Final Project

Peng-Sheng Chen

Department of Computer Science
National Chung Cheng University

Example

- Simple expression calculator: evaluate expressions containing +, -, * and variable assignments.
- Reference from http://www.antlr.org/wiki/display/ANTLR3/Expression+evaluator

```
$
x=1
y=2
3*(x+y) <EOF>
9
$
```

Tokens in Lexer

```
ID  :   ('a'..'z'|'A'..'Z')+ ;
INT :   '0'..'9'+ ;
NEWLINE:'\r'? '\n' ;
WS  :   (' '|'\t')+ {skip();} ;
```

```
$
x=1
y=2
3*(x+y) <EOF>
9
$
```

Grammar in Parser (1)

```
prog:   stat+ ;

stat:   expr NEWLINE
    |   ID '=' expr NEWLINE
    |   NEWLINE
    ;

expr
    :   multExpr
        (   '+' multExpr
        |   '-' multExpr
        )*
    ;
```

Grammar in Parser (2)

```
multExpr
    :   atom ('*' atom)*
    ;

atom
    :   INT
    |   ID
    |   '(' expr ')'
    ;
```

Actions in Parser (1)

The @members section is where you place instance variables and methods that will be placed and used in the generated parser

```
grammar Expr;

@header {
import java.util.HashMap;
}

@members {
/** Map variable name to Integer object holding value
*/
HashMap memory = new HashMap();
}

prog:   stat+ ;
```

Actions in Parser (2)

```
prog:   stat+ ;

stat:   expr NEWLINE
        { System.out.println($expr.value); }
    |   ID '=' expr NEWLINE
        { memory.put($ID.text,
                     new Integer($expr.value));}
    |   NEWLINE
    ;
```

Actions in Parser (3)

```
expr returns [int value]
     :   multExpr
         (   '+' multExpr
         |   '-' multExpr
         )*
     ;
```

Actions in Parser (4)

```
expr returns [int value]
    :   a=multExpr {$value = $a.value;}
        (   '+' b=multExpr {$value += $b.value;}
        |   '-' c=multExpr {$value -= $c.value;}
        )*
    ;
```

Actions in Parser (5)

```
multExpr returns [int value]
    :   a=atom {$value = $a.value;}
        ('*' b=atom {$value *= $b.value;})*
    ;
```

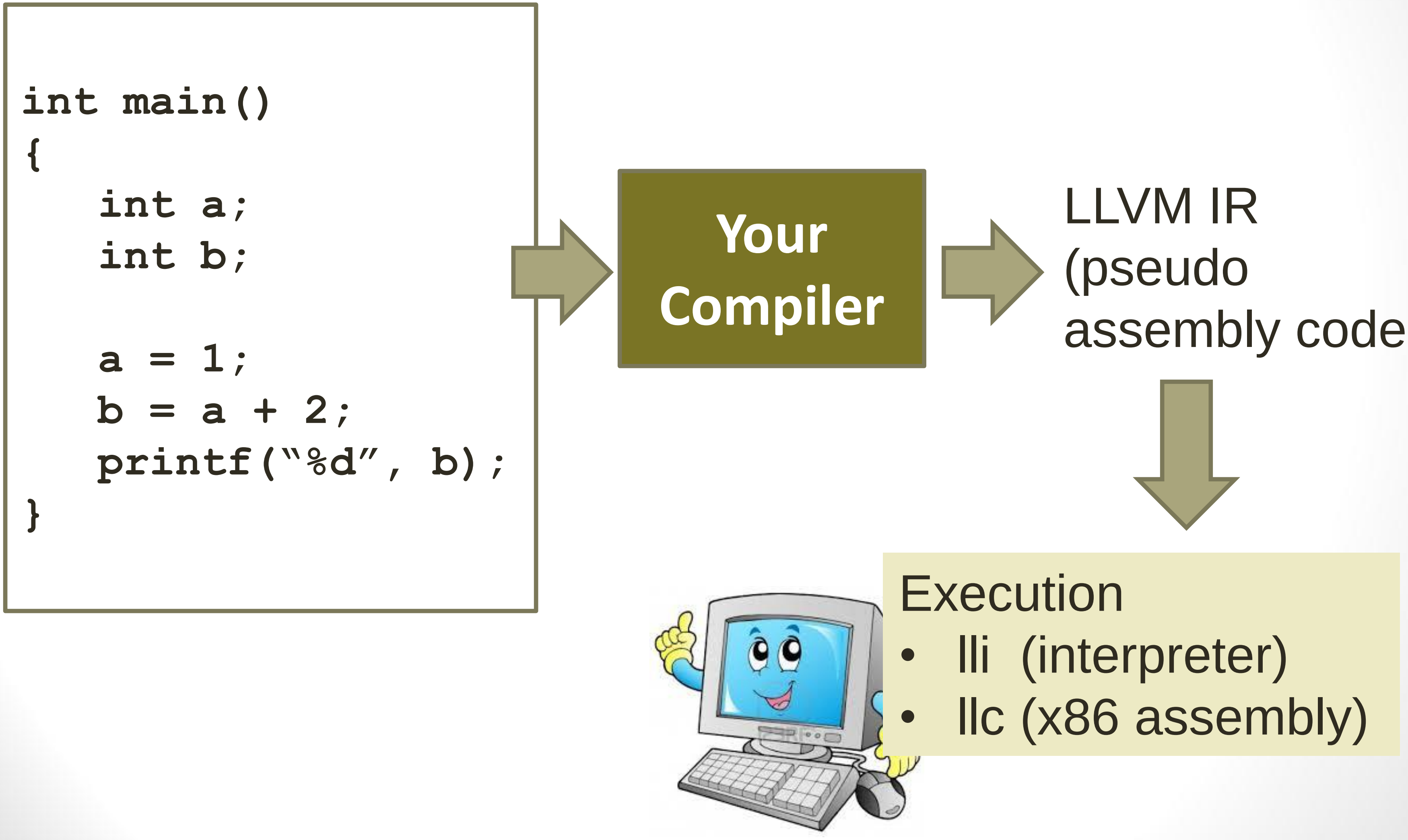
Actions in Parser (6)

```
atom returns [int value]
    : INT {$value = Integer.parseInt($INT.text);}
    | ID
      {
      Integer v = (Integer)memory.get($ID.text);
      if (v != null)
         $value = v.intValue();
      else
         System.err.println("undefined var: "+$ID.text);
      }
    | '(' expr ')’ {$value = $expr.value;}
    ;
```

Test Class

```
import org.antlr.runtime.*;


public class TestExpr {
   public static void main(String[] args)
   {
      CharStream input = new ANTLRFileStream(args[0]);
      ExprLexer lexer = new ExprLexer(input);
      CommonTokenStream tokens = new
CommonTokenStream(lexer);

      ExprParser parser = new ExprParser(tokens);
      parser.prog();
    }
}
```

Construct Your Compiler



```
int main()
{
   int a;
   int b;

   a = 1;
   b = a + 2;
   printf(“%d”, b);
}
```

Your Compiler

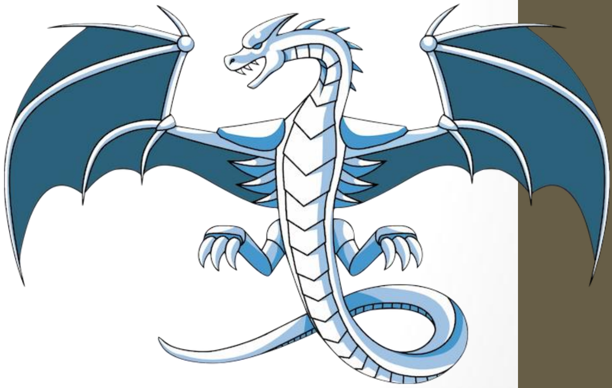
LLVM IR (pseudo assembly code

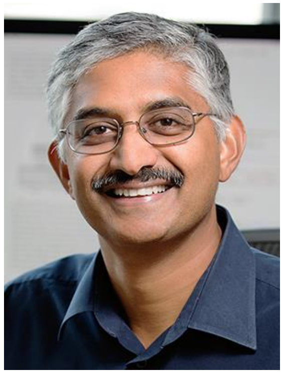
Execution

- lli (interpreter)
- llc (x86 assembly)

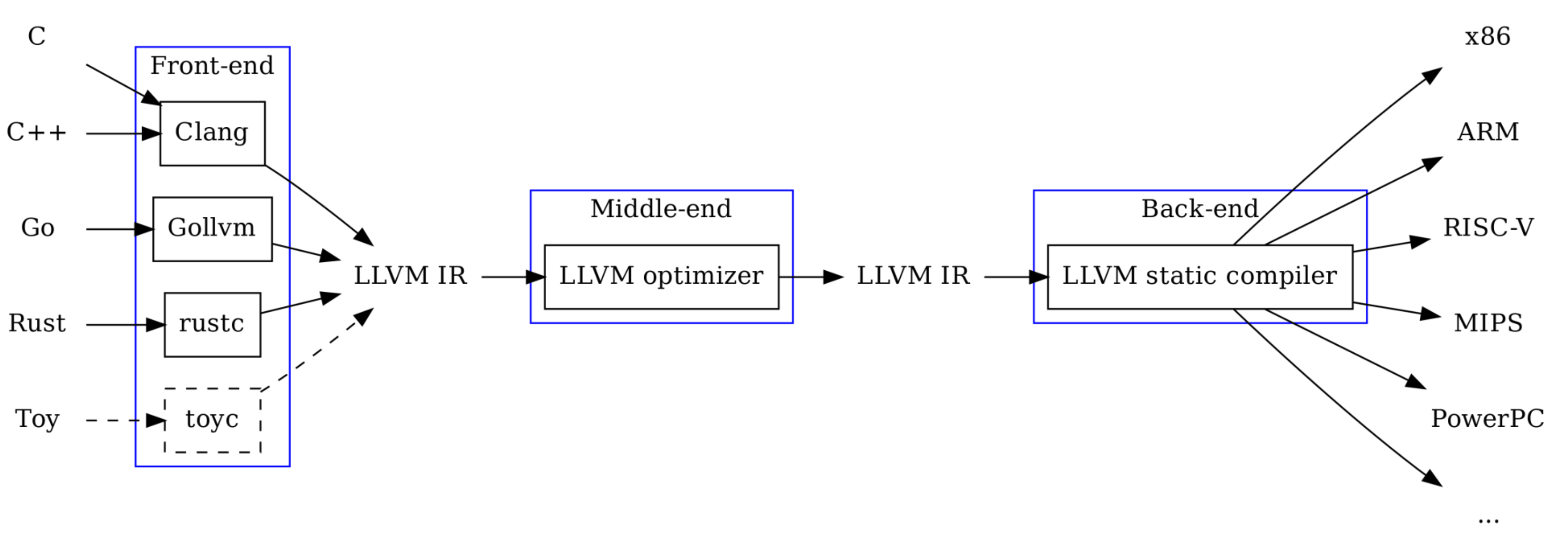
LLVM Compiler Infrastructure

- LLVM began as a research project at the University of Illinois, 2000.
- Goal: Provide a modern, SSA-based compilation strategy capable of supporting both static and dynamic compilation of arbitrary programming languages.
- Lead authors:
    - Chris Lattner
    - Vikram Adve (advisor)





