Refactoring with Clang

For fun & benefits

C++ FRUG #18
@jeremydemeule
jeremy.demeule@gmail.com

What is Clang?

The C-family compiler infrastructure of the LLVM compiler infrastructure

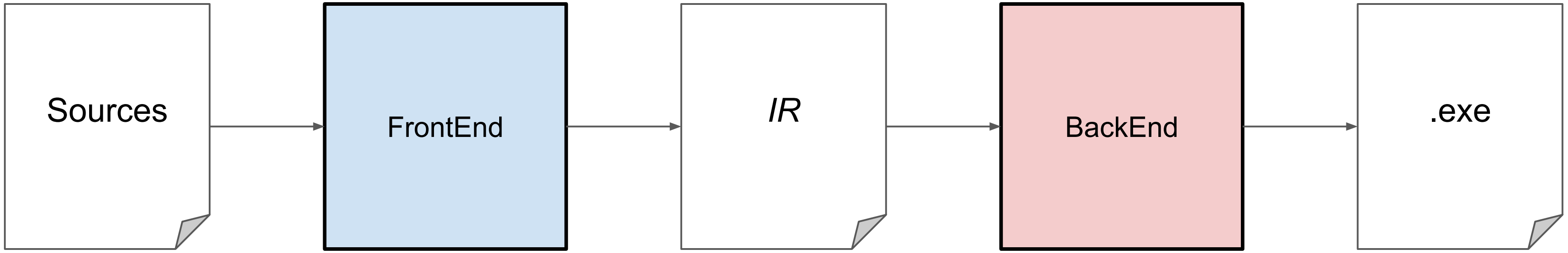
A production quality C++ compiler

A modern compiler designed as an library

Modern Compiler

Separation between FrontEnd / BackEnd

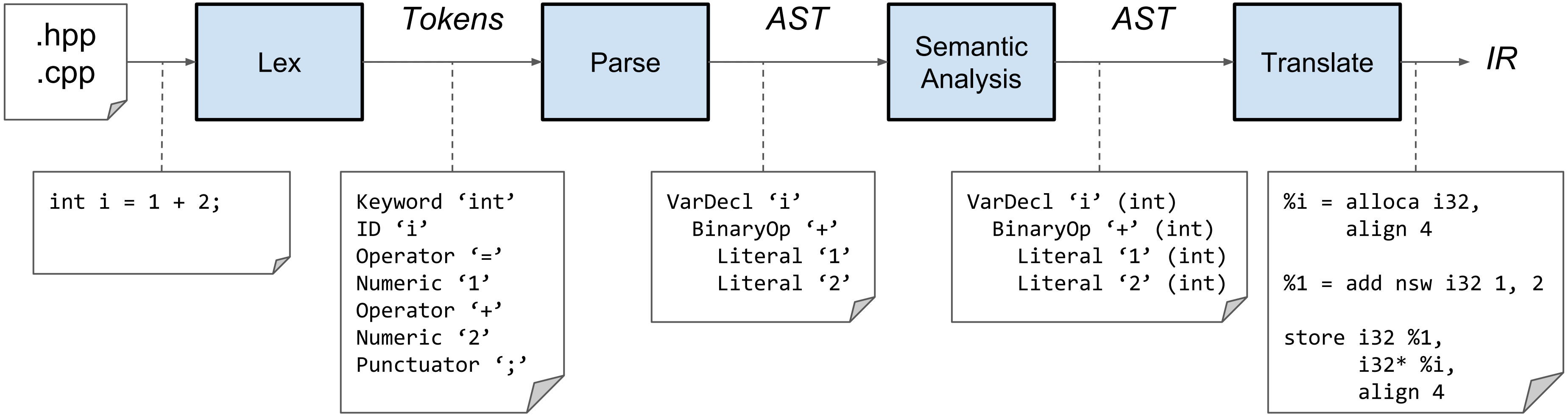
- Better portability
- Powerful optimizations (inlining, SSA…)



Modern Compiler

FrontEnd / Clang

Focus on Front End



```
int i = 1 + 2;
```

```
Keyword ‘int’
ID ‘i’
Operator ‘=’
Numeric ‘1’
Operator ‘+’
Numeric ‘2’
Punctuator ‘;’
```

```
VarDecl ‘i’
  BinaryOp ‘+’
    Literal ‘1’
    Literal ‘2’
```

```
VarDecl ‘i’ (int)
  BinaryOp ‘+’ (int)
    Literal ‘1’ (int)
    Literal ‘2’ (int)
```

```
%i = alloca i32,
     align 4

%1 = add nsw i32 1, 2

store i32 %1,
      i32* %i,
      align 4
```

Let’s talk about refactoring

How to rename A::getValue to A::getInt ?

```
class A {
public:
   int getValue() const;
};

int A::getValue() const { return 42; }


int main(int argc, char** argv) {
   A a;

   return a.getValue();
}
```

Let’s talk about refactoring

How to rename A::getValue to A::getInt ?

```
class A {
public:
   int getValue() const;
};

int A::getValue() const { return 42; }
```

```
class B {
public:
   int getValue() const;
};

int B::getValue() const { return 23; }
```

```
int main(int argc, char** argv) {
   A a;
   B b;

   return a.getValue() + b.getValue();
}
```

Let’s talk about refactoring

What if?

- I have to work on massive codeline...
- I want to remove parameters...
- I want to add some template args…

```

void*
createDBStorage(KeyFunctionCB pItemKeyFn,
                SelectFunction pSelectFunction,
                MISHORT iItemSize,    // <- sizeof(T)
                MISHORT iItemKeySize, // <- sizeof(K)
                MISHORT iItemKS,
                MISHORT iItemPerBlock,
                MILONG  lMaxItem,
                MISHORT iKeyItemPerBlock,
                MILONG  lKeyMaxItem,
                MISHORT iID);
```

```
template <typename T, typename K>
void*
createDBStorage(KeyFunctionCB pItemKeyFn,
                SelectFunction pSelectFunction,


                MISHORT iItemKS,
                MISHORT iItemPerBlock,
                MILONG  lMaxItem,
                MISHORT iKeyItemPerBlock,
                MILONG  lKeyMaxItem,
                MISHORT iID);
```

Let’s build a tool to refactor for us

Too much parameters, let’s simplify the example.

```cpp
template <typename ItemT>
void foo(int iID);

// Deprecated
void foo(int iItemSize, // <- sizeof(T)
         int iID);

struct Item {/*...*/};

int main() {
   foo(sizeof(Item), 42);
}
```

```cpp
template <typename ItemT>
void foo(int iID);

// Deprecated
void foo(int iItemSize, // <- sizeof(T)
         int iID);

struct Item {/*...*/};

int main() {
   foo<Item>(42);
}
```

From 2 dynamic parameters where the first is sizeof(...) -> only 1 dynamic parameter and 1 statically computed.

Let’s build a tool to refactor for us

Did we succeed?

Let’s build a tool to refactor for us

This technique was already used:

- To change functions signature
- Add **`const`** before `char*` automatically

Some tools already exists

- Static analysis - *clang-check / clang-tidy*
- Code formatting - *clang-format*
- Code refactoring - *clang-rename (will be replaced by clang-refactor)*
- Code linter / updater - *clang-tidy*

The end

Any Questions?

Credit: @FioraAeterna