Lecture 2

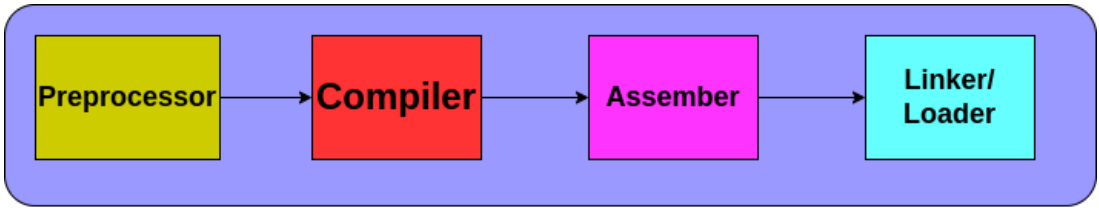
Compiler Design

Awanish Pandey

Department of Computer Science and Engineering
Indian Institute of Technology
Roorkee

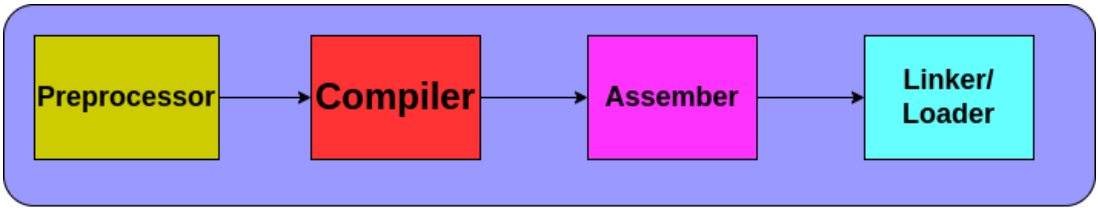
January 22, 2025

Basic flow



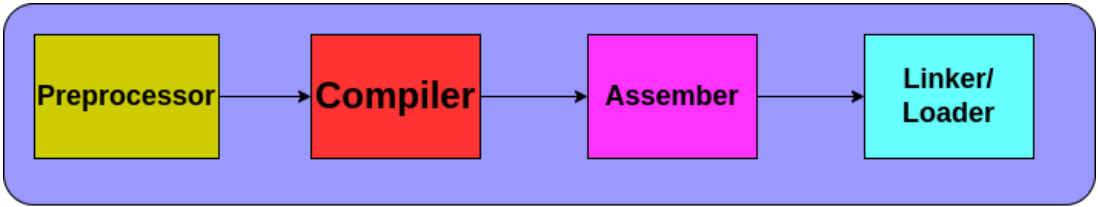
- `gcc --save-temps fileName.c` → it will save all the temporary files being created in process of compilation of c file.

Basic flow



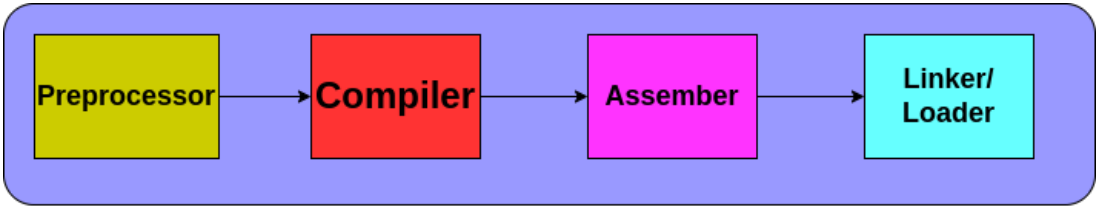
- `gcc --save-temps fileName.c`
- Preprocessing: includes header file and macro expansion.

Basic flow



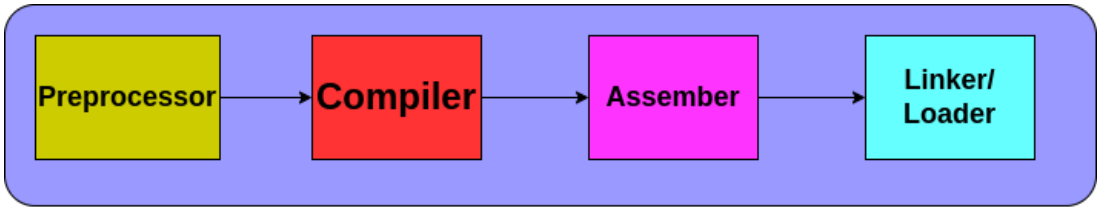
- `gcc --save-temps fileName.c`
- Preprocessing: includes header file and macro expansion.
- Compiler: generate assembly file

Basic flow



- `gcc --save-temps fileName.c`
- Preprocessing: includes header file and macro expansion.
- Compiler: generate assembly file
- Assembler: generate the relocatable object file

Basic flow



- `gcc --save-temps fileName.c`
- Preprocessing: includes header file and macro expansion.
- Compiler: generate assembly file
- Assembler: generate the relocatable object file
- Linker/loader: link different object files into a single binary. Load it to the main memory.