LLVM



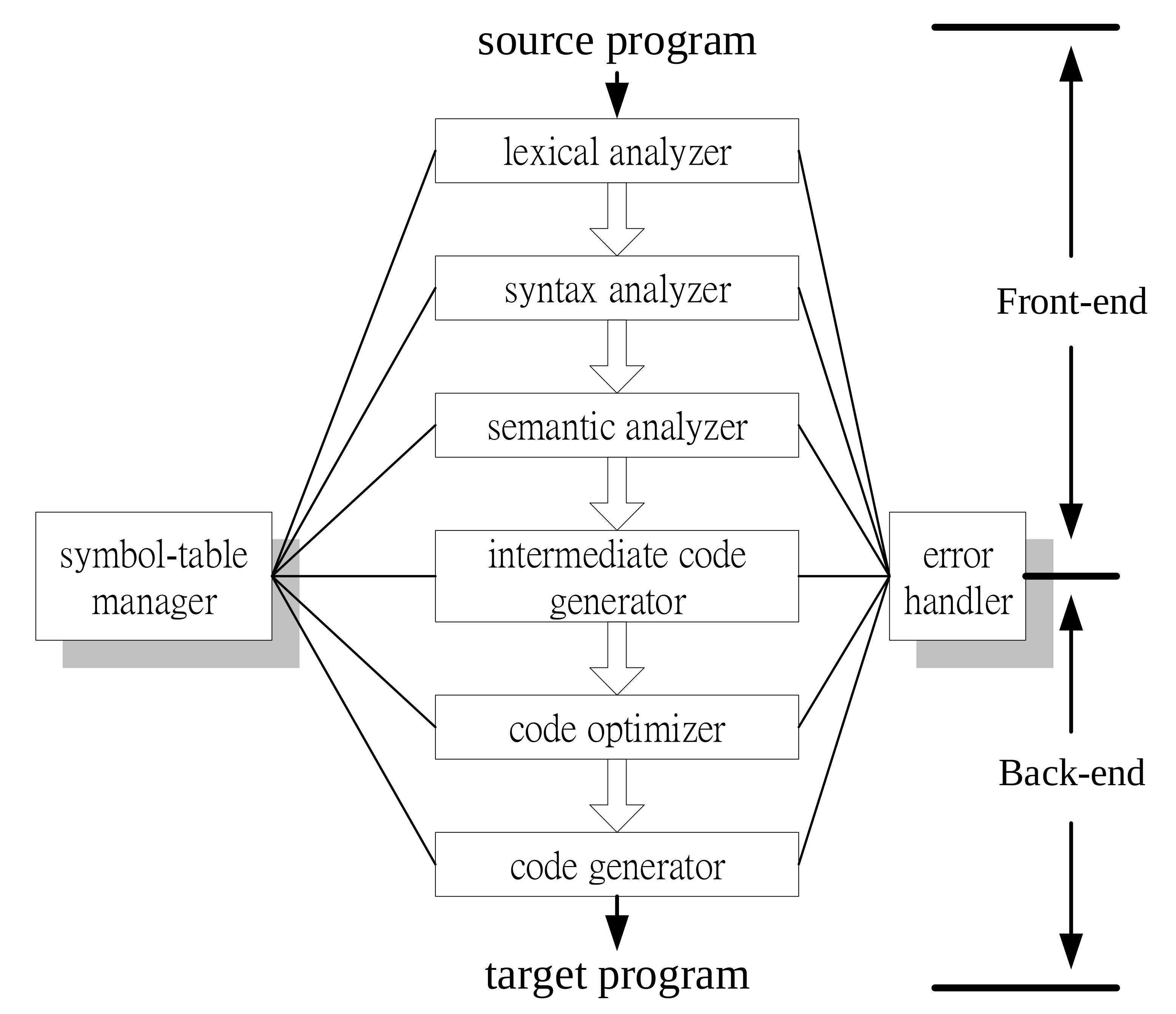
Reference from https://blog.gopheracademy.com/advent-2018/llvm-ir-and-go/

License

- **License: Apache 2.0 License with LLVM exceptions**
    - Allow commercial products to be derived from LLVM with few restrictions and without a requirement for making any derived works also open source.
- **Allow users to:**
    - freely download and use LLVM (in whole or in part) for personal, internal, or commercial purposes.
    - include LLVM in packages or distributions you create.
    - combine LLVM with code licensed under every other major open source license (including BSD, MIT, GPLv2, GPLv3…).
    - make changes to LLVM code without being required to contribute it back to the project - contributions are appreciated though!

Some Industry Users

Company	Description
Apple	All of Apple’s operating systems, iOS, macOS, tvOS and watchOS, are built with LLVM technologies.
Intel	OpenCL / Intel C/C++ compiler
Nvidia	OpenCL runtime compiler (Clang + LLVM)
Sony	CPU compiler for the PlayStation®4 system.
AMD	AOCC (AMD optimizing C/C++ compiler)



source program

lexical analyzer

syntax analyzer

semantic analyzer

intermediate code generator

code optimizer

code generator

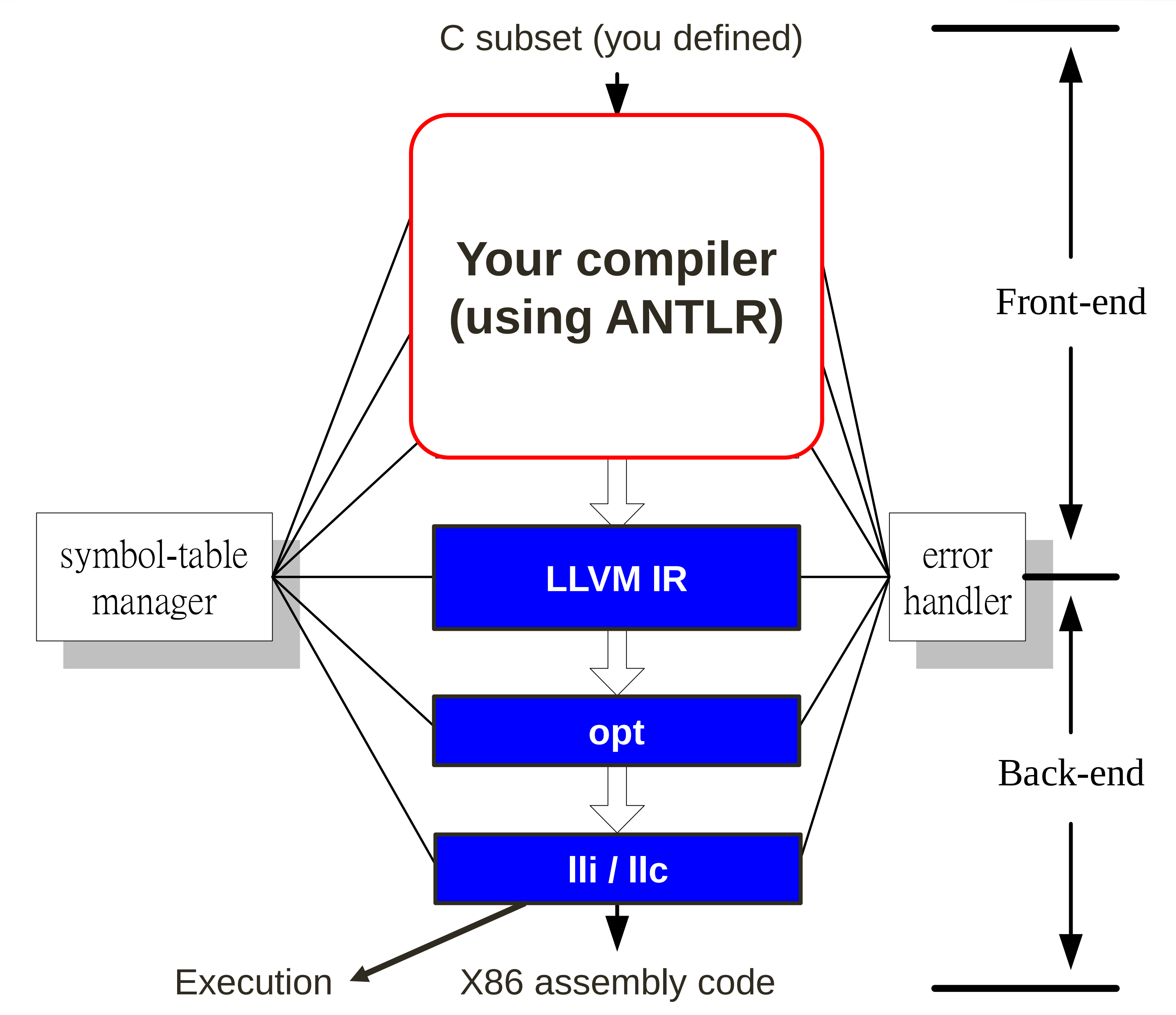
target program

symbol-table manager

error handler

Front-end

Back-end


C subset (you defined)
Your compiler (using ANTLR)
symbol-table manager
LLVM IR
opt
lli / llc
error handler
Front-end
Back-end
Execution
X86 assembly code

Example

C program

```c
int f(int a, int b) {
    return a + 2*b;
}

int main() {
    return f(10, 20);
}
```

LLVM IR

```llvm
define i32 @f(i32 %a, i32 %b) {
; <label>:0
    %1 = mul i32 2, %b
    %2 = add i32 %a, %1
    ret i32 %2
}

define i32 @main() {
; <label>:0
    %1 = call i32 @f(i32 10, i32 20)
    ret i32 %1
}
```

LLVM IR: Identifier

- global
  - Global identifiers (functions, global variables) begin with the '**@**' character.
- Local
  - Local identifiers (register names, types) begin with the ‘**%**’ character.

LLVM IR: Identifier Format

- Named value (建議採用)
  - @main, %str, %foo, …
- Unnamed value
  - %1, %2, … (需要依照順序)
- Constant
  - Boolean: true, false
  - Integer: 123, …
  - Floating point: 123.421, 1.23e+2, …

LLVM IR: Others

- Comments are delimited with a '**;**' and go until the end of line.
- Unnamed temporaries are numbered sequentially (using a per-function incrementing counter, **starting with 0**).
  - Note that basic blocks and unnamed function parameters are included in this numbering.

Example (1)

- Global variables, functions and aliases may have an optional runtime preemption specifier.

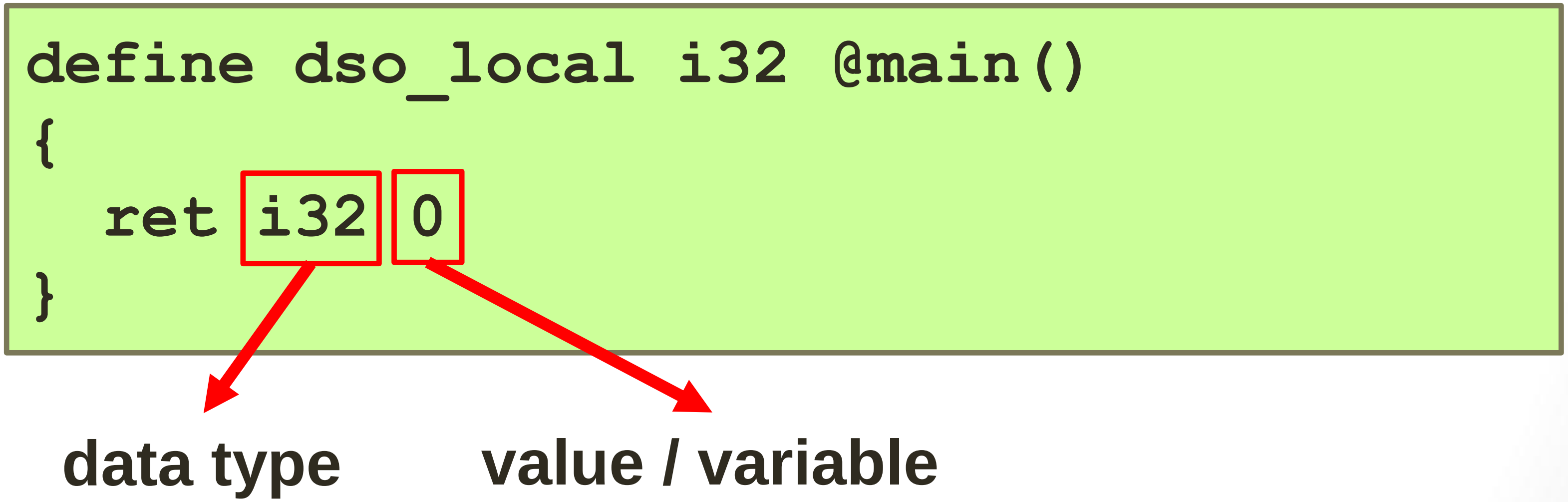
```
int main(void)
{
    return 0;
}
```

```
define dso_local i32 @main()
{
   ret i32 0
}
```

- **dso_preemptable**: Indicates that the function or variable may be replaced by a symbol from outside the linkage unit at runtime.
- **dso_local**: A function or variable marked as dso_local will resolve to a symbol within the same linkage unit.

