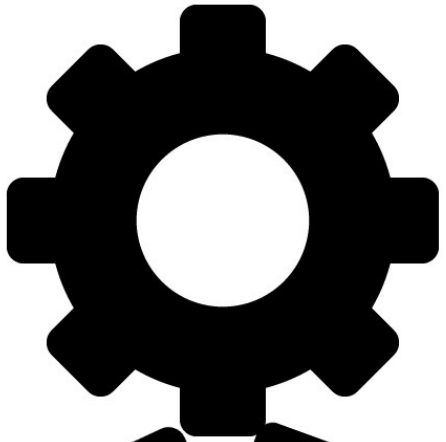
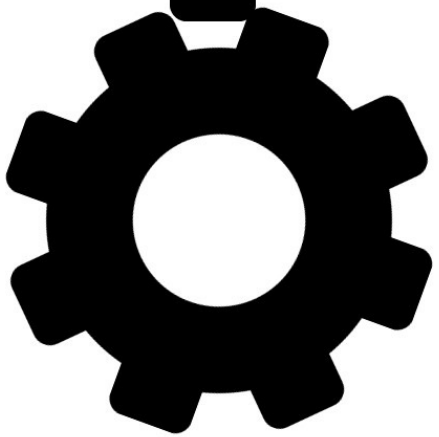
C  mpiler

C  nstruction

LC

Simone Campanoni
simone.campanoni@northwestern.edu

Outline

- LC
- Parsing
- Translating high level control structures

LC

```
p      ::= f+
f      ::= T name ( (type name)* ) scope
scope  ::= { i* }
i      ::= i1 | i2 | scope
i1     ::= name <- s | name <- t op t | name <- name([t])+ | name([t])+ <- s | name <- length name t |
           name( args? ) | name <- name( args? ) |
           name <- new Array(args) | name <- new Tuple(t)
i2     ::= type names | if (cond) scope else scope | return (t)? |
           while (cond) scope | do scope while (cond) | for (i1?; cond? ; i1?) scope | continue | break
T      ::= type | void
type   ::= int([])* | tuple | code
args   ::= t | t (, t)*
s      ::= t | name
t      ::= name | N
N      ::= (+|-)? [0-9]+
op     ::= + | - | * | & | << | >> | cmp
cmp    ::= < | <= | = | >= | >
name   ::= [a-zA-Z_][a-zA-Z_0-9]*
cond   ::= t cmp t
names  ::= name | name (, name)*
```

- No labels
- No goto
- High level control structures only (e.g., loops)

LC example 0: if then else

```
void main (){
    int index
    index <- 0
    if (index < 10) {
        index <- index + 1
    } else {
        index <- index – 1
    }
    return
}
```

LC

```
p       ::= f+
f       ::= T name ( (type name)* ) scope
scope   ::= { i* }
i       ::= i1 | i2 | scope
i1      ::= name <- s | name <- t op t | name <- name([t])+ |  name([t])+ <- s | name <- length name t |
            name( args? ) | name <- name( args? ) |
            name <- new Array(args) | name <- new Tuple(t)
i2      ::= type names | if (cond) scope else scope | return (t)? |
            while (cond) scope | do scope while (cond) | for (i1?; cond? ; i1?) scope | continue | break
T       ::= type | void
type    ::= int([])* | tuple | code
args    ::= t | t (, t)*
s       ::= t | name
t       ::= name | N
N       ::= (+|-)? [0-9]+
op      ::= + | - | * | & | << | >> | cmp
cmp     ::= < | <= | = | >= | >
name    ::= [a-zA-Z_][a-zA-Z_0-9]*
cond    ::= t cmp t
names   ::= name | name (, name)*
```

LC example 1: while loop

```
void main (){
    int index
    index <- 0
    while (index < 10) {
        index <- index + 1
    }
    return
}
```