How to translate easily

- Translate in steps. Each step handles a reasonably simple, logical, and well-defined task

How to translate easily

- Translate in steps. Each step handles a reasonably simple, logical, and well-defined task
- Design a series of program representations

How to translate easily

- Translate in steps. Each step handles a reasonably simple, logical, and well-defined task
- Design a series of program representations
- Intermediate representations should be amenable to program manipulation of various kinds (type checking, optimization, code generation etc.)

How to translate easily

- Translate in steps. Each step handles a reasonably simple, logical, and well-defined task
- Design a series of program representations
- Intermediate representations should be amenable to program manipulation of various kinds (type checking, optimization, code generation etc.)
- Representations become more machine-specific and less language-specific as the translation proceeds

The first few steps

- The first few steps can be understood by analogies to how humans comprehend a natural language

The first few steps

- The first few steps can be understood by analogies to how humans comprehend a natural language
- The first step is recognizing/knowing alphabets of a language. For example

The first few steps

- The first few steps can be understood by analogies to how humans comprehend a natural language
- The first step is recognizing/knowing alphabets of a language. For example
  - English text consists of lower and upper case alphabets, digits, punctuations and white spaces

The first few steps

- The first few steps can be understood by analogies to how humans comprehend a natural language
- The first step is recognizing/knowing alphabets of a language. For example
  - English text consists of lower and upper case alphabets, digits, punctuations and white spaces
  - Written programs consist of characters from the ASCII characters set (normally 9-13, 32-126)

The first few steps

- The first few steps can be understood by analogies to how humans comprehend a natural language
- The first step is recognizing/knowing alphabets of a language. For example
  - English text consists of lower and upper case alphabets, digits, punctuations and white spaces
  - Written programs consist of characters from the ASCII characters set (normally 9-13, 32-126)
- The next step to understand the sentence is recognizing words (lexical analysis)

The first few steps

- The first few steps can be understood by analogies to how humans comprehend a natural language
- The first step is recognizing/knowing alphabets of a language. For example
  - English text consists of lower and upper case alphabets, digits, punctuations and white spaces
  - Written programs consist of characters from the ASCII characters set (normally 9-13, 32-126)
- The next step to understand the sentence is recognizing words (lexical analysis)
  - English language words can be found in dictionaries

The first few steps

- The first few steps can be understood by analogies to how humans comprehend a natural language
- The first step is recognizing/knowing alphabets of a language. For example
  - English text consists of lower and upper case alphabets, digits, punctuations and white spaces
  - Written programs consist of characters from the ASCII characters set (normally 9-13, 32-126)
- The next step to understand the sentence is recognizing words (lexical analysis)
  - English language words can be found in dictionaries
  - Programming languages have a dictionary (keywords etc.) and rules for constructing words (identifiers, numbers etc.)

Lexical Analysis

- Recognizing words is not completely trivial. For example:

Lexical Analysis

- Recognizing words is not completely trivial. For example:
  ist his ase nte nce?

Lexical Analysis

- Recognizing words is not completely trivial. For example: ist his ase nte nce?
- Therefore, we must know what the word separators are

Lexical Analysis

- Recognizing words is not completely trivial. For example:
  ist his ase nte nce?
- Therefore, we must know what the word separators are
- The language must define rules for breaking a sentence into a sequence of words.

Lexical Analysis

- Recognizing words is not completely trivial. For example:
  ist his ase nte nce?
- Therefore, we must know what the word separators are
- The language must define rules for breaking a sentence into a sequence of words.
- Normally, white spaces and punctuations are word separators in languages.

Lexical Analysis

- Recognizing words is not completely trivial. For example:
  ist his ase nte nce?
- Therefore, we must know what the word separators are
- The language must define rules for breaking a sentence into a sequence of words.
- Normally, white spaces and punctuations are word separators in languages.
- In programming languages, a character from a different class may also be treated as word separator.

