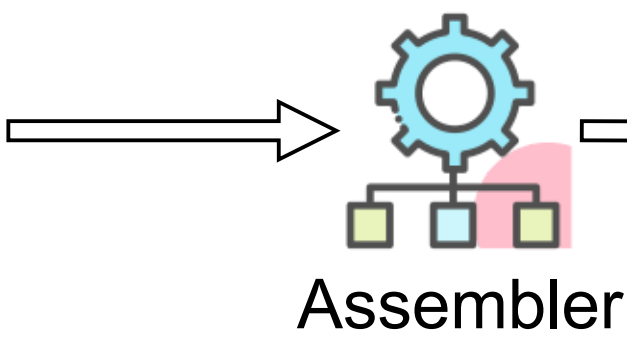


```
// Adds 1 + ... + 100
@i
M=1 // i=1
@sum
M=B // sum=0
(LOOP)
@i
D=M // D=i
@iB0
D=D-A // D=i-100
@END
D;JGT // if (i-100)>0 goto END
@i
D=M // D=i
@sum
M=D+M // sum=sum+i
@i
M=M+1 // i=i+1
@LOOP
0;JMP // goto LOOP
(END)
@END
0;JMP // infinite loop
```

Assembly code



Assembler

```
110101010000000
001100110010000
110111110010000
001100110010000
110101010000100
001100110010000
111110000010010
000000001100100
110010011010100
000000000010011
110001100000001
001001100100000
111110000010000
001100110010000
111110111001000
000000000000101
110101010000111
000000000021001
110101010000111
```

Machine code

Pre-pass: Read source and Clean-up codes

- Read the ASM code form file.
- Remove comments (// and all stuff after it until a EOL '\0')
- Remove Spaces
- Save the cleaned-up code into a queue (FIFO buffer).
- Initialize symbol table (e.g., pre-allocate R0-R15).

Label	RAM address	(hexa)
SP	0	0x0000
LCL	1	0x0001
ARG	2	0x0002
THIS	3	0x0003
THAT	4	0x0004
R0-R15	0-15	0x0000-0x000F
SCREEN	16384	0x4000
KBD	24576	0x6000

```
// Adds 1 + ... + 100
@i
M=1 // i=1
@sum
M=B // sum=0
(LOOP)
@i
D=M // D=i
@iB0
D=D-A // D=i-100
@END
D;JGT // if (i-100)>0 goto END
@i
D=M // D=i
@sum
M=D+M // sum=sum+i
@i
M=M+1 // i=i+1
@LOOP
@LOOP
0;JMP // goto LOOP
(END)
@END
0;JMP // infinite loop
```

.asm file

```
@i
M=1
@sum
M=B
(LOOP)
@i
D=M
@iB0
D=D-A
@END
D;JGT
@i
D=M
@sum
M=D+M
@i
M=M+1
@LOOP
@LOOP
0;JMP
(END)
@END
0;JMP
```

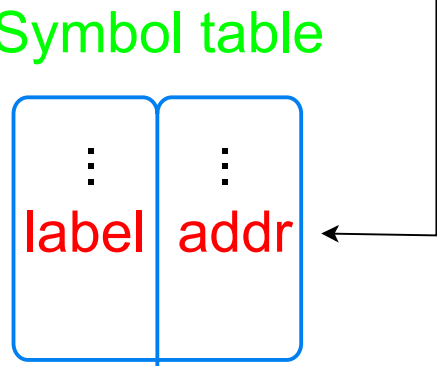
queue1



Symbol Table
<ul style="list-style-type: none">• SP : 0• LCL :1• ARG :2• THIS :3• THAT :4• R0 :0
...

First-pass: Build symbol Table

- Count lines and find labels "(Xxx)".
- The label has the same ROM address as the instruction next (below) to it.
- Xxx can not begin with a number.
- Add the XXX label and its corresponding ROM address to the symbol table.
- Erase the (Xxx) line.
- Save the processed code into a queue.



```
@i
M=1
@sum
M=B
(LOOP)
@i
D=M
@iB0
D=D-A
@END
D;JGT
@i
D=M
@sum
M=D+M
@i
M=M+1
@LOOP
@LOOP
0;JMP
(END)
@END
0;JMP
```

