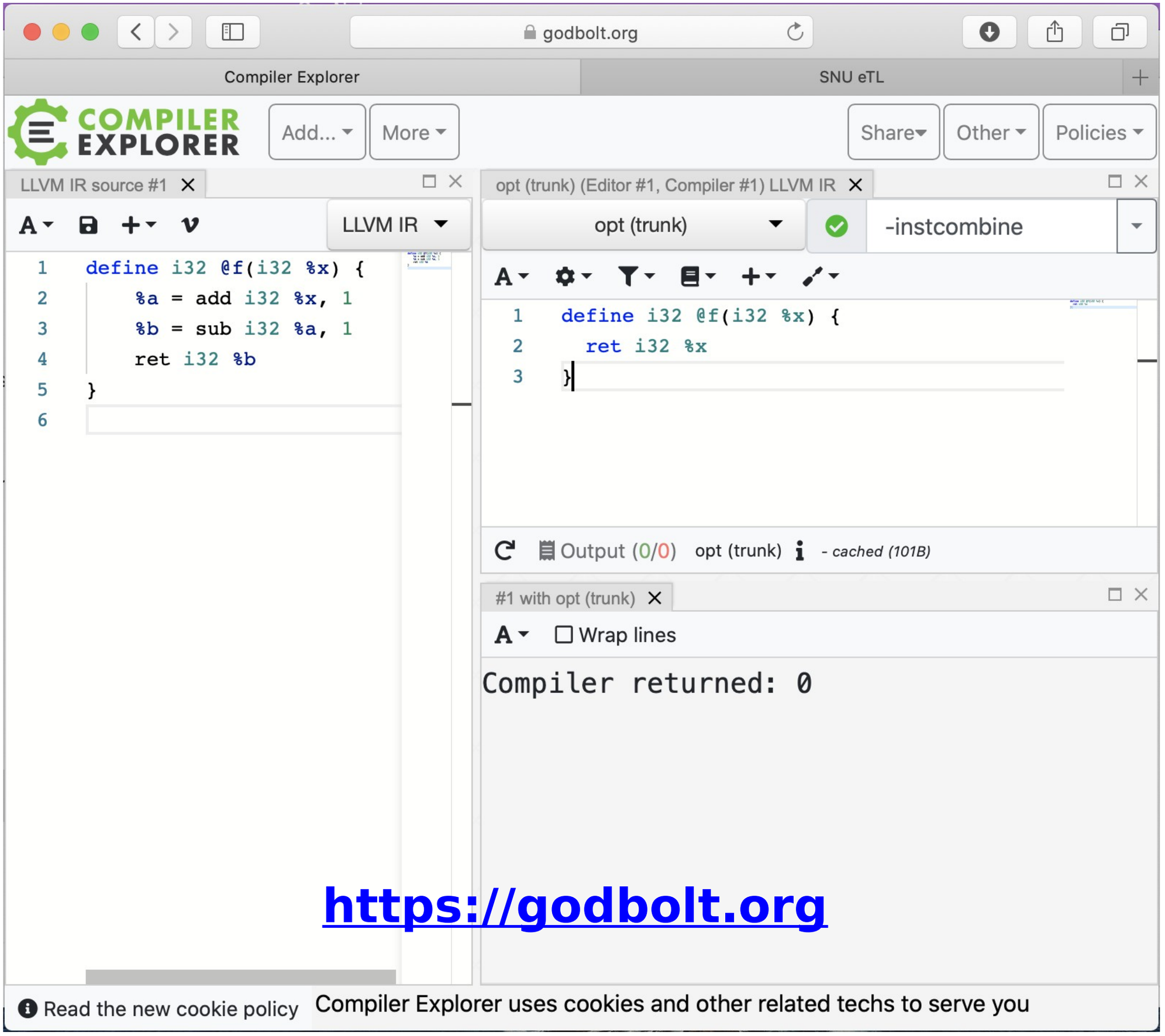
# Using LLVM IR

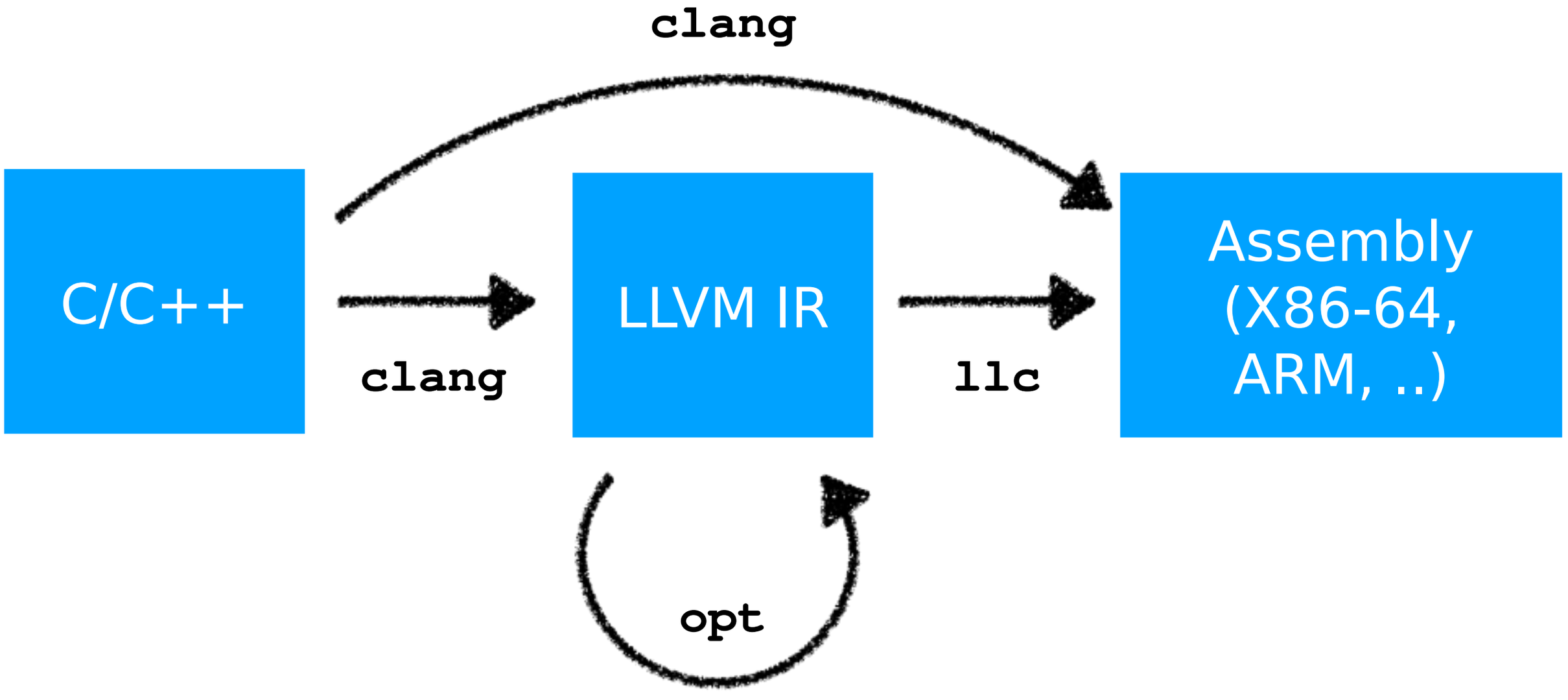
2022. 3. 21
SWPP Practice Session
Seunghyeon Nam
(slides by Juneyoung Lee)

# Building LLVM

- We’re going to announce Assignment 2 this week
- You’ll have to use the built LLVM; the updated build script will be uploaded together
- Let us know if there are any issues


godbolt.org
Compiler Explorer
SNU eTL
COMPILER EXPLORER
Add...
More
Share
Other
Policies
LLVM IR source #1
LLVM IR
1 define i32 @f(i32 %x) {
2 %a = add i32 %x, 1
3 %b = sub i32 %a, 1
4 ret i32 %b
5 }
6
opt (trunk) (Editor #1, Compiler #1) LLVM IR
opt (trunk)
-instcombine
1 define i32 @f(i32 %x) {
2 ret i32 %x
3 }
Output (0/0) opt (trunk) - cached (101B)
#1 with opt (trunk)
Wrap lines
Compiler returned: 0
https://godbolt.org
Read the new cookie policy
Compiler Explorer uses cookies and other related techs to serve you

# Converting LLVM IR from/to *



# Example - fibonacci

```
1   unsigned fib(unsigned n) {
2       unsigned answ;
3       if (n <= 1)
4           answ = n;
5       else
6           answ = fib(n - 1) + fib(n - 2);
7       return answ;
8   }
```