Example (1)

```
int main(void)
{
    return 0;
}
```



```
define dso_local i32 @main()
{
   ret i32 0
}
```

Example (2)

```
int a;
int main(void)
{
    a = 1;
    return 0;
}
```

Initial value

```
@a = dso_local global i32 0, align 4

define dso_local i32 @main()
{
  store i32 1, i32* @a
  ret i32 0
}
```

Example (3)   1/6

```
int a, b;
int main(void)
{
    a = 1;
    b = a + 4;
    return 0;
}
```

```
@a = dso_local global i32 0, align 4
@b = dso_local global i32 0, align 4

define dso_local i32 @main()
{
  store i32 1, i32* @a
  %1 = load i32, i32* @a
  %2 = add nsw i32 %1, 4
  store i32 %2, i32* @b
  ret i32 0
}
```

Example (3) 2/6

```
int a, b;
int main(void)
{
   a = 1;
   b = a + 4;
   return 0;
}
```

```
@a = dso_local global i32 0, align 4
@b = dso_local global i32 0, align 4

define dso_local i32 @main()
{
  store i32 1, i32* @a
  %1 = load i32, i32* @a
  %2 = add nsw i32 %1, 4
  store i32 %2, i32* @b
  ret i32 0
}
```

Example (3) 3/6

```
int a, b;
int main(void)
{
   a = 1;
   b = a + 4;
   return 0;
}
```

```
@a = dso_local global i32 0, align 4
@b = dso_local global i32 0, align 4

define dso_local i32 @main()
{
  store i32 1, i32* @a
  %1 = load i32, i32* @a
  %2 = add nsw i32 %1, 4
  store i32 %2, i32* @b
  ret i32 0
}
```

Load a to register %1

Example (3) 4/6

```
int a, b;
int main(void)
{
    a = 1;
    b = a + 4;
    return 0;
}
```

nuw and nsw stand for “No Unsigned Wrap” and “No Signed Wrap”, respectively. If the nuw and/or nsw keywords are present, the result value of the add is a poison value if unsigned and/or signed overflow, respectively, occurs.

```
@a = dso_local global i32 0, align 4
@b = dso_local global i32 0, align 4

define dso_local i32 @main()
{
  store i32 1, i32* @a
  %1 = load i32, i32* @a
  %2 = add nsw i32 %1, 4
  store i32 %2, i32* @b
  ret i32 0
}
```

Add register %1 and 4, and then store the result to register %2

Example (3) 5/6

```
int a, b;
int main(void)
{
    a = 1;
    b = a + 4;
    return 0;
}
```

```
@a = dso_local global i32 0, align 4
@b = dso_local global i32 0, align 4

define dso_local i32 @main()
{
  store i32 1, i32* @a
  %1 = load i32, i32* @a
  %2 = add nsw i32 %1, 4
  store i32 %2, i32* @b
  ret i32 0
}
```

store register %2 to b

Example (3) 6/6

```
int a, b;
int main(void)
{
    a = 1;
    b = a + 4;
    return 0;
}
```

使用named variable，避免因為unnamed variable之數字不連續，造成錯誤。

```
@a = dso_local global i32 0, align 4
@b = dso_local global i32 0, align 4

define dso_local i32 @main()
{
  store i32 1, i32* @a
  %t1 = load i32, i32* @a
  %t2 = add nsw i32 %t1, 4
  store i32 %t2, i32* @b
  ret i32 0
}
```

Example (4) 1/7

```
int main(void)
{
   int a, b;
   a = 1;
   b = a + 4;
   return 0;
}
```

The ‘**alloca**’ instruction allocates memory on the stack frame of the currently executing function, to be automatically released when this function returns to its caller.

```
define dso_local i32 @main() {
  %1 = alloca i32, align 4
  %2 = alloca i32, align 4
  store i32 1, i32* %1
  %3 = load i32, i32* %1
  %4 = add nsw i32 %3, 4
  store i32 %4, i32* %2
  ret i32 0
}
```

Example (4) 2/7

```
int main(void)
{
   int a, b;
   a = 1;
   b = a + 4;
   return 0;
}
```

```
define dso_local i32 @main() {
  %1 = alloca i32, align 4
  %2 = alloca i32, align 4
  store i32 1, i32* %1
  %3 = load i32, i32* %1
  %4 = add nsw i32 %3, 4
  store i32 %4, i32* %2
  ret i32 0
}
```

Example (4) 3/7

```
int main(void)
{
   int a, b;
   a = 1;
   b = a + 4;
   return 0;
}
```

```
define dso_local i32 @main() {
  %1 = alloca i32, align 4
  %2 = alloca i32, align 4
  store i32 1, i32* %1
  %3 = load i32, i32* %1
  %4 = add nsw i32 %3, 4
  store i32 %4, i32* %2
  ret i32 0
}
```

Example (4) 4/7

```
int main(void)
{
   int a, b;
   a = 1;
   b = a + 4;
   return 0;
}
```

```
define dso_local i32 @main() {
  %1 = alloca i32, align 4
  %2 = alloca i32, align 4
  store i32 1, i32* %1
  %3 = load i32, i32* %1
  %4 = add nsw i32 %3, 4
  store i32 %4, i32* %2
  ret i32 0
}
```

Load a to register %3

Example (4) 5/7

```
int main(void)
{
   int a, b;
   a = 1;
   b = a + 4;
   return 0;
}
```

```
define dso_local i32 @main() {
  %1 = alloca i32, align 4
  %2 = alloca i32, align 4
  store i32 1, i32* %1
  %3 = load i32, i32* %1
  %4 = add nsw i32 %3, 4
  store i32 %4, i32* %2
  ret i32 0
}
```

Add register %3 and 4, and then store the result to register %4

Example (4) 6/7

```
int main(void)
{
    int a, b;
    a = 1;
    b = a + 4;
    return 0;
}
```

```
define dso_local i32 @main() {
  %1 = alloca i32, align 4
  %2 = alloca i32, align 4
  store i32 1, i32* %1
  %3 = load i32, i32* %1
  %4 = add nsw i32 %3, 4
  store i32 %4, i32* %2
  ret i32 0
}
```

store register %4 to b

Example (4) 7/7

```
int main(void)
{
   int a, b;
   a = 1;
   b = a + 4;
   return 0;
}
```

使用named variable，避免因為unnamed variable之數字不連續，造成錯誤。

```
define dso_local i32 @main() {
  %t1 = alloca i32, align 4
  %t2 = alloca i32, align 4
  store i32 1, i32* %t1
  %t3 = load i32, i32* %t1
  %t4 = add nsw i32 %t3, 4
  store i32 %t4, i32* %t2
  ret i32 0
}
```

Binary Operations

- add, fadd
- sub, fsub
- mul, fmul
- udiv => unsigned integer, sdiv => singed integer, fdiv
- urem => unsigned integer, srem => signed integer, frem

- Reference: https://llvm.org/docs/LangRef.html#binary-operations

Example (5) 1/7

```
int main(void)
{
   printf(“Hello World\n”);
   return 0;
}
```

```
declare dso_local i32 @printf(i8*, ...)
@str = private unnamed_addr constant [13 x i8] c"Hello World\0A\00"
define dso_local i32 @main()
{
  %1 = call i32 (i8*, ...) @printf(i8* getelementptr inbounds ([13 x
i8], [13 x i8]* @str, i64 0, i64 0))
  ret i32 0
}
```

Example (5) 2/7

0xA => \n
0x0 => 0

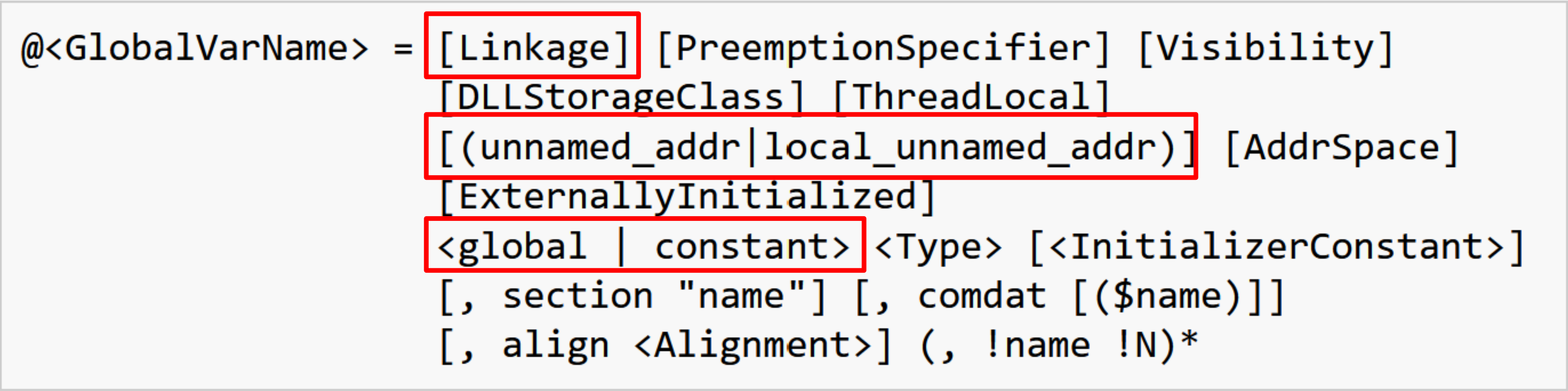
```
private unnamed_addr constant [13 x i8] c"Hello World\0A\00"
```

- In LLVM IR syntax, a backslash followed by two hex characters means a character whose ASCII value is the same as that defined by the two characters.

- [13 x i8] => Array of 13 8-bit integer values.
- private unnamed_addr constant
    - Global variables can be marked with unnamed_addr which indicates that the address is not significant, only the content.
    - Global values with "private" linkage are only directly accessible by objects in the current module.
    - "constant" indicates that the contents of the variable will never be **modified**.