queue1

```
@i
M=1
@sum
M=B
(LOOP)
@i
D=M
@iB0
D=D-A
@END
D;JGT
@i
D=M
@sum
M=D+M
@i
M=M+1
@LOOP
@LOOP
0;JMP
(END)
@END
0;JMP
```

queue2

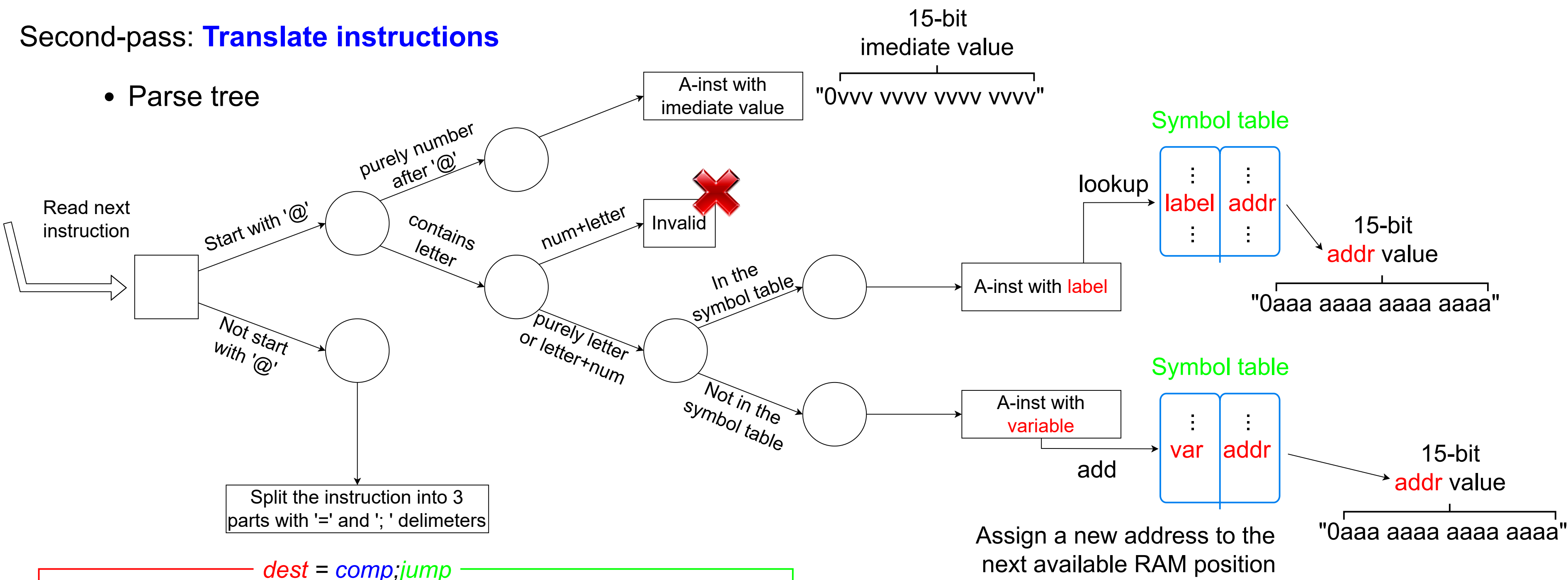


Symbol Table
<ul style="list-style-type: none">• SP : 0• LCL :1• ARG :2• THIS :3• THAT :4• R0 :0
...
<ul style="list-style-type: none">• LOOP : 5• END : 19