**https://godbolt.org/z/QkuFvZ**

# Example - fibonacci

```c
unsigned fib(unsigned n) {
    unsigned answ;
    if (n <= 1)
        answ = n;
    else
        answ = fib(n - 1) + fib(n - 2);
    return answ;
}
```

```
1   define i32 @fib(i32 %n) {
2   entry:
3     %cmp = icmp ult i32 %n, 2
4     br i1 %cmp, label %if.end, label %if.else
5
6   if.else:
7     %sub = add i32 %n, -1
8     %call = call i32 @fib(i32 %sub)
9     %sub1 = add i32 %n, -2
10    %call2 = call i32 @fib(i32 %sub1)
11    %add = add i32 %call2, %call
12    br label %if.end
13
14  if.end:
15    %answ.0 = phi i32 [ %add, %if.else ], [ %n, %entry ]
16    ret i32 %answ.0
17  }
18
```

**3 Basic blocks: entry, if.else, if.end**

# Example - fibonacci

**i32: integer, 32 bits**

```c
unsigned fib(unsigned n) {
    unsigned answ;
    if (n <= 1)
        answ = n;
    else
        answ = fib(n - 1) + fib(n - 2);
    return answ;
}
```

```
1   define i32 @fib(i32 %n) {
2   entry:
3     %cmp = icmp ult i32 %n, 2
4     br i1 %cmp, label %if.end, label %if.else
5
6   if.else:
7     %sub = add i32 %n, -1
8     %call = call i32 @fib(i32 %sub)
9     %sub1 = add i32 %n, -2
10    %call2 = call i32 @fib(i32 %sub1)
11    %add = add i32 %call2, %call
12    br label %if.end
13
14  if.end:
15    %answ.0 = phi i32 [ %add, %if.else ], [ %n, %entry ]
16    ret i32 %answ.0
17  }
18
```

# Example - fibonacci

**icmp: integer comparison**
**ult: unsigned comparison, less than**

```c
unsigned fib(unsigned n) {
    unsigned answ;
    if (n <= 1)
        answ = n;
    else
        answ = fib(n - 1) + fib(n - 2);
    return answ;
}
```

```
1   define i32 @fib(i32 %n) {
2   entry:
3     %cmp = icmp ult i32 %n, 2
4     br i1 %cmp, label %if.end, label %if.else
5
6   if.else:
7     %sub = add i32 %n, -1
8     %call = call i32 @fib(i32 %sub)
9     %sub1 = add i32 %n, -2
10    %call2 = call i32 @fib(i32 %sub1)
11    %add = add i32 %call2, %call
12    br label %if.end
13
14  if.end:
15    %answ.0 = phi i32 [ %add, %if.else ], [ %n, %entry ]
16    ret i32 %answ.0
17  }
18
```

**LLVM Language Reference Manual: https://llvm.org/docs/LangRef.html**

# Example - fibonacci

```c
unsigned fib(unsigned n) {
    unsigned answ;
    if (n <= 1)
        answ = n;
    else
        answ = fib(n - 1) + fib(n - 2);
    return answ;
}
```

**Note that the branch condition is inversed**
**There is no special reason in this case.. :/**

```
1   define i32 @fib(i32 %n) {
2   entry:
3     %cmp = icmp ult i32 %n, 2
4     br i1 %cmp, label %if.end, label %if.else
5
6   if.else:
7     %sub = add i32 %n, -1
8     %call = call i32 @fib(i32 %sub)
9     %sub1 = add i32 %n, -2
10    %call2 = call i32 @fib(i32 %sub1)
11    %add = add i32 %call2, %call
12    br label %if.end
13
14  if.end:
15    %answ.0 = phi i32 [ %add, %if.else ], [ %n, %entry ]
16    ret i32 %answ.0
17  }
18
```

# Example - fibonacci

```c
unsigned fib(unsigned n) {
    unsigned answ;
    if (n <= 1)
        answ = n;
    else
        answ = fib(n - 1) + fib(n - 2);
    return answ;
}
```

```
1   define i32 @fib(i32 %n) {
2   entry:
3     %cmp = icmp ult i32 %n, 2
4     br i1 %cmp, label %if.end, label %if.else
5
6   if.else:
7     %sub = add i32 %n, -1
8     %call = call i32 @fib(i32 %sub)
9     %sub1 = add i32 %n, -2
10    %call2 = call i32 @fib(i32 %sub1)
11    %add = add i32 %call2, %call
12    br label %if.end
13
14  if.end:
15    %answ.0 = phi i32 [ %add, %if.else ], [ %n, %entry ]
16    ret i32 %answ.0
17  }
18
```