Example (5) 3/7

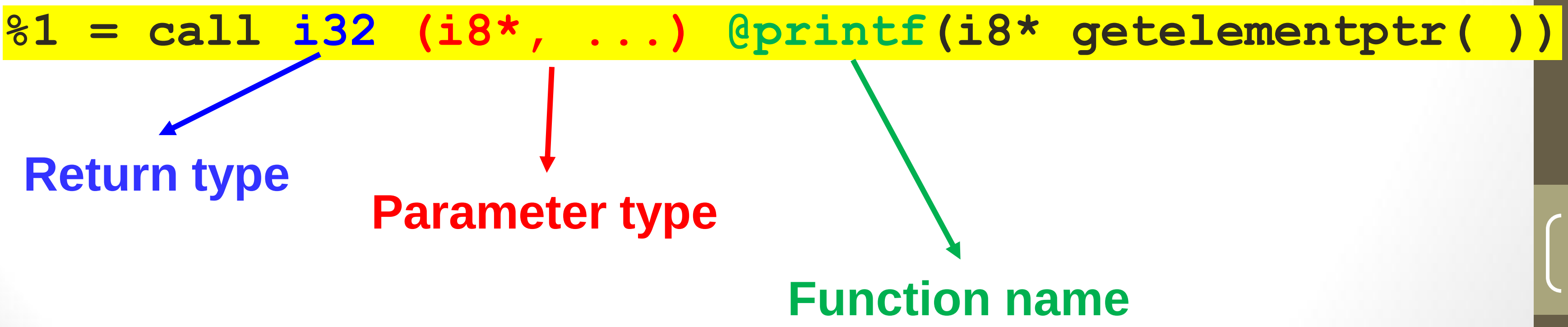
- Syntax of global variable



```
@<GlobalVarName> = [Linkage] [PreemptionSpecifier] [Visibility]
                   [DLLStorageClass] [ThreadLocal]
                   [(unnamed_addr|local_unnamed_addr)] [AddrSpace]
                   [ExternallyInitialized]
                   <global | constant> <Type> [<InitializerConstant>]
                   [, section "name"] [, comdat [($name)]]
                   [, align <Alignment>] (, !name !N)*
```

Example (5) 4/7

```
declare dso_local i32 @printf(i8*, ...)
@str = private unnamed_addr constant [13 x i8] c"Hello World\0A\00"
define dso_local i32 @main()
{
  %1 = call i32 (i8*, ...) @printf(i8* getelementptr inbounds ([13 x
i8], [13 x i8]* @str, i64 0, i64 0))
  ret i32 0
}
```



```
%1 = call i32 (i8*, ...) @printf(i8* getelementptr( ))
```

Return type

Parameter type

Function name

Example (5) 5/7

```
declare dso_local i32 @printf(i8*, ...)
@str = private unnamed_addr constant [13 x i8] c"Hello World\0A\00"
define dso_local i32 @main()
{
  %1 = call i32 (i8*, ...) @printf(i8* getelementptr inbounds ([13 x
i8], [13 x i8]* @str, i64 0, i64 0))
  ret i32 0
}
```

```
getelementptr inbounds ([13 x i8], [13 x i8]* @str, i64 0, i64 0))
```

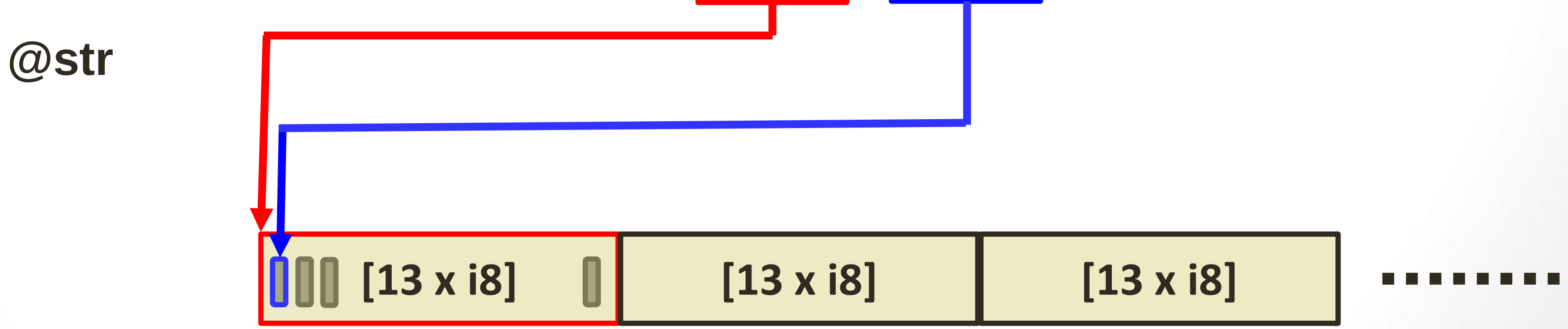
The ‘**getelementptr**’ instruction is used to get the address of a **subelement** of an aggregate data structure. It performs address calculation only and does not access memory.

Example (5) 6/7

```
getelementptr inbounds ([13 x i8], [13 x i8]* @str, i64 0, i64 0))
```

- **inbounds**: the result value of the **getelementptr** is a poison value if one of the pre-defined rules is violated.
- **[13 x i8]**: 欲存取的aggregate data type
- **[13 x i8]* @str**: 指向欲存取的aggregate data type之指標

```
([13 x i8], [13 x i8]* @str, i64 0, i64 0))
```



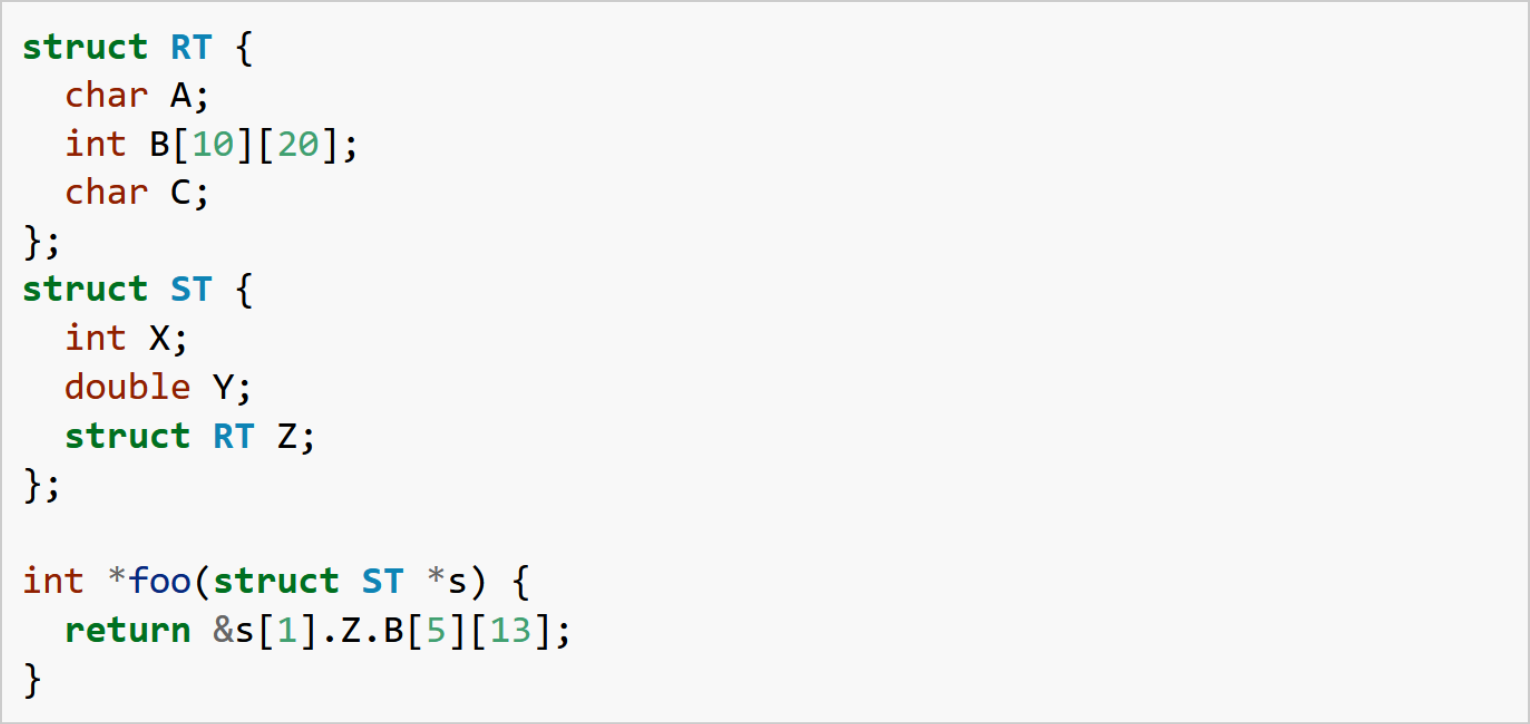
Example (5) 7/7

```
int main(void)
{
   printf(“Hello World\n”);
   return 0;
}
```

```
declare dso_local i32 @printf(i8*, ...)
@str = private unnamed_addr constant [13 x i8] c"Hello World\0A\00"
define dso_local i32 @main()
{
  %t1 = call i32 (i8*, ...) @printf(i8* getelementptr inbounds ([13 x
i8], [13 x i8]* @str, i64 0, i64 0))
  ret i32 0
}
```

使用named variable，避免因為unnamed variable之數字不連續，造成錯誤。

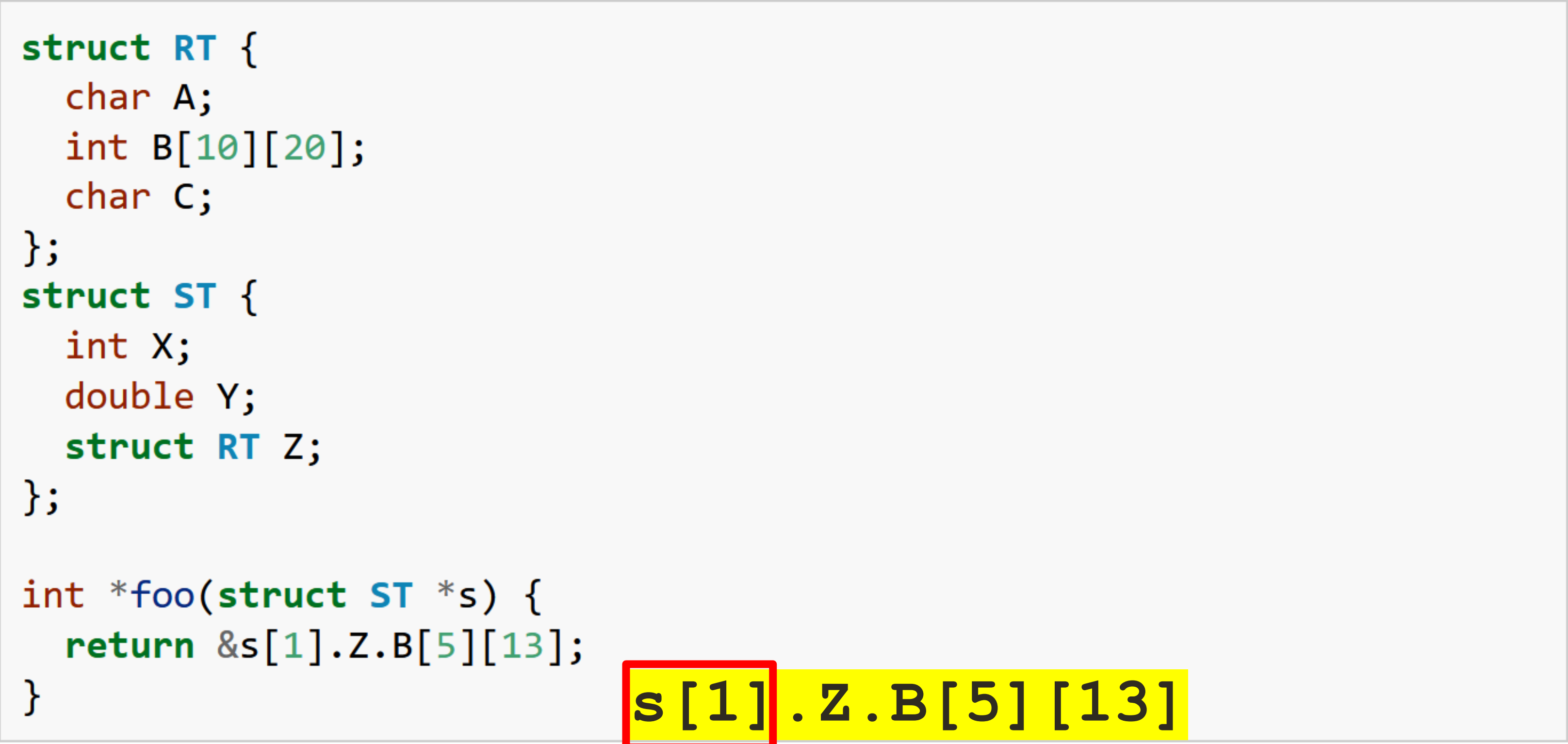
Example (6) 1/5



```c
struct RT {
  char A;
  int B[10][20];
  char C;
};
struct ST {
  int X;
  double Y;
  struct RT Z;
};

int *foo(struct ST *s) {
  return &s[1].Z.B[5][13];
}
```

```llvm
%struct.RT = type { i8, [10 x [20 x i32]], i8 }
%struct.ST = type { i32, double, %struct.RT }
define i32* @foo(%struct.ST* %s) {
  %a = getelementptr inbounds %struct.ST, %struct.ST* %s, i64 1, i32
2, i32 1, i64 5, i64 13
  ret i32* %a
}
```