LC

```
p       ::= f+
f       ::= T name ( (type name)* ) scope
scope   ::= { i* }
i       ::= i1 | i2 | scope
i1      ::= name <- s | name <- t op t | name <- name([t])+ |  name([t])+ <- s | name <- length name t |
            name( args? ) | name <- name( args? ) |
            name <- new Array(args) | name <- new Tuple(t)
i2      ::= type names | if (cond) scope else scope | return (t)? |
            while (cond) scope | do scope while (cond) | for (i1?; cond? ; i1?) scope | continue | break
T       ::= type | void
type    ::= int([])* | tuple | code
args    ::= t | t (, t)*
s       ::= t | name
t       ::= name | N
N       ::= (+|-)? [0-9]+
op      ::= + | - | * | & | << | >> | cmp
cmp     ::= < | <= | = | >= | >
name    ::= [a-zA-Z_][a-zA-Z_0-9]*
cond    ::= t cmp t
names   ::= name | name (, name)*
```

LC example 2: do while loop

```
void main (){
    int index
    index <- 0
    do {
        index <- index + 1
    } while (index < 10)
    return
}
```

LC

```
p       ::= f+
f       ::= T name ( (type name)* ) scope
scope   ::= { i* }
i       ::= i1 | i2 | scope
i1      ::= name <- s | name <- t op t | name <- name([t])+ |  name([t])+ <- s | name <- length name t |
            name( args? ) | name <- name( args? ) |
            name <- new Array(args) | name <- new Tuple(t)
i2      ::= type names | if (cond) scope else scope | return (t)? |
            while (cond) scope | do scope while (cond) | for (i1?; cond? ; i1?) scope | continue | break
T       ::= type | void
type    ::= int([])* | tuple | code
args    ::= t | t (, t)*
s       ::= t | name
t       ::= name | N
N       ::= (+|-)? [0-9]+
op      ::= + | - | * | & | << | >> | cmp
cmp     ::= < | <= | = | >= | >
name    ::= [a-zA-Z_][a-zA-Z_0-9]*
cond    ::= t cmp t
names   ::= name | name (, name)*
```

LC example 3: for loop

```
void main (){
    int index
    for (index <- 0; index < 10; index <- index + 1){
        myF(index)
    }
    return
}
```

Outline

- LC

- Parsing

- Translating high level control structures

LC parser almost the same as the LB one

- Only difference: you need to parse the high level control structures

```
if (index < 10) {
   index <- index + 1
} else {
   index <- index – 1
}
```

```
struct if_else_rule:
   pegtl::seq<
      str_if,
      ‘(‘, condition_rule, ‘)‘,
      scope,
      str_else,
      scope
     > {};
```

- Problem: you want to append a new LC instruction “if” before anything that is inside the two scopes

LC parser almost the same as the LB one

- Only difference: you need to parse the high level control structures

```
if (index < 10) {
   index <- index + 1
} else {
   index <- index – 1
}
```

```
struct if_else_rule:
   pegtl::seq<
      str_if,
      ‘(‘, condition_rule, ‘)‘,
      scope,
      str_else,
      scope
   > {};
```

Solution:

- Create a high-level-control-structure (HLCS) stack
- The ”if” instruction class includes pointers to the two scopes (then, else, branches)

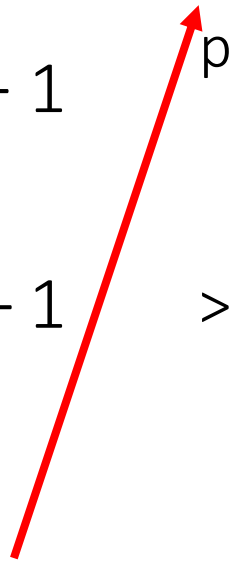
LC parser almost the same as the LB one

- Only difference: you need to parse the high level control structures

```
if (index < 10) {
   index <- index + 1
} else {
   index <- index – 1
}
```

```
struct if_else_begin_rule:
  pegtl::seq<
    str_if,
    '(', condition_rule, ')',
  > {};
```

```
struct if_else_rule:
  pegtl::seq<
    if_else_begin_rule,
    scope,
    str_else,
    scope
  > {};
```



- Create an “if” instruction and append it to the current scope
- Push the “if” instruction just created on top of the HLCS stack

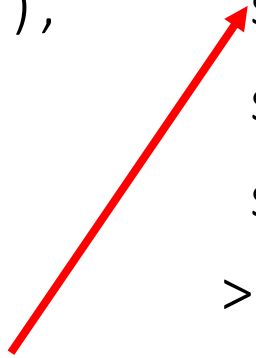
LC parser almost the same as the LB one

- Only difference: you need to parse the high level control structures

```
if (index < 10) {
   index <- index + 1
} else {
   index <- index – 1
}
```

```
struct if_else_begin_rule:
  pegtl::seq<
    str_if,
    ‘(‘, condition_rule, ‘)‘,
  > {};
```

```
struct if_else_rule:
  pegtl::seq<
    if_else_begin_rule,
    scope,
    str_else,
    scope
  > {};
```