# Example - fibonacci

```c
unsigned fib(unsigned n) {
    unsigned answ;
    if (n <= 1)
        answ = n;
    else
        answ = fib(n - 1) + fib(n - 2);
    return answ;
}
```

```
1   define i32 @fib(i32 %n) {
2   entry:
3     %cmp = icmp ult i32 %n, 2
4     br i1 %cmp, label %if.end, label %if.else
5
6   if.else:
7     %sub = add i32 %n, -1
8     %call = call i32 @fib(i32 %sub)
9     %sub1 = add i32 %n, -2
10    %call2 = call i32 @fib(i32 %sub1)
11    %add = add i32 %call2, %call
12    br label %if.end
13
14  if.end:
15    %answ.0 = phi i32 [ %add, %if.else ], [ %n, %entry ]
16    ret i32 %answ.0
17  }
18
```

# Example - fibonacci

```c
unsigned fib(unsigned n) {
    unsigned answ;
    if (n <= 1)
        answ = n;
    else
        answ = fib(n - 1) + fib(n - 2);
    return answ;
}
```

```
1   define i32 @fib(i32 %n) {
2   entry:
3     %cmp = icmp ult i32 %n, 2
4     br i1 %cmp, label %if.end, label %if.else
5
6   if.else:
7     %sub = add i32 %n, -1
8     %call = call i32 @fib(i32 %sub)
9     %sub1 = add i32 %n, -2
10    %call2 = call i32 @fib(i32 %sub1)
11    %add = add i32 %call2, %call
12    br label %if.end
13
14  if.end:
15    %answ.0 = phi i32 [ %add, %if.else ], [ %n, %entry ]
16    ret i32 %answ.0
17  }
18
```

# Example - fibonacci

```c
unsigned fib(unsigned n) {
    unsigned answ;
    if (n <= 1)
        answ = n;
    else
        answ = fib(n - 1) + fib(n - 2);
    return answ;
}
```

**Multiple definitions of variables in different bloc
are merged with a phi node.**

```
1   define i32 @fib(i32 %n) {
2   entry:
3     %cmp = icmp ult i32 %n, 2
4     br i1 %cmp, label %if.end, label %if.else
5
6   if.else:
7     %sub = add i32 %n, -1
8     %call = call i32 @fib(i32 %sub)
9     %sub1 = add i32 %n, -2
10    %call2 = call i32 @fib(i32 %sub1)
11    %add = add i32 %call2, %call
12    br label %if.end
13
14  if.end:
15    %answ.0 = phi i32 [ %add, %if.else ], [ %n, %entry ]
16    ret i32 %answ.0
17  }
18
```

# Example - fibonacci

```c
unsigned fib(unsigned n) {
    unsigned answ;
    if (n <= 1)
        answ = n;
    else
        answ = fib(n - 1) + fib(n - 2);
    return answ;
}
```

```
1   define i32 @fib(i32 %n) {
2   entry:
3     %cmp = icmp ult i32 %n, 2
4     br i1 %cmp, label %if.end, label %if.else
5
6   if.else:
7     %sub = add i32 %n, -1
8     %call = call i32 @fib(i32 %sub)
9     %sub1 = add i32 %n, -2
10    %call2 = call i32 @fib(i32 %sub1)
11    %add = add i32 %call2, %call
12    br label %if.end
13
14  if.end:
15    %answ.0 = phi i32 [ %add, %if.else ], [ %n, %entry ]
16    ret i32 %answ.0
17  }
18
```

# Play with fibonacci

- Please store the C program as fib.c
- C -> IR:

```
bin/clang -S -emit-llvm -O1 -g0 \
          -fno-discard-value-names fib.c -o -
```

- IR -> Assembly:

```
bin/llc -o fib.s fib.ll
```

# Example 2 - average

```
1   double answer;
2
3   void average(double *numbers) {
4       double x = numbers[0];
5       double y = numbers[1];
6       answer = (x + y) / 2;
7   }
```