Example (6) 2/5

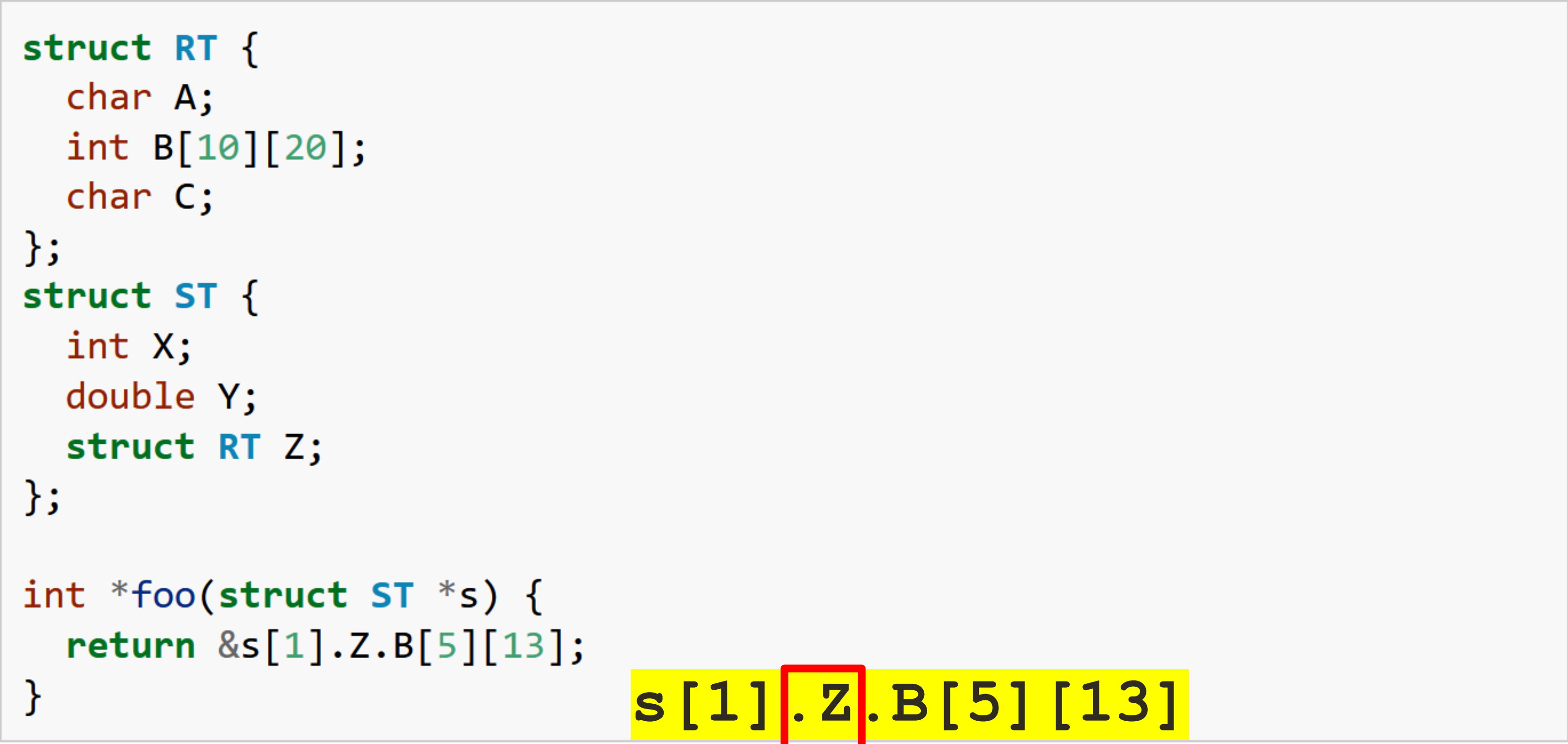


```
struct RT {
  char A;
  int B[10][20];
  char C;
};
struct ST {
  int X;
  double Y;
  struct RT Z;
};

int *foo(struct ST *s) {
  return &s[1].Z.B[5][13];
}
```

```
s[1].Z.B[5][13]
```

```
%struct.ST, %struct.ST* %s, i64 1, i32 2, i32 1, i64 5, i64 13
```

Example (6) 3/5



```c
struct RT {
  char A;
  int B[10][20];
  char C;
};
struct ST {
  int X;
  double Y;
  struct RT Z;
};

int *foo(struct ST *s) {
  return &s[1].Z.B[5][13];
}
```

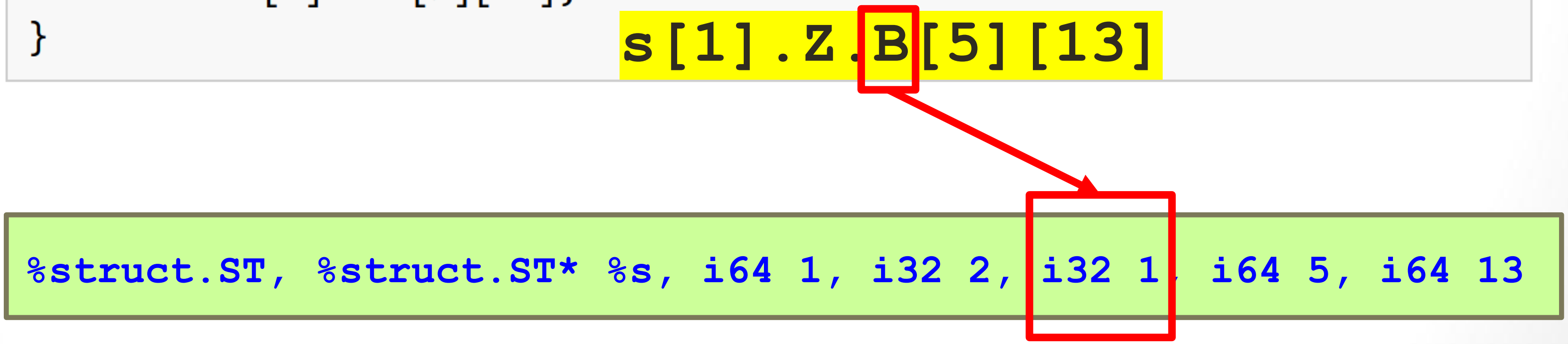
`s[1].Z.B[5][13]`

```
%struct.ST, %struct.ST* %s, i64 1, i32 2, i32 1, i64 5, i64 13
```

Example (6)

4/5

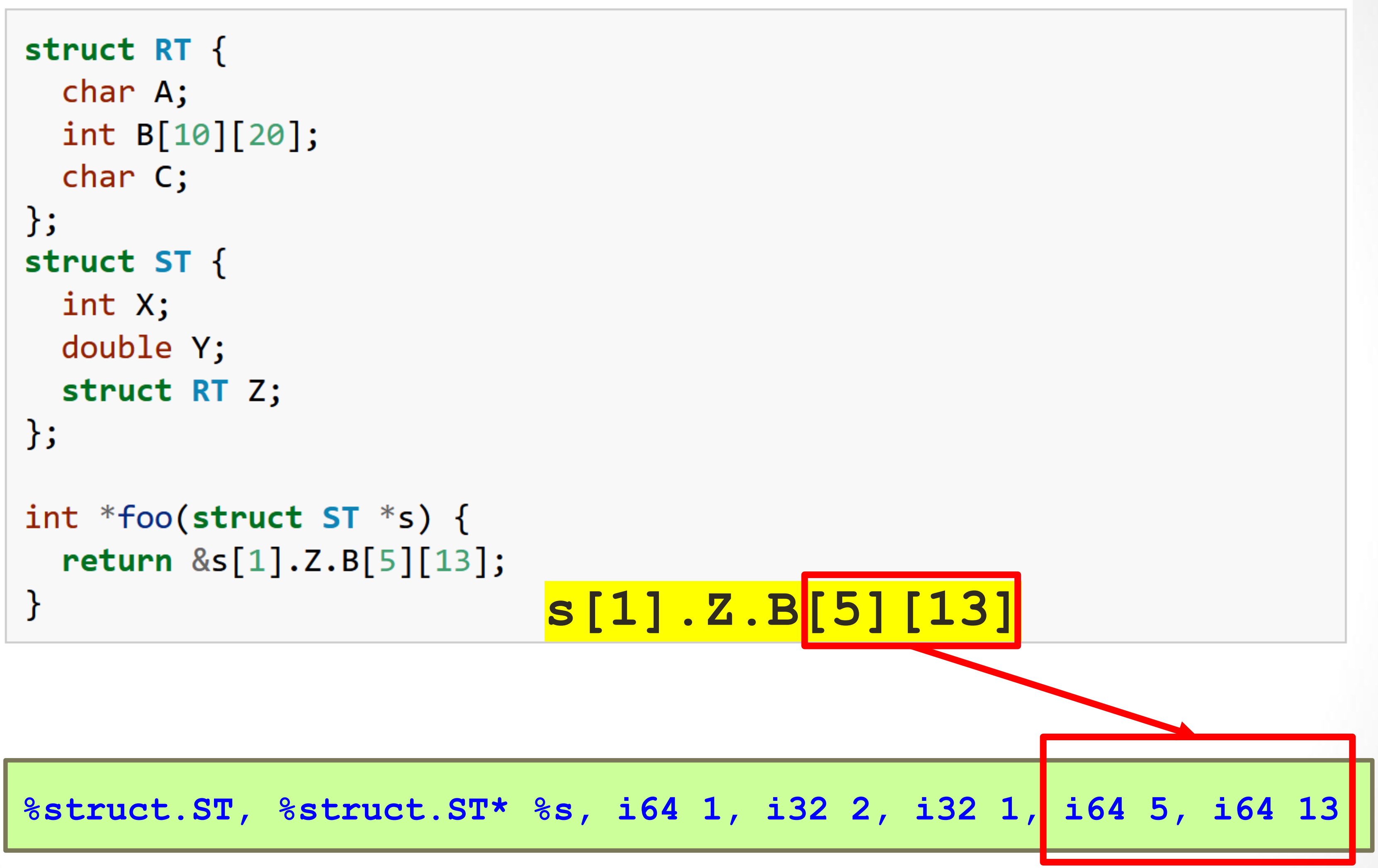
```
struct RT {
  char A;
  int B[10][20];
  char C;
};
struct ST {
  int X;
  double Y;
  struct RT Z;
};

int *foo(struct ST *s) {
  return &s[1].Z.B[5][13];
}
```



```
s[1].Z.B[5][13]
```

```
%struct.ST, %struct.ST* %s, i64 1, i32 2, i32 1, i64 5, i64 13
```

