



Universidade Federal de Uberlândia
Faculdade de Computação



Projeto da Disciplina

Curso de Bacharelado em Ciência da Computação
GBC071 - Construção de Compiladores
Prof. Luiz Gustavo Almeida Martins

Visão Geral do Projeto

- Características gerais:
 - A gramática da linguagem deve ser representada na notação EBNF
 - Uso de palavras reservadas no analisador léxico
- Foco no **front-end** do compilador
 - Não será implementada as etapas de otimização e geração de código
 - Será adotado algum *back-end* disponível no ambiente de compilação
- Uso do ambiente de compilação **LLVM**
 - Deve-se gerar uma IR compatível com o ambiente
- **Semântica estática:**
 - Realizada em tempo de compilação
 - Inserção de código no arquivo de especificação do *parser+lexer*
- **Compilador de um passo:**
 - Todas as fases entrelaçadas
 - Compilador dirigido por sintaxe



Universidade Federal de Uberlândia
Faculdade de Computação



Projeto: Especificação da Linguagem

Componentes Básicos

- Estrutura principal:

- Sintaxe:

programa

bloco

- *program* funciona similar ao *int main()* do C

- Bloco:

- Sintaxe:

inicio

declaração

comandos

fim

Componentes Básicos

- **Declaração de variáveis:**

- **Sintaxe:**

tipo id;

- Cada variável deve ter seu tipo definido explicitamente
 - **Ex:** *int x; int y;*
 - Inicialmente usaremos apenas o tipo *int*
 - Deve garantir constantes com valores entre -32768 e +32767 (semântica)

- **Comando de seleção**

- **Sintaxe:**

se (cond)

bloco

Componentes Básicos

- **Comando de repetição:**

- **Sintaxe:**

enquanto (cond)
bloco

- **Comando de atribuição:**

- **Sintaxe:**

id = expressao;

Componentes Básicos

- **Condições:**

- Permite **operadores relacionais**

- Igual (`==`), diferente (`<>`), menor (`<`), maior (`>`), menor ou igual (`<=`), maior ou igual (`>=`)

- **Expressões:**

- Permite **operadores aritméticos**

- Soma (`+`), subtração (`-`), multiplicação (`*`), divisão (`/`)

- Permite **constantes** compatíveis com os tipos definidos na linguagem (*int*, *char*, *real*)

Variações

- **Declaração de variáveis:**

- Declaração de estruturas homogêneas (vetores)
 - **Sintaxe:** *tipo[tam] id;*
- Outros tipos de dados
 - *char* e *real*
- Declaração implícita do tipo
 - **Sintaxe:** *tipo id₁, id₂, ..., id_N;*

- **Comando de seleção (*if-then-else*):**

se (*cond*)

bloco

else

bloco

Variações

- **Comandos de repetição:**

- Implementar o comando ***do-while***

- **Sintaxe:**

faça

bloco

enquanto (condição);

- Implementar o comando ***for***

- **Sintaxe:**

para (atribuição; condição; atribuição)

bloco

- Na **análise semântica** deve-se verificar se a variável do lado esquerdo de ambas as atribuições é a mesma

Variações

- **Comando de atribuição:**

- Permite variáveis indexadas do lado esquerdo
- **Sintaxe:**

id[pos] = expressao;

- **Condições:**

- Permitir **operadores lógicos** (*not, and, or*) aplicados em condições relacionais

Ambiente de Compilação

- **Compilador LLVM** (site: <https://llvm.org/>)
 - Execução dos componentes (*toolchains*) por linha de comando
 - Similar ao GCC
 - Usa *flags* para direcionar/personalizar a compilação
 - Ex: *-lm* para funções matemáticas
 - Plataformas suportadas (fonte: *llvm.org*):

OS	Arquitetura	Compiladores
Linux	x861	GCC, Clang
Linux	amd64	GCC, Clang
Linux	ARM	GCC, Clang
Linux	PowerPC	GCC, Clang
Solaris	V9 (Ultrasparc)	GCC
FreeBSD	x861	GCC, Clang
FreeBSD	amd64	GCC, Clang
NetBSD	x861	GCC, Clang
NetBSD	amd64	GCC, Clang
MacOS2	PowerPC	GCC
MacOS	x86	GCC, Clang
Win32 (Cigwin)	x861, 3	GCC
Windows	x861	Visual Studio
Win64	x86-64	Visual Studio