# Example 2 - average

```
1   double answer;
2
3   void average(double *numbers) {
4       double x = numbers[0];
5       double y = numbers[1];
6       answer = (x + y) / 2;
7   }
```

```
1   @answer = global double 0.000000e+00
2
3   define void @average(double* %numbers) {
4   entry:
5     %0 = load double, double* %numbers
6     %arrayidx1 = getelementptr inbounds double, double* %numbers, i64 1
7     %1 = load double, double* %arrayidx1
8     %add = fadd double %0, %1
9     %div = fmul double %add, 5.000000e-01
10    store double %div, double* @answer
11    ret void
12  }
```

# Example 2 - average

```
1   double answer;
2
3   void average(double *numbers) {
4       double x = numbers[0];
5       double y = numbers[1];
6       answer = (x + y) / 2;
7   }
```

**Global variables have prefix @**

```
1   @answer = global double 0.000000e+00
2
3   define void @average(double* %numbers) {
4   entry:
5     %0 = load double, double* %numbers
6     %arrayidx1 = getelementptr inbounds double, double* %numbers, i64 1
7     %1 = load double, double* %arrayidx1
8     %add = fadd double %0, %1
9     %div = fmul double %add, 5.000000e-01
10    store double %div, double* @answer
11    ret void
12  }
```

# Example 2 - average

```
1   double answer;
2
3   void average(double *numbers) {
4       double x = numbers[0];
5       double y = numbers[1];
6       answer = (x + y) / 2;
7   }
```

**Dereference** `%numbers`

A variable with numeric name (should increase by 1)

```
ble 0.000000e+00

3   define void @average(double* %numbers) {
4   entry:
5     %0 = load double, double* %numbers
6     %arrayidx1 = getelementptr inbounds double, double* %numbers, i64 1
7     %1 = load double, double* %arrayidx1
8     %add = fadd double %0, %1
9     %div = fmul double %add, 5.000000e-01
10    store double %div, double* @answer
11    ret void
12  }
```

# Example 2 - average

```
1   double answer;
2
3   void average(double *numbers) {
4       double x = numbers[0];
5       double y = numbers[1];
6       answer = (x + y) / 2;
7   }
```

`numbers[1]` **is** `*(numbers + 1)`
**Let's calculate** `(numbers + 1)` **first**

```
1   @answer = global double 0.000000e+00
2
3   define void @average(double* %numbers) {
4   entry:
5     %0 = load double, double* %numbers
6     %arrayidx1 = getelementptr inbounds double, double* %numbers, i64 1
7     %1 = load double, double* %arrayidx1
8     %add = fadd double %0, %1
9     %div = fmul double %add, 5.000000e-01
10    store double %div, double* @answer
11    ret void
12  }
```

# Example 2 - average

```
1   double answer;
2
3   void average(double *numbers) {
4       double x = numbers[0];
5       double y = numbers[1];
6       answer = (x + y) / 2;
7   }
```

**Dereference** `(numbers+1)`

```
1   @answer = global double 0.000000e+00
2
3   define void @average(double* %numbers) {
4   entry:
5     %0 = load double, double* %numbers
6     %arrayidx1 = getelementptr inbounds double, double* %numbers, i64 1
7     %1 = load double, double* %arrayidx1
8     %add = fadd double %0, %1
9     %div = fmul double %add, 5.000000e-01
10    store double %div, double* @answer
11    ret void
12  }
```

# Example 2 - average

```
1   double answer;
2
3   void average(double *numbers) {
4       double x = numbers[0];
5       double y = numbers[1];
6       answer = (x + y) / 2;
7   }
```

**Calculate its average**

```
1   @answer = global double 0.000000e+00
2
3   define void @average(double* %numbers) {
4   entry:
5     %0 = load double, double* %numbers
6     %arrayidx1 = getelementptr inbounds double, double* %numbers, i64 1
7     %1 = load double, double* %arrayidx1
8     %add = fadd double %0, %1
9     %div = fmul double %add, 5.000000e-01
10    store double %div, double* @answer
11    ret void
12  }
```

# Example 2 - average

```
1   double answer;
2
3   void average(double *numbers) {
4       double x = numbers[0];
5       double y = numbers[1];
6       answer = (x + y) / 2;
7   }
```

**Store the result to a global variable**

```
1   @answer = global double 0.000000e+00
2
3   define void @average(double* %numbers) {
4   entry:
5     %0 = load double, double* %numbers
6     %arrayidx1 = getelementptr inbounds double, double* %numbers, i64 1
7     %1 = load double, double* %arrayidx1
8     %add = fadd double %0, %1
9     %div = fmul double %add, 5.000000e-01
10    store double %div, double* @answer
11    ret void
12  }
```