Example (6) 5/5



```
struct RT {
  char A;
  int B[10][20];
  char C;
};
struct ST {
  int X;
  double Y;
  struct RT Z;
};

int *foo(struct ST *s) {
  return &s[1].Z.B[5][13];
}
```

`s[1].Z.B[5][13]`

```
%struct.ST, %struct.ST* %s, i64 1, i32 2, i32 1, i64 5, i64 13
```

Example (7) 1/5

```
int func(int a)
{
   return a+1;
}




int main(void)
{
   int b;
   b = func(2);
   return 0;
}
```

```
define dso_local i32 @func(i32 %0) {
  %2 = alloca i32, align 4
  store i32 %0, i32* %2
  %3 = load i32, i32* %2
  %4 = add nsw i32 %3, 1
  ret i32 %4
}


define dso_local i32 @main() {
  %1 = alloca i32, align 4
  %2 = call i32 @func(i32 2)
  store i32 %2, i32* %1
  ret i32 0
}
```

Example (7) 2/5

```c
int func(int a)
{
   return a+1;
}



int main(void)
{
   int b;
   b = func(2);
   return 0;
}
```

```llvm
define dso_local i32 @func(i32 %0) {
  %2 = alloca i32, align 4
  store i32 %0, i32* %2
  %3 = load i32, i32* %2
  %4 = add nsw i32 %3, 1
  ret i32 %4
}


define dso_local i32 @main() {
  %1 = alloca i32, align 4
  %2 = call i32 @func(i32 2)
  store i32 %2, i32* %1
  ret i32 0
}
```

Example (7) 3/5

```c
int func(int a)
{
    return a+1;
}




int main(void)
{
    int b;
    b = func(2);
    return 0;
}
```

```llvm
define dso_local i32 @func(i32 %0) {
  %2 = alloca i32, align 4
  store i32 %0, i32* %2
  %3 = load i32, i32* %2
  %4 = add nsw i32 %3, 1
  ret i32 %4
}


define dso_local i32 @main() {
  %1 = alloca i32, align 4
  %2 = call i32 @func(i32 2)
  store i32 %2, i32* %1
  ret i32 0
}
```

a = 2

Example (7) 4/5

```
int func(int a)
{
    return a+1;
}
```

```
define dso_local i32 @func(i32 %0) {
  %2 = alloca i32, align 4
  store i32 %0, i32* %2
  %3 = load i32, i32* %2
  %4 = add nsw i32 %3, 1
  ret i32 %4
}
```

a + 1

```
int main(void)
{
    int b;
   b = func(2);
    return 0;
}
```

```
define dso_local i32 @main() {
  %1 = alloca i32, align 4
  %2 = call i32 @func(i32 2)
  store i32 %2, i32* %1
  ret i32 0
}
```

Example (7) 5/5

```c
int func(int a)
{
   return a+1;
}



int main(void)
{
   int b;
   b = func(2);
   return 0;
}
```

```llvm
define dso_local i32 @func(i32 %t0) {
  %a = alloca i32, align 4
  store i32 %t0, i32* %a
  %t1 = load i32, i32* %a
  %t2 = add nsw i32 %t1, 1
  ret i32 %t2
}

define dso_local i32 @main() {
  %b = alloca i32, align 4
  %t3 = call i32 @func(i32 2)
  store i32 %t3, i32* %b
  ret i32 0
}
```

使用named variable，避免因為unnamed variable之數字不連續，造成錯誤。

Example (8) 1/7

```
int nums[3] = {1, 2, 3};
int main(void)
{
    int a;
    a = nums[2];
    return 0;
}
```

```
@nums = dso_local global [3 x i32] [i32 1, i32 2, i32 3]
define dso_local i32 @main() {
  %2 = alloca i32, align 4
  %3 = load i32, i32* getelementptr inbounds ([3 x i32], [3 x i32]*
@nums, i64 0, i64 2)
  store i32 %3, i32* %2
  ret i32 0
}
```

Example (8)

1/7

```
int nums[3] = {1, 2, 3};
int main(void)
{
     int a;
     a = nums[2];
     return 0;
}
```

```
@nums = dso_local global [3 x i32] [i32 1, i32 2, i32 3]
define dso_local i32 @main() {
  %2 = alloca i32, align 4
  %3 = load i32, i32* getelementptr inbounds ([3 x i32], [3 x i32]*
@nums, i64 0, i64 2)
  store i32 %3, i32* %2
  ret i32 0
}
```

Example (8)

2/7

```
int nums[3] = {1, 2, 3};
int main(void)
{
    int a;
    a = nums[2];
    return 0;
}
```

```
@nums = dso_local global [3 x i32] [i32 1, i32 2, i32 3]
define dso_local i32 @main() {
  %2 = alloca i32, align 4
  %3 = load i32, i32* getelementptr inbounds ([3 x i32], [3 x i32]*
@nums, i64 0, i64 2)
  store i32 %3, i32* %2
  ret i32 0
}
```

Example (8) 3/7

```
int nums[3] = {1, 2, 3};
int main(void)
{
    int a;
    a = nums[2];
    return 0;
}
```

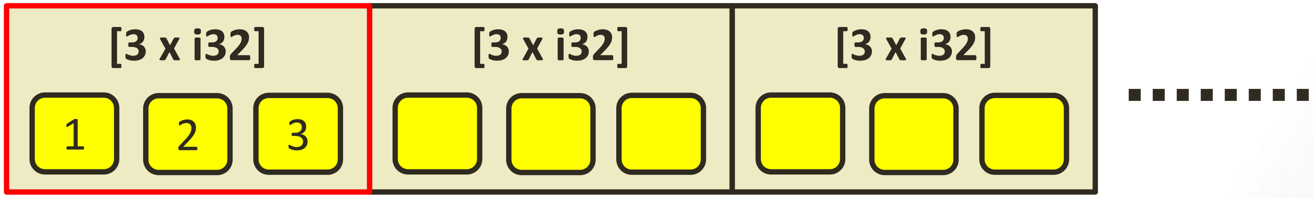
```
@nums = dso_local global [3 x i32] [i32 1, i32 2, i32 3]
define dso_local i32 @main() {
  %2 = alloca i32, align 4
  %3 = load i32, i32* getelementptr inbounds ([3 x i32], [3 x i32]*
@nums, i64 0, i64 2)
  store i32 %3, i32* %2
  ret i32 0
}
```

Example (8) 4/7

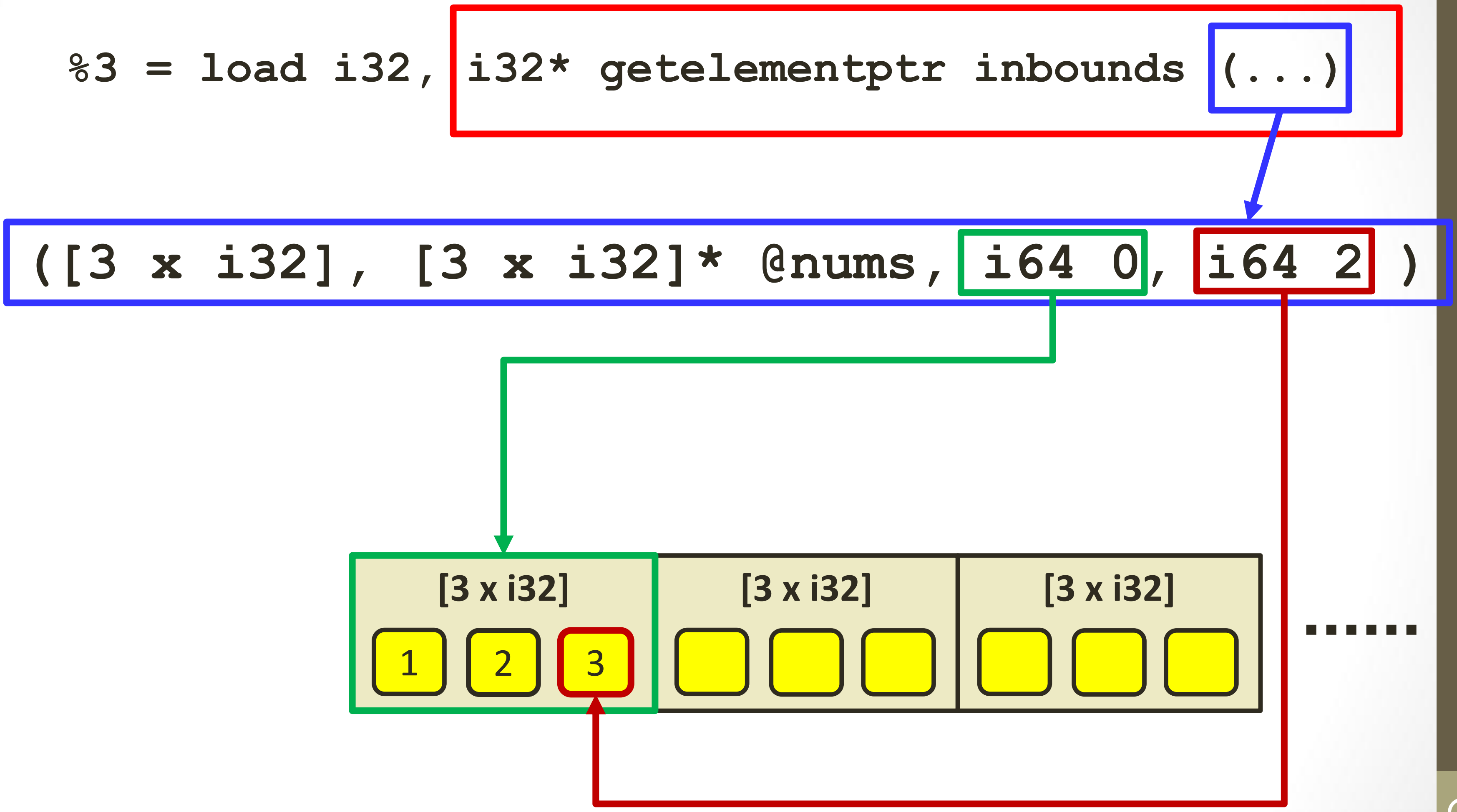
```
%3 = load i32, i32* getelementptr inbounds (...)
```

```
([3 x i32], [3 x i32]* @nums, i64 0, i64 2)
```

