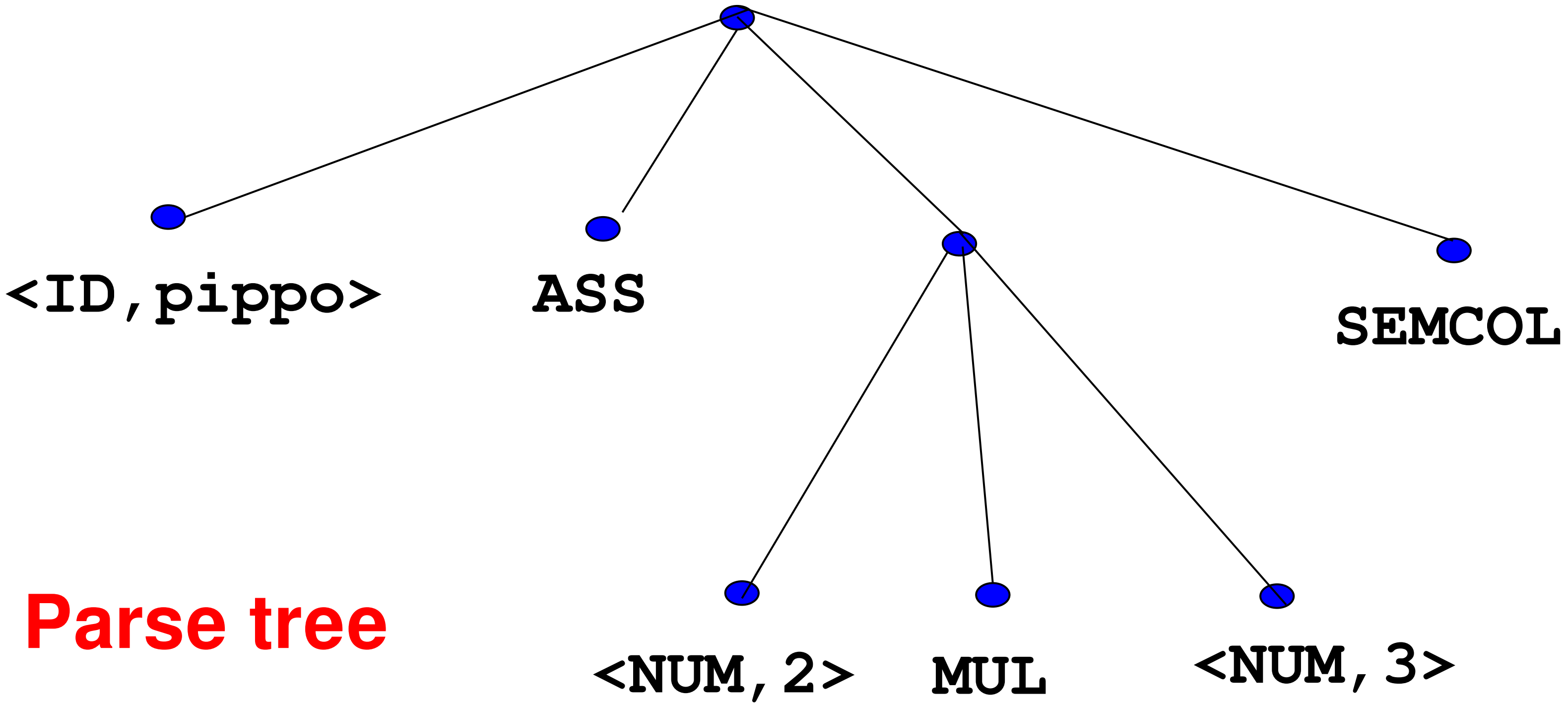
Syntax analysis

- Checks whether a stream of tokens “adheres” to the given grammar

- If so, converts the stream of tokens into a **parse tree** (or better, into an **abstract syntax tree**)