Lexical Analysis

- Recognizing words is not completely trivial. For example:
  ist his ase nte nce?
- Therefore, we must know what the word separators are
- The language must define rules for breaking a sentence into a sequence of words.
- Normally, white spaces and punctuations are word separators in languages.
- In programming languages, a character from a different class may also be treated as word separator.
- The lexical analyzer breaks a sentence into a sequence of words or tokens:

Lexical Analysis

- Recognizing words is not completely trivial. For example:
  ist his ase nte nce?
- Therefore, we must know what the word separators are
- The language must define rules for breaking a sentence into a sequence of words.
- Normally, white spaces and punctuations are word separators in languages.
- In programming languages, a character from a different class may also be treated as word separator.
- The lexical analyzer breaks a sentence into a sequence of words or tokens:
  - `if a == b then a = 1 ; else a = 2 ;`

Lexical Analysis

- Recognizing words is not completely trivial. For example:
  ist his ase nte nce?
- Therefore, we must know what the word separators are
- The language must define rules for breaking a sentence into a sequence of words.
- Normally, white spaces and punctuations are word separators in languages.
- In programming languages, a character from a different class may also be treated as word separator.
- The lexical analyzer breaks a sentence into a sequence of words or tokens:
  - `if a == b then a = 1 ; else a = 2 ;`
  - Sequence of words (total 14 words)
    `"if", "a", "==", "b", "then", "a", "=", "1", ";", "else", "a", "=", "2", ";",`

total 14 tokens are present in the given statement.

What next?

- Understand the structure of the sentence.

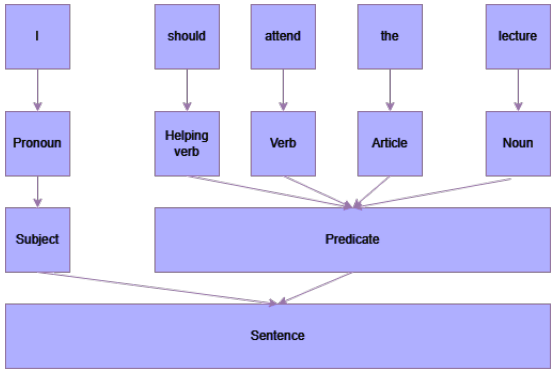
What next?

- Understand the structure of the sentence.
- this is called **syntax checking / parsing**

What next?

- Understand the structure of the sentence.
- this is called **syntax checking / parsing**

checking if the given statement is structurally correct or not.



Understanding the meaning

- Once the sentence structure is understood, we try to understand the sentence's meaning (semantic analysis).

Understanding the meaning

- Once the sentence structure is understood, we try to understand the sentence's meaning (semantic analysis).
- Prateek said Nitin left his assignment at home.

Understanding the meaning

- Once the sentence structure is understood, we try to understand the sentence's meaning (semantic analysis).
- Prateek said Nitin left his assignment at home.
- Amit said Amit left his assignment at home.

Understanding the meaning

- Once the sentence structure is understood, we try to understand the sentence's meaning (semantic analysis).
- Prateek said Nitin left his assignment at home.
- Amit said Amit left his assignment at home.
- Compiler's solution

Understanding the meaning

- Once the sentence structure is understood, we try to understand the sentence's meaning (semantic analysis).
- Prateek said Nitin left his assignment at home.
- Amit said Amit left his assignment at home.
- Compiler's solution
  - Scoping

Understanding the meaning

- Once the sentence structure is understood, we try to understand the sentence's meaning (semantic analysis).
- Prateek said Nitin left his assignment at home.
- Amit said Amit left his assignment at home.
- Compiler's solution
  - Scoping
  - Namespaces

Understanding the meaning

- Once the sentence structure is understood, we try to understand the sentence's meaning (semantic analysis).
- Prateek said Nitin left his assignment at home.
- Amit said Amit left his assignment at home.
- Compiler's solution
  - Scoping
  - Namespaces
  - Variable binding

Understanding the meaning

- Once the sentence structure is understood, we try to understand the sentence's meaning (semantic analysis).
- Prateek said Nitin left his assignment at home.
- Amit said Amit left his assignment at home.
- Compiler's solution
  - Scoping
  - Namespaces
  - Variable binding
- Amit left her work at home.