- **inbounds**: the result value of the **getelementptr** is a poison value if one of the pre-defined rules is violated.
- **[3 x i32]**: 欲存取的aggregate data type
- **[3 x i32]* @nums**: 指向欲存取的aggregate data type之指標



Example (8) 5/7



```
%3 = load i32, i32* getelementptr inbounds (...)
```

```
([3 x i32], [3 x i32]* @nums, i64 0, i64 2 )
```

Example (8) 6/7

```
int nums[3] = {1, 2, 3};
int main(void)
{
    int a;
    a = nums[2];
    return 0;
}
```

```
@nums = dso_local global [3 x i32] [i32 1, i32 2, i32 3]
define dso_local i32 @main() {
  %2 = alloca i32, align 4
  %3 = load i32, i32* getelementptr inbounds ([3 x i32], [3 x i32]*
@nums, i64 0, i64 2)
  store i32 %3, i32* %2
  ret i32 0
}
```

Example (8)　　7/7

```
int nums[3] = {1, 2, 3};
int main(void)
{
    int a;
    a = nums[2];
    return 0;
}
```

```
@nums = dso_local global [3 x i32] [i32 1, i32 2, i32 3]
define dso_local i32 @main() {
  %a = alloca i32, align 4
  %t1 = load i32, i32* getelementptr inbounds ([3 x i32], [3 x i32]*
@nums, i64 0, i64 2)
  store i32 %t1, i32* %a
  ret i32 0
}
```

使用named variable，避免因為unnamed variable之數字不連續，造成錯誤。

Example (9): if-then-else construct (1/5)

```
int main()
{
   int a, b;

   a = 1;
   if (a > 0)
      b = 0;
   else
      b = a + 2;
}
```

詳細指令說明請參閱 https://llvm.org/docs/LangRef.html#icmp-instruction

Example (9) 2/5

```
define dso_local i32 @main() {
  %t1 = alloca i32, align 4     ; allocate a in stack frame
  %t2 = alloca i32, align 4     ; allocate b in stack frame
  store i32 1, i32* %t1, align 4     ; a = 1
  %t3 = load i32, i32* %t1, align 4 ; load a to %t3
  %cond = icmp sgt i32 %t3, 0        ; a > 0?
  br i1 %cond, label %Ltrue, label %Lfalse

Ltrue:                                 ; If-then part
  store i32 0, i32* %t2, align 4     ; b = 0
  br label %Lend

Lfalse:                                ; If-else part
  %t5 = load i32, i32* %t1, align 4  ; load a to %t5
  %t6 = add nsw i32 %t5, 2            ; %t6 = a + 2
  store i32 %t6, i32* %t2, align 4    ; store %t6 to b
  br label %Lend

Lend:
  ret i32 0
}
```

Comparison: icmp 3/5

```
; yields i1 or <N x i1>:result
<result> = icmp <cond> <ty> <op1>, <op2>
```

The ‘icmp’ instruction returns a boolean value or a vector of boolean values based on comparison of its two integer, integer vector, pointer, or pointer vector operands.

- eq: equal
- ne: not equal
- ugt: unsigned greater than
- uge: unsigned greater or equal
- ult: unsigned less than
- ule: unsigned less or equal
- sgt: signed greater than
- sge: signed greater or equal
- slt: signed less than
- sle: signed less or equal

br instruction 4/5

```
; Conditional branch
br i1 <cond>, label <iftrue>, label <iffalse>


; Unconditional branch
br label <dest>
```

- The conditional branch form of the ‘br’ instruction takes a single ‘**i1**’ value and two ‘**label**’ values.
- The unconditional form of the ‘br’ instruction takes a single ‘label’ value as a target.

Example (9) 5/5

```
define dso_local i32 @main() {
  %t1 = alloca i32, align 4    ; allocate a in stack frame
  %t2 = alloca i32, align 4    ; allocate b in stack frame
  store i32 1, i32* %t1, align 4    ; a = 1
  %t3 = load i32, i32* %t1, align 4 ; load a to %t3
  %cond = icmp sgt i32 %t3, 0       ; a > 0?
  br i1 %cond, label %Ltrue, label %Lfalse

Ltrue:                             ; If-then part
  store i32 0, i32* %t2, align 4    ; b = 0
  br label %Lend

Lfalse:                            ; If-else part
  %t5 = load i32, i32* %t1, align 4  ; load a to %t5
  %t6 = add nsw i32 %t5, 2           ; %t6 = a + 2
  store i32 %t6, i32* %t2, align 4   ; store %t6 to b
  br label %Lend

Lend:
  ret i32 0
}
```

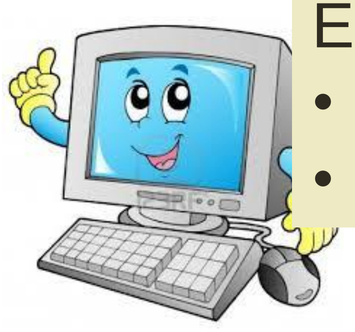
Construct Your Compiler (1/2)

```
int main()
{
   int a;
   int b;

   a = 1;
   b = a + 2;
   printf(“%d”, b);
}
```

Your Compiler

LLVM IR
(pseudo
assembly code

Execution
- lli (interpreter)
- llc (x86 assembly)



Construct Your Compiler (2/2)

```
int main()
{
    int a;
    int b;
    a = 1;
    b = a + 2;
    printf(“%d”, b);
}
```

```
declare dso_local i32 @printf(i8*, ...)
@.str = private unnamed_addr constant [4 x i8] c"%d\0A\00"
define dso_local i32 @main(){
  %a = alloca i32, align 4
  %b = alloca i32, align 4
  store i32 1, i32* %a
  %t1 = load i32, i32* %a
  %t2 = add nsw i32 %t1, 2
  store i32 %t2, i32* %b
  %t3 = load i32, i32* %b
  %t4 = call i32 (i8*, ...) @printf(i8* getelementptr
inbounds ([4 x i8], [4 x i8]* @.str, i64 0, i64 0), i32 %t3)
  ret i32 0
}
```

a=1

b=a+2

Project 4: Sample Program

Grammar (1)

```
statement
    : Identifier '=' arith_expression
    |  IF '(' arith_expression ')`
       if_then_statements
    ;

arith_expression
    : multExpr
      ( ‘+’ multExpr
      | ‘-’ multExpr
      )*
    ;
```

Grammar (2)

```
multExpr
     : signExpr
      ( '*' signExpr
      | '/' signExpr
      )*
     ;

signExpr
     : primaryExpr
     | '-' primaryExpr
     ;
```

Grammar (3)

```
primaryExpr
    : Integer_constant
    | Floating_point_constant
    | Identifier
    | '(' arith_expression ')'
    ;
```

C subset to LLVM IR

- Deliver information
  - Synthesized attributes
  - Inherited attributes

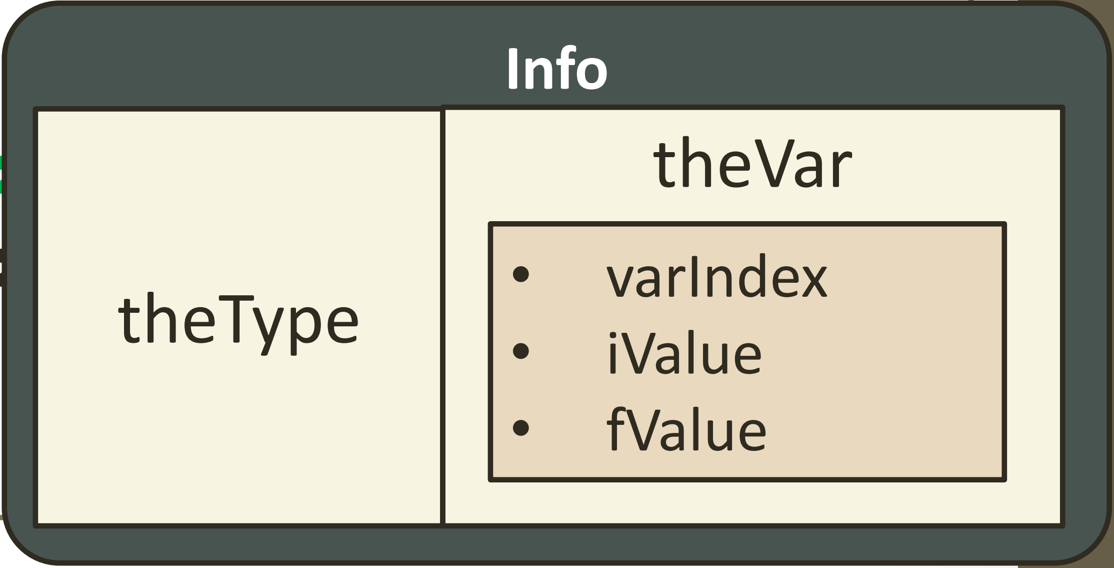
Observation (1)

```
statement
    : Identifier '=' arith_expression
    ;
arith_expression
    : multExpr
      ( ‘+’ multExpr
      | ‘-’ multExpr
      )*
    ;
```

```
%t2 = add nsw i32 %t1, i32 %t3

%t2 = add nsw i32 %t1, 123
```

Observation (2)

```
statement
     : Identifier '=' arith_expression
     ;
arith_expression
     : multExpr
       ( ‘+’ multE
       | ‘-’ multE
       )*
     ;
```



Info

theType

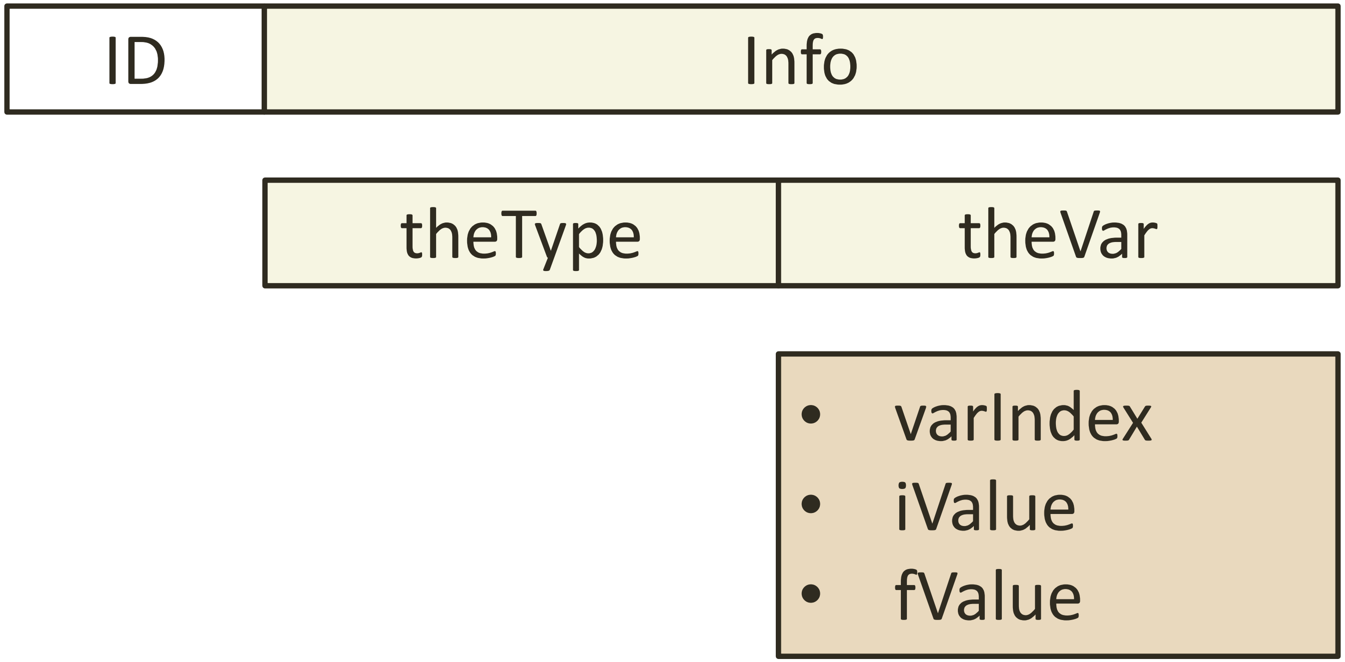
theVar

- varIndex
- iValue
- fValue

```
%t2 = add nsw i32 %t1, i32 %t3

%t2 = add nsw i32 %t1, 123
```