Syntax analysis

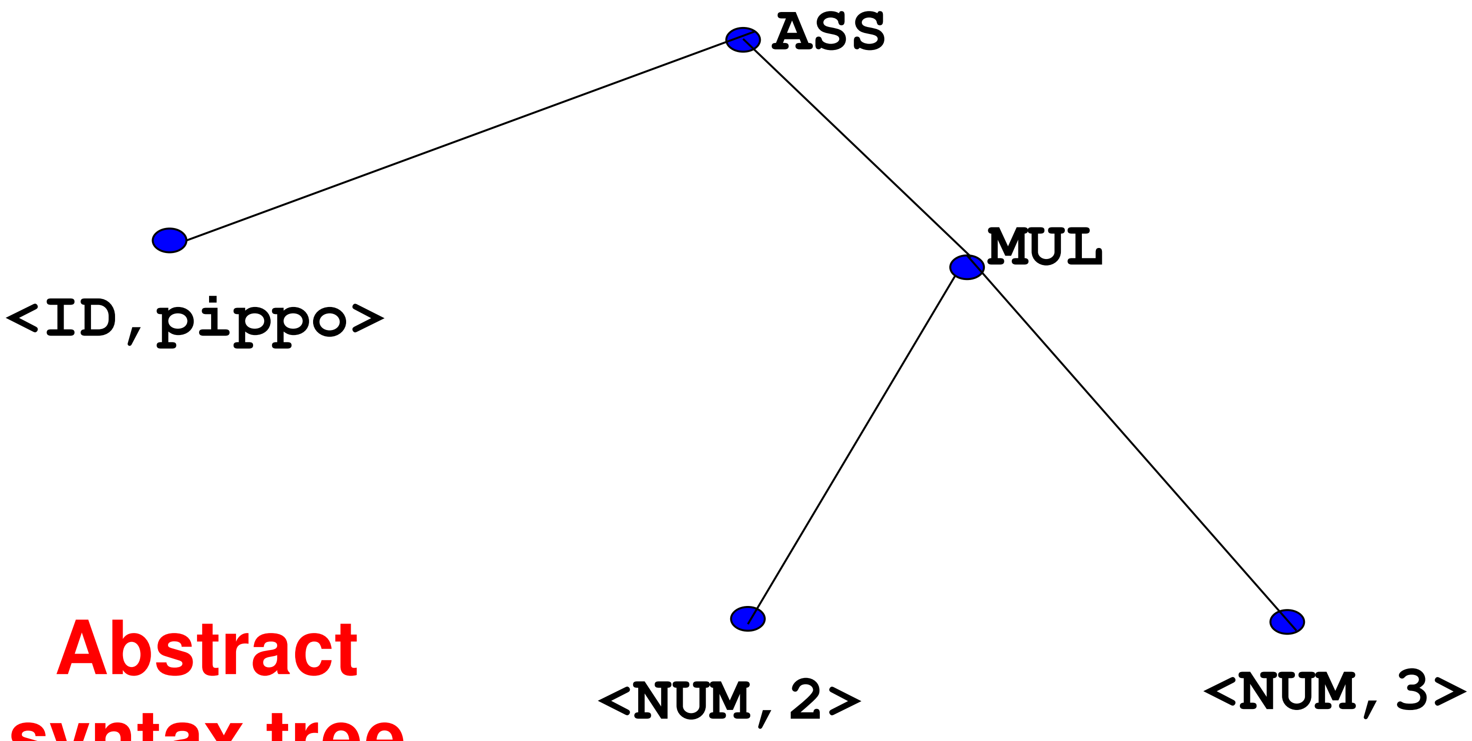
`<ID,pippo> ASS <NUM,2> MUL <NUM,3> SEMCOL`



Parse tree

Syntax analysis

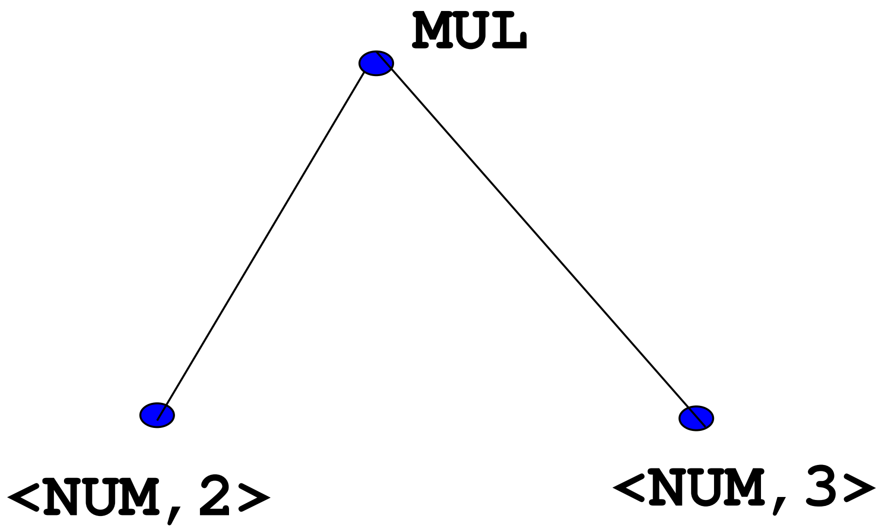
<ID,pippo> ASS <NUM,2> MUL <NUM,3> SEMCOL



Abstract syntax tree

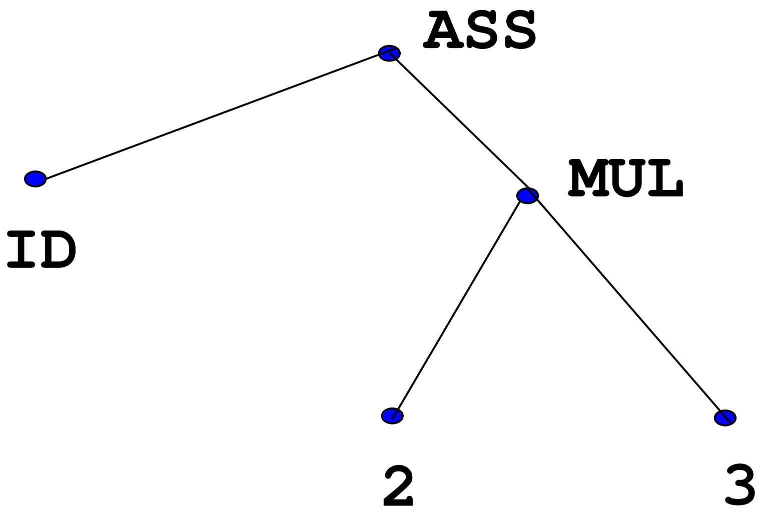
Semantic analysis

Are we actually multiplying two integers?