Understanding the meaning

- Once the sentence structure is understood, we try to understand the sentence's meaning (semantic analysis).
- Prateek said Nitin left his assignment at home.
- Amit said Amit left his assignment at home.
- Compiler's solution
  - Scoping
  - Namespaces
  - Variable binding
- Amit left her work at home.
- Torid/Valdis/Torbjorg left her bag in the car.

Understanding the meaning

- Once the sentence structure is understood, we try to understand the sentence's meaning (semantic analysis).
- Prateek said Nitin left his assignment at home.
- Amit said Amit left his assignment at home.

"his" corresponds to whom?

- Compiler's solution
  - Scoping
  - Namespaces
  - Variable binding
- Amit left her work at home.
- Torid/Valdis/Torbjorg left her bag in the car.
- Is this correct? `a = b + c;` based on **C** Programming Language

will depend on the declaration types of variables a, b and c and compiler (do compiler allow different types to be added together or not)

Front End Passes

```
if (b == 0) a = b;
```

Front End Passes

```
if (b == 0) a = b;
```

- Lexical Analysis

Front End Passes

```
if (b == 0) a = b;
```

- Lexical Analysis
  - Recognize tokens
  - Error reporting

Front End Passes

```
if (b == 0) a = b;
```

- Lexical Analysis
  - Recognize tokens
  - Error reporting
- Syntax Analysis
  - Check syntax and construct AST
  - Error reporting and recovery

Front End Passes

```
if (b == 0) a = b;
```

- Lexical Analysis
  - Recognize tokens
  - Error reporting
- Syntax Analysis
  - Check syntax and construct AST
  - Error reporting and recovery
- Semantic Analysis
  - Disambiguate overloaded operator
  - Type checking
  - Error reporting

Code Optimization

- Précis writing

Code Optimization

- Précis writing
- Run faster

Code Optimization

- Précis writing
- Run faster
- Use fewer resources (memory, registers, space, fewer fetches)

Code Optimization

- Précis writing
- Run faster
- Use fewer resources (memory, registers, space, fewer fetches)
- Common optimization

Code Optimization

- Précis writing
- Run faster
- Use fewer resources (memory, registers, space, fewer fetches)
- Common optimization
  - Common sub-expression elimination

Code Optimization

- Précis writing
- Run faster
- Use fewer resources (memory, registers, space, fewer fetches)
- Common optimization
  - Common sub-expression elimination
  - Copy propagation

Code Optimization

- Précis writing
- Run faster
- Use fewer resources (memory, registers, space, fewer fetches)
- Common optimization
  - Common sub-expression elimination
  - Copy propagation
  - Dead code elimination

Code Optimization

- Précis writing
- Run faster
- Use fewer resources (memory, registers, space, fewer fetches)
- Common optimization
  - Common sub-expression elimination
  - Copy propagation
  - Dead code elimination
  - Code motion

Code Optimization

- Précis writing
- Run faster
- Use fewer resources (memory, registers, space, fewer fetches)
- Common optimization
  - Common sub-expression elimination
  - Copy propagation
  - Dead code elimination
  - Code motion
  - Strength reduction

Code Optimization

- Précis writing
- Run faster
- Use fewer resources (memory, registers, space, fewer fetches)
- Common optimization
  - Common sub-expression elimination
  - Copy propagation
  - Dead code elimination
  - Code motion
  - Strength reduction
  - Constant folding

Code Optimization

- Précis writing
- Run faster
- Use fewer resources (memory, registers, space, fewer fetches)
- Common optimization
  - Common sub-expression elimination
  - Copy propagation
  - Dead code elimination
  - Code motion
  - Strength reduction
  - Constant folding

$PI = 3.14159$
$Area = 4 * PI * R^2$
$Volume = (4/3) * PI * R^3$

Code Optimization

- Précis writing
- Run faster
- Use fewer resources (memory, registers, space, fewer fetches)
- Common optimization
  - Common sub-expression elimination
  - Copy propagation
  - Dead code elimination
  - Code motion
  - Strength reduction
  - Constant folding

Code motion, also known as frequency reduction,

when doing only copying of some variables like b = a and then using b instead of a, then replace b with a until b's or a's value gets changed.

replace the value of constant to where it has been used in the program at compile time only

$$PI = 3.14159$$
$$Area = 4 * PI * R^2$$
$$Volume = (4/3) * PI * R^3$$

$$X = R * R$$
$$Area = 12.56636 * X$$
$$Volume = 4.18879 * X * R$$