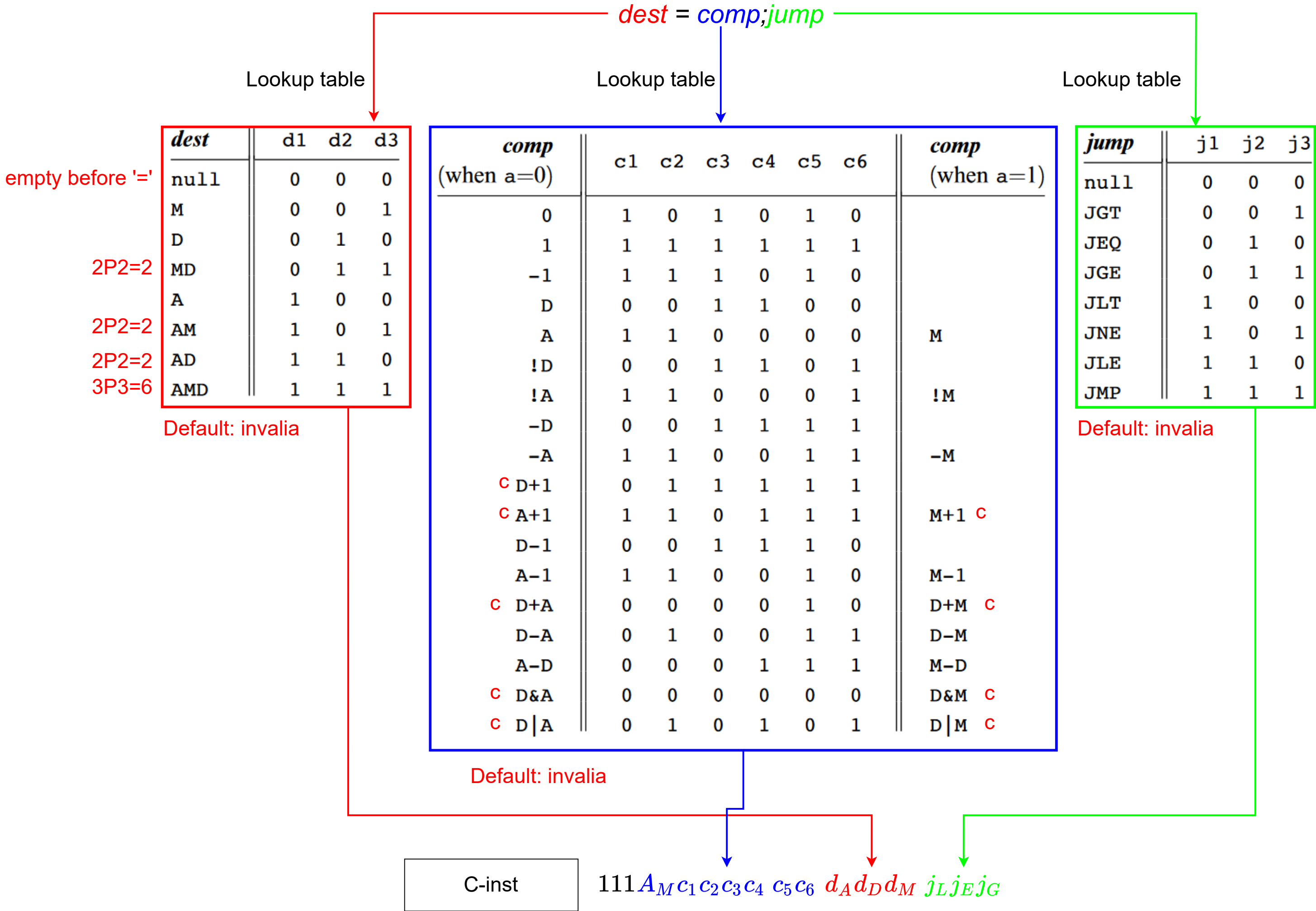
```
@i
M=1
@sum
M=B
(LOOP)
@i
D=M
@iB0
D=D-A
@END
D;JGT
@i
D=M
@sum
M=D+M
@i
M=M+1
@LOOP
@LOOP
0;JMP
(END)
@END
0;JMP
```

Second-pass: Translate instructions

- Parse tree



Assign a new address to the next available RAM position

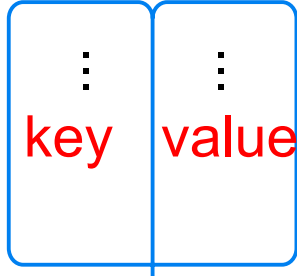


```
// Adds 1 + ... + 100
@i
M=1 // i=1
@sum
M=B // sum=0
(LOOP)
@i
D=M // D=i
@iB0
D=D-A // D=i-100
@END
D;JGT // if (i-100)>0 goto END
@i
D=M // D=i
@sum
M=D+M // sum=sum+i
@i
M=M+1 // i=i+1
@LOOP
@LOOP
0;JMP // goto LOOP
(END)
@END
0;JMP // infinite loop
```

Implementation: C++

- I know C++.
- C++ is Object-oriented
- C++ supports STL e.g., map and queue.

C++ std::map



Symbol table