Intermediate code generation

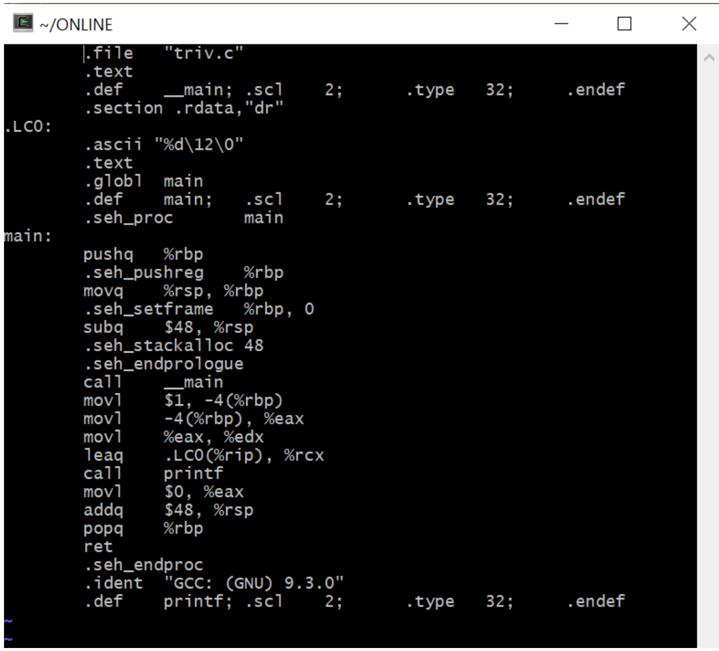
– Converts a parse tree into a textual intermediate code



```
MOVE 2,R1
MULT 3,R1
MOVE R1,ID1
```

Target code generation

- Translates intermediate code into target machine code (assembly code)



```
        .file   "triv.c"
        .text
        .def    __main; .scl    2;      .type   32;     .endef
        .section .rdata,"dr"
.LC0:
        .ascii "%d\12\0"
        .text
        .globl  main
        .def    main;   .scl    2;      .type   32;     .endef
        .seh_proc       main
main:
        pushq   %rbp
        .seh_pushreg    %rbp
        movq    %rsp, %rbp
        .seh_setframe   %rbp, 0
        subq    $48, %rsp
        .seh_stackalloc 48
        .seh_endprologue
        call    __main
        movl    $1, -4(%rbp)
        movl    -4(%rbp), %eax
        movl    %eax, %edx
        leaq    .LC0(%rip), %rcx
        call    printf
        movl    $0, %eax
        addq    $48, %rsp
        popq    %rbp
        ret
        .seh_endproc
        .ident  "GCC: (GNU) 9.3.0"
        .def    printf; .scl    2;      .type   32;     .endef
~
~
```

Plausible targets

- Machine code not necessarily the target of compilation
  - May be a virtual machine
  - May be another language

Front-end & Back-end

- Front-end
  - From lexical analysis all the way down to intermediate code generation
- Back-end:
  - All the rest
- Modularity:
  - N languages, N front-ends
  - K machines, K back-ends
  - N*K compilers, overall

That said

Goals & Expected Learning

- Pervasive structures (finite state automata)
- Methodologies at the basis of other areas, too (e.g., parsing of natural languages, soundness of queries to databases)

In a Nutshell

- Regular languages and finite state automata (lexical analysis)
- Context-free languages and parsing (syntax analysis)
- Syntax driven definitions (semantic analysis & symbol tables)

In a Nutshell (ctd.)

- Intermediate code generation
- Machine code generation
- Register allocation
- Runtime environment

Main Textbook

- Introduction to Compiler Design (Second Edition), Torben AEgidius Mogensen, Springer, 2017, 978-3319669656

  – Preliminary version available at

  http://hjemmesider.diku.dk/~torbenm/Basics/index.html

Verification

- Optional individual project +
- Written test +
- Oral test

Optional project

- Individual

- To hand in during before the January exam session

- Max 2 points, to sum up to the score of the written test

Written Test

- Closed books & No electronics
- 10 questions for open and succinct answers
  - **You study you get**
  - Questions on basic knowledge, application of known algorithms to instances of the problem
- Max 30 points

Written Test (ctd)

- Compulsory subscription on esse3
  - In case of subscription & no-show the student undergoes a skip-session
- At most 3 submissions of the written test in 2023

Oral Test & Final Evaluation

- Oral test: Max 32 points

- Final score: written and oral contribute 50% each