**UNIVERSIDADE DE SÃO PAULO**
INSTITUTO DE CIÊNCIAS MATEMÁTICAS E DE COMPUTAÇÃO
DEPARTAMENTO DE CIÊNCIAS DE COMPUTAÇÃO

SCC0217 - Linguagens de Programação e Compiladores
Prof. Adinovam Henriques de Macedo Pimenta

# TRABALHO 1

## I. OBJETIVO

O objetivo deste trabalho é implementar um analisador léxico para a linguagem de programação C- (pronuncia-se "C menos"). Esta linguagem, é um subconjunto simplificado da linguagem C.

## II. CONVENÇÕES LÉXICAS DA LINGUAGEM C-

1. As palavras-chave da linguagem são as seguintes:

   **`else if int return void while input output`**

   Todas as palavras-chave devem ser escritas em letra minúscula.
   As palavras **`input`** e **`output`** representam funções que, por questões de simplificação, substituem as funções **`scanf`** e **`printf`**, respectivamente.

2. A linguagem C- é composta pelos seguintes símbolos especiais:

   **`+ - * / < <= > >= == != = ; , ( ) [ ] { } /* */`**

   Os tokens para estes símbolos são definidos pelas tabelas abaixo:

| **Símbolo** | **Token** |
|---|---|
| + | SOMA |
| - | SUB |
| * | MUL |
| / | DIV |
| < | MENOR |
| <= | MEIGUAL |
| > | MAIOR |
| >= | MAIGUAL |
| == | IGUAL |

| | |
|---|---|
| != | DIF |
| = | ATRIB |
| ; | PV |
| , | V |
| ( | AP |
| ) | FP |
| [ | ACO |
| ] | FCO |
| { | ACH |
| } | FCH |

Os símbolos “/*” e “*/” não são considerados *tokens* e são ignorados pelo compilador.

3. Há, ainda, os *tokens* ***ID*** e ***NUM***, definidos pelas expressões regulares a seguir:

   ***ID = letra letra****
   ***NUM = digito digito****
   ***letra = a | ... | z | A | ... | Z***
   ***digito = 0 | ... | 9***

   Existe diferença entre letra maiúscula e minúscula.

4. Espaço em branco é composto por espaços, tabulações e mudanças de linha. O espaço em branco é ignorado, exceto como separador de ***IDs***, ***NUMs*** e palavras-chave.

5. Comentários são cercados pela notação usual da linguagem C /* ... */ e podem ser colocados em qualquer lugar que possa ser ocupado por um espaço em branco (ou seja, comentários não podem ser colocados dentro de *tokens*), e podem incluir mais de uma linha. Os comentários não podem ser aninhados (ou seja, não pode existir um comentário dentro de outro comentário).

## III. IMPLEMENTAÇÃO

A implementação deste trabalho pode ser feita à mão ou por meio do uso de geradores automáticos de analisador léxico.

A linguagem de implementação deverá ser a linguagem C ou C++.

## IV. FUNCIONAMENTO DO ANALISADOR LÉXICO

O analisador léxico é uma parte do projeto que será desenvolvido nesta disciplina. O nome do compilador deverá ser **gcc-** e nesta primeira etapa do trabalho este compilador irá apenas reportar erros léxicos encontrados nos programas escritos em C-.

O programador irá salvar os programas em C- com a extensão .c- e para compilar o programa ele deverá executar o programa em modo texto passando como argumento o nome do arquivo que deverá ser compilado conforme o exemplo abaixo:

```
root@linux:~$ gcc- HelloWorld.c-
```

A saída do programa deverá ser a lista de tokens encontrados. Essa lista não deverá ser impressa na tela, mas sim salva no arquivo “relatorio.txt”.

Ao encontrar um ou mais erros, o compilador deve continuar o processo de compilação.

Cada erro encontrado deverá ser reportado.

Ao final da compilação, o compilador deverá fornecer a quantidade de erros encontrada.

**Exemplo 1:** compilação do programa “soma.c-” sem erro.

```
/* Este programa calcula a soma de dois números fornecidos
pelo usuário */
void main(){
     int x; int y; int resultado;
     x = input(); y=input();
     resultado = x+y;
     output(resultado);
}
```

**Figura 1**: Conteúdo do arquivo soma.c-

```
root@linux:~$ gcc- soma.c-
```

**Figura 2**: comando para compilar o arquivo “soma.c-”

```
0 erro(s) encontrado(s)
void ID
main ID
( AP
) FP
{ ACH
int INT
x ID
; PV
int INT
y ID
; PV
int INT
resultado ID
; PV
x ID
= ATRIB
input ID
( AP
) FP
; PV
y ID
= ATRIB
input ID
( AP
) FP
; PV
resultado ID
= ATRIB
x ID
+ SOMA
y ID
; PV
output ID
( AP
resultado ID
) FP
; PV
} FCH
```

**Figura 3**: conteúdo do arquivo “relatório.txt”

**Exemplo 2:** compilação do programa "soma.c-" com erro.

```
/* Este programa calcula a soma de dois números fornecidos
pelo usuário */
void main(){
     int x; int y; int resu§ltado;
     x = input(); y=input();
     resultado = x+y;
     output(resultado);
}
```

**Figura 1**: Conteúdo do arquivo soma.c-

```
root@linux:~$ gcc- soma.c-
```

**Figura 2**: comando para compilar o arquivo "soma.c-"

```
1 erro(s) encontrado(s)
void ID
main ID
( AP
) FP
{ ACH
int INT
x ID
; PV
int INT
y ID
; PV
int INT
resu ID
§ ERRO
ltado ID
; PV
x ID
= ATRIB
input ID
( AP
) FP
; PV
y ID
= ATRIB
input ID
( AP
) FP
; PV
resultado ID
= ATRIB
x ID
+ SOMA
y ID
; PV
output ID
( AP
resultado ID
) FP
; PV
} FCH
```

**Figura 3**: conteúdo do arquivo "relatório.txt"

## V. ENTREGÁVEIS

O grupo deve criar um arquivo chamado “nomes.txt” contendo o nome completo do aluno, o número USP e a turma em que está matriculado.

Todos os arquivos usados para construir o projeto (*.c, *.cpp, *.h) e o arquivo “nomes.txt” devem ser compactados em um único arquivo que deverá ser enviado para o meu e-mail (adinovam@icmc.usp.br) com o assunto “Compiladores – Trabalho 1”. Este arquivo deverá ser enviado por apenas um representante do grupo.

## VI. GRUPOS

Este trabalho deve ser feito por grupos de até 5 pessoas. Um mesmo grupo pode ser composto por alunos de turmas diferentes.

## VII. DATA DE ENTREGA

O trabalho deverá ser entregue por e-mail até o dia 2 de abril de 2018. Quem perder o prazo receberá uma penalidade na nota de 1 ponto por dia de atraso.