At “{“:

- Check if the opening scope should be attached to an “if” instruction:
  - Is there an “if” on the HLCS stack?
- If no: append the scope to the parent scope

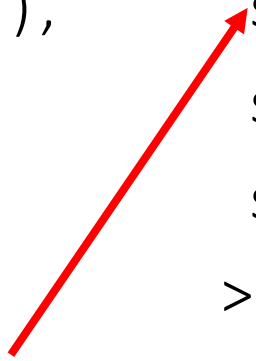
LC parser almost the same as the LB one

- Only difference: you need to parse the high level control structures

```
if (index < 10) {
   index <- index + 1
} else {
   index <- index – 1
}
```

```
struct if_else_begin_rule:
  pegtl::seq<
    str_if,
    ‘(‘, condition_rule, ‘)‘,
  > {};
```

```
struct if_else_rule:
  pegtl::seq<
    if_else_begin_rule,
    scope,
    str_else,
    scope
  > {};
```



At “{“:

- Check if the opening scope should be attached to an “if” instruction:
  - If yes: attach the opening scope to the “if” on top of the HLCS stack
  - If that “if” has both branches with attached scopes, then pop it from the HLCS stack

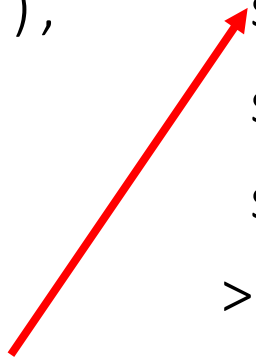
LC parser almost the same as the LB one

- Only difference: you need to parse the high level control structures

```
if (index < 10) {
   index <- index + 1
} else {
   index <- index – 1
}
```

```
struct if_else_begin_rule:
  pegtl::seq<
    str_if,
    ‘(‘, condition_rule, ‘)‘,
  > {};
```

```
struct if_else_rule:
  pegtl::seq<
    if_else_begin_rule,
    scope,
    str_else,
    scope
  > {};
```



Attaching the opening scope to the “if” on top of the HLCS stack

- If the top “if” doesn’t have a scope on the ”then” branch: attach the opening scope to the “then” branch

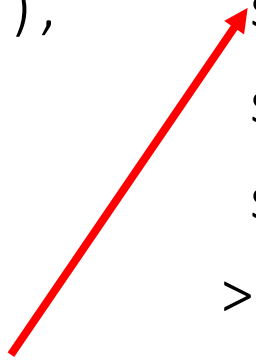
LC parser almost the same as the LB one

- Only difference: you need to parse the high level control structures

```
if (index < 10) {
   index <- index + 1
} else {
   index <- index – 1
}
```

```
struct if_else_begin_rule:
  pegtl::seq<
    str_if,
    ‘(‘, condition_rule, ‘)‘,
  > {};
```

```
struct if_else_rule:
  pegtl::seq<
    if_else_begin_rule,
    scope,
    str_else,
    scope
  > {};
```



Attaching the opening scope to the “if” on top of the HLCS stack

- If the top “if” has a scope on the ”then” branch:
  attach the opening scope to the “else” branch

Outline

- LC

- Parsing

- Translating high level control structures

Translation of the LC ”if” to LB code

- Create 3 new LB labels: :LT, :LF, :LE
- Translate the LC condition to LB code and append the LB code to the current innermost scope
- Append the :LT label
- Translate the “then” scope
- Append a jump to :LE
- Append the :LF label
- Translate the “else” scope
- Append the :LE label

```
if (index < 10) {
   index <- index + 1
} else {
   index <- index – 1
}
```

```
if (index < 10) :LT :LF
:LT
…
br :LE
:LF
…
:LE
```

Homework #8

Write a compiler that translates an LC program (.c) to an LB one

- You need to generate prog.b
- You need to pass all tests in the framework

Always have faith in your ability

Success will come your way eventually

Best of luck!