Ambiente de Compilação

- **Compilação direta:**

- Sintaxe: **clang -o exeCode sourceCode.c**

- **Compilação em etapas:**

- **Análise (front-end):**

- Sintaxe: **clang sourceCode.c -emit-llvm -S -o IRCode.ll**
- **-emit-llvm** deve ser usado com as opções **-S** para gerar IR (.ll) ou **-c** para gerar *bitcode* (.bc)

- **Otimização (middle-end):**

- Sintaxe: **opt <seq> IRCode.ll -S -o IRCodeOptim.ll**
- **<seq>** representa a sequência de otimização que deve ser aplicada na IR
 - Ex: -O1, -O2, -O3, “-tti -tbaa -verify -domtree -sroa -early-cse -basicaa -aa -gvn-hoist”

- **Síntase (back-end):**

- Código Assembly: **llc IRCode.ll -o asmCode.s**
- Código de máquina: **clang -o exeCode IRCode.ll** OU
clang -o exeCode asmCode.s OU
gcc asmCode.s -o exeCode (alternativa com GCC)

Primeira Etapa do Projeto

- Para cada elemento estrutural da linguagem, verificar como seria a IR no LLVM
 - **1º passo:** construir um **programa vazio** em C (sem declarações e comandos na *main()*) e executar o **front-end CLANG** e analisar a IR gerada
 - **2º passo:** incluir cada estrutura pretendida e verificar as mudanças na IR
- Gerar um relatório apresentando os códigos testados, as IR obtidas e uma descrição das observações (**mapeamento entre os códigos**)

Primeira Etapa do Projeto



```
int main()
```

```
{  
  
}
```

/* Código fonte */

```
; ModuleID = 'teste.c'
```

```
target datalayout = "e-m:e-i64:64-f80:128-n8:16:32:64-S128"
```

```
target triple = "x86_64-unknown-linux-gnu"
```

```
; Function Attrs: nounwind uwtable
```

```
define i32 @main() #0 {
```

```
ret i32 0
```

```
}
```

```
attributes #0 = { nounwind uwtable "disable-tail-calls"="false" "less-  
precise-fpmad"="false" "no-frame-pointer-elim"="true" "no-frame-pointer-  
elim-non-leaf" "no-infs-fp-math"="false" "no-nans-fp-math"="false" "  
"stack-protector-buffer-size"="8" "target-cpu"="x86-64" "target-  
features"="+sse,+sse2" "unsafe-fp-math"="false" "use-soft-  
float"="false" }
```

```
!llvm.ident = !{!0}
```

```
!0 = !{"clang version 3.7.1 (tags/RELEASE_371/final)"}
```

/* Código intermediário (SSA) */

Primeira Etapa do Projeto

```
int main()
```

```
{
```

```
    int x;
```

```
}
```

```
/* Código fonte */
```

```
; ModuleID = 'teste.c'
```

```
target datalayout = "e-m:e-i64:64-f80:128-n8:16:32:64-S128"
```

```
target triple = "x86_64-unknown-linux-gnu"
```

```
; Function Attrs: nounwind uwtable
```

```
define i32 @main() #0 {
```

```
    %x = alloca i32, align 4
```

```
    ret i32 0
```

```
}
```

```
attributes #0 = { nounwind uwtable "disable-tail-calls"="false" "less-precise-fpmad"="false" "no-frame-pointer-elim"="true" "no-frame-pointer-elim-non-leaf" "no-infs-fp-math"="false" "no-nans-fp-math"="false" "stack-protector-buffer-size"="8" "target-cpu"="x86-64" "target-features"="+sse,+sse2" "unsafe-fp-math"="false" "use-soft-float"="false" }
```

```
!llvm.ident = !{!0}
```

```
!0 = !{"clang version 3.7.1 (tags/RELEASE_371/final)"}
```

```
/* Código intermediário (SSA) */
```

Segunda Etapa do Projeto

- **Especificação da linguagem:**
 - Definição das palavras reservadas
 - Definição da gramática com as estruturas da linguagem
 - Definição das expressões regulares para a identificação dos lexemas aceitos pela linguagem
- Incluir ao relatório da etapa anterior, a lista de palavras reservadas, a gramática da linguagem e as expressões regulares para a identificação dos lexemas aceitos

Terceira Etapa do Projeto

- **Tradução direcionada por sintaxe:**
 - Implementação do analisador léxico
 - Implementação do analisador sintático
 - Implementação do analisador semântico
 - Geração do código intermediário
- Esta etapa será definida ao decorrer do curso