Symbol table

(自己規劃與設計)



ID	Info

theType	theVar

- varIndex
- iValue
- fValue

int a; → %t0 = alloca i32, align 4

a	INT	0 (varIndex)

- 不需要處理register allocation的問題，變數可以有無限多個，register allocation由llvm後端完成。 **(自己規劃與設計)**

```
@members {
    boolean TRACEON = false;
    HashMap<String, Info> symtab = new
HashMap<String,Info>();


    List<String> TextCode = new ArrayList<String>();

    ...
}
```

```
primaryExpr returns [Info theInfo]
@init {theInfo = new Info();}:
     Integer_constant
   {  $theInfo.theType = Type.CONST_INT;
      $theInfo.theVar.iValue =Integer.parseInt($Integer_constant.text); }
   | Floating_point_constant
   | Identifier
   {
     // get type information from symtab.
     Type the_type = symtab.get($Identifier.text).theType;
     $theInfo.theType = the_type;

     // get variable index from symtab.
     int vIndex = symtab.get($Identifier.text).theVar.varIndex;
     switch (the_type) {
     case INT:
          // get a new temporary variable and
          // load the variable into the temporary variable.
          // Ex: \%tx = load i32, i32* \%ty.
          TextCode.add("\%t" + varCount + "=load i32, i32* \%t" + vIndex);

          // Now, Identifier's value is at the temporary variable \%t[varCount].
          // Therefore, update it.
          $theInfo.theVar.varIndex = varCount;
          varCount ++;
          break;
     case FLOAT:
```

```
multExpr returns [Info theInfo]
@init {theInfo = new Info();}
      : a = signExpr {$theInfo=$a.theInfo;}
       ( '*' signExpr
       | '/' signExpr
        )*
      ;


signExpr returns [Info theInfo]
@init {theInfo = new Info();}
      : a=primaryExpr {$theInfo=$a.theInfo;}
      | '-' primaryExpr
      ;
```

```
arith_expression returns [Info theInfo]
@init {theInfo = new Info();}
       : a = multExpr {$theInfo=$a.theInfo;}
      ( '+' b = multExpr
        { // code generation.
          if (($a.theInfo.theType == Type.INT) &&
              ($b.theInfo.theType == Type.INT)) {
             TextCode.add("\%t" + varCount + " = add nsw i32
\%t" + $theInfo.theVar.varIndex + ", \%t" +
$b.theInfo.theVar.varIndex);


             // Update arith_expression's theInfo.
             theInfo.theType = Type.INT;
             $theInfo.theVar.varIndex = varCount;
             varCount ++; }
          }
       | '-' c = multExpr
```

```
assign_stmt: Identifier '=' arith_expression
    {
      Info theRHS = $arith_expression.theInfo;
      Info theLHS = symtab.get($Identifier.text);

      if ((theLHS.theType == Type.INT) &&
          (theRHS.theType == Type.INT)) {

         // issue store insruction.
         // Ex: store i32 \%tx, i32* \%ty
         TextCode.add("store i32 \%t" + theRHS.theVar.varIndex +
", i32* \%t" + theLHS.theVar.varIndex);
      } else if ((theLHS.theType == Type.INT) &&
                 (theRHS.theType == Type.CONST_INT)) {

         // issue store insruction.
         // Ex: store i32 value, i32* \%ty
         TextCode.add("store i32 " + theRHS.theVar.iValue + ",
i32* \%t" + theLHS.theVar.varIndex);
      }
    }
    ;
```

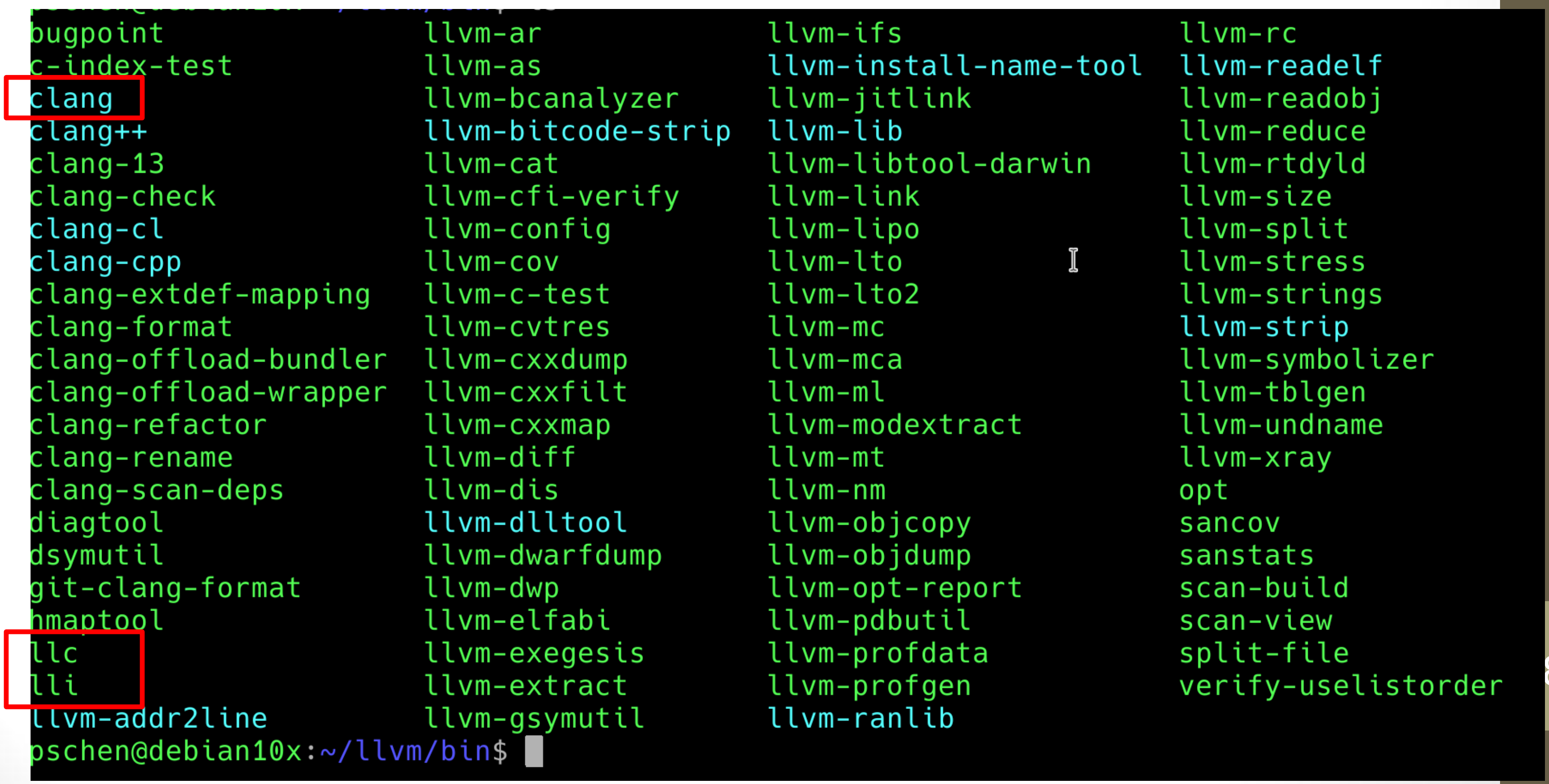
Support Function: printf()

```
int main(void)
{
   printf(“Hello World\n”);
   return 0;
}
```

```
declare dso_local i32 @printf(i8*, ...)
@str = private unnamed_addr constant [13 x i8] c"Hello World\0A\00"
define dso_local i32 @main()
{
  %t1 = call i32 (i8*, ...) @printf(i8* getelementptr inbounds ([13 x
i8], [13 x i8]* @str, i64 0, i64 0))
  ret i32 0
}
```

Installation: LLVM

- $ sudo –s
- # apt-get install clang



```
bugpoint               llvm-ar             llvm-ifs                llvm-rc
c-index-test           llvm-as             llvm-install-name-tool  llvm-readelf
clang                  llvm-bcanalyzer     llvm-jitlink            llvm-readobj
clang++                llvm-bitcode-strip  llvm-lib                llvm-reduce
clang-13               llvm-cat            llvm-libtool-darwin     llvm-rtdyld
clang-check            llvm-cfi-verify     llvm-link               llvm-size
clang-cl               llvm-config         llvm-lipo               llvm-split
clang-cpp              llvm-cov            llvm-lto                llvm-stress
clang-extdef-mapping   llvm-c-test         llvm-lto2               llvm-strings
clang-format           llvm-cvtres         llvm-mc                 llvm-strip
clang-offload-bundler  llvm-cxxdump        llvm-mca                llvm-symbolizer
clang-offload-wrapper  llvm-cxxfilt        llvm-ml                 llvm-tblgen
clang-refactor         llvm-cxxmap         llvm-modextract         llvm-undname
clang-rename           llvm-diff           llvm-mt                 llvm-xray
clang-scan-deps        llvm-dis            llvm-nm                 opt
diagtool               llvm-dlltool        llvm-objcopy            sancov
dsymutil               llvm-dwarfdump      llvm-objdump            sanstats
git-clang-format       llvm-dwp            llvm-opt-report         scan-build
hmaptool               llvm-elfabi         llvm-pdbutil            scan-view
llc                    llvm-exegesis       llvm-profdata           split-file
lli                    llvm-extract        llvm-profgen            verify-uselistorder
llvm-addr2line         llvm-gsymutil       llvm-ranlib
pschen@debian10x:~/llvm/bin$
```

Clang Options: -emit-llvm -S

Use Clang to generate LLVM-IR code

- `$clang -S -emit-llvm test.c`

若不知道C code相對應的LLVM-IR code是什麼，可以使用”clang -S -emit-llvm”產生LLVM IR，以利觀察。

Use ANTLR from the command-line (1)

- `$ java -cp antlr-3.5.2-complete.jar org.antlr.Tool myCompiler.g`

- 產生
  - myCompilerLexer.java
  - myCompilerParser.java
  - myCompilertokens

Use ANTLR from the command-line (2)

- **Compile**

  -
    ```
    $javac  -cp  ./antlr-3.5.2-complete.jar
    myCompilerLexer.java
    myCompilerParser.java
    myCompiler_test.java
    ```
- **Execute your compiler**

  -
    ```
    $java  -cp  ./antlr-3.5.2-complete.jar:.
    myCompiler_test  input.c
    ```

    (產生input.ll)

Use ANTLR from the command-line (3)

- **Execute your assembly code**
  - `$lli input.ll` (interpreter)
  - `$llc input.ll` (generate input.s)
  - `$gcc input.s` (generate a.out)
  - `./a.out`

Backup