Avaliação 1 - 05 de fevereiro de 2021

Conceitos e Linguagens de Programação

Vinícius do Amaral Brunheroto

**1)**

Vocabulário = { <horário> , <horas> , <minutos> , <turno> ,

<dígito> , h , min , : , am , pm , 0 , 1 , ... , 9 }

Terminais = { h , min , : , am , pm , 0 , 1 , ... , 9 }

Símbolo inicial = <horário>

Produções:

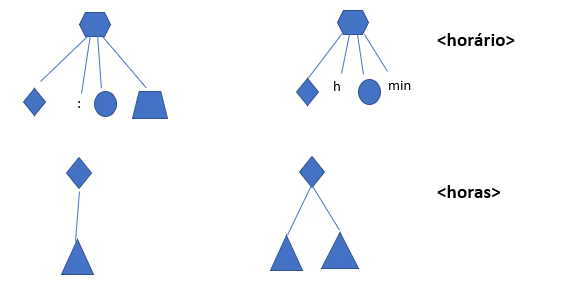
<horário> → <horas>:<minutos><turno> | <horas>h<minutos>min

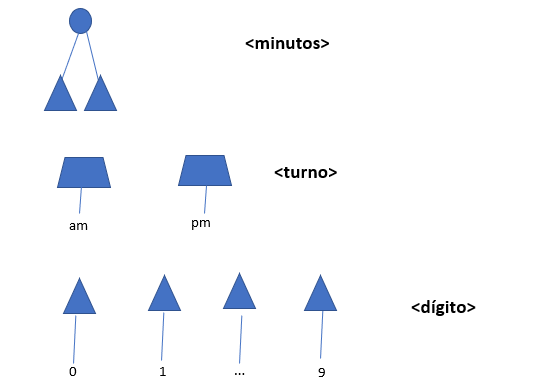
<horas> → <dígito> | <dígito><dígito>

<minutos> → <dígito><dígito>

<turno> → am | pm

<dígito> → 0 | 1 | ... | 9





**8h30min**

min

h

8

0

3

**29:30am**

:

am

2

9

3

0

**2)**

**a)**

É uma pergunta mal formulada, porque interpretador é um ambiente de execução enquanto compilador não, ou seja, foram feitos para finalidades diferentes.

O compilador converte instruções de um programa recebido em instruções de outra linguagem. Pensando nos compiladores convencionais não faz sentido em falar de desempenho e portabilidade, pois ele não realiza ações descritas do programa, mas sim falar dos programas que ele gera nos ambientes de execução destinados.

Enquanto isso, o interpretador convencional pode ser aplicativo e ambiente de execução ao mesmo tempo, pode ser considerado um programa para que outros programas possam rodar nele.

Entretanto, existem muitos interpretadores que são implementados com a técnica Just-in-time Compiler(JIT), então ele recebe o programa e compila internamente para linguagem de máquina. Como o processo de compilação é demorado, a compilação JIT compila apenas a parte mais usada do programa.

Então para esses tipos de interpretadores, implementados com a técnica compilação JIT, os programas compilados passam a ter melhor desempenho do que programas em linguagem de máquina gerados por compiladores convencionais(clássicos), porque os interpretadores são embutidos com informações de execução do programa(por conta de sua finalidade principal) e isso pode ajudar em possíveis otimizações.

**b)**

O processo de compilação em si se inicia em um código fonte do qual o compilador acessa.

Na primeira fase conhecida como ANÁLISE, o compilador processa análise léxica transformando unidades em lexemas (tokens) ,sintática que analisa a estrutura sintática das “frases”, usando regras sintáticas e semântica que pode usar de suas etapas operacional e denotacional sobre o código fonte.

A segunda fase é conhecida como SÍNTESE, ocorre a geração de código intermediário que permite a geração de instruções para uma máquina abstrata, otimização que analisa o código no formato intermediário e tenta melhorá-lo de tal forma que venha resultar um código de máquina mais rápido em tempo de execução e geração do código que tenta analisar o código já otimizado e gerar um código objeto definitivo para uma máquina alvo.

Então, o processo é finalizado com um código objeto em linguagem de montagem.

Obviamente, esse código em linguagem de montagem passará por outras etapas até ser executado no ambiente de execução, mas essas outras etapas pertencem ao processamento do programa.

**3)**

**c)** Alocação estática onde qualquer propriedade pode ser definida de antemão a partir do código, como exemplo principal variáveis globais que não se alteram entre o início e o fim da execução do programa.

O principal objetivo é definir e usar variáveis que funcionam como constantes, ou seja, melhor chamando, dados dentro do programa, em que não é preciso ficar alterando seus valores, quando a demanda do programa não necessita de alterações nos dados declarados.

Alocação dinâmica em PILHA: as variáveis são alocadas em uma região da memória conhecida como pilha, registro de ativação(RA) ou stack frame, em que cada posição corresponde à invocação individual de uma função, com dados referentes a ela(parâmetros e funções).

A variável local é criada após a ativação da chamada da função e ela deixa de ser necessária antes do fim da chamada, ou seja, o tempo de vida dessas variáveis locais é menor do que a chamada da função.

É usada quando o programa conta com propriedades que só podem ser definidas durante a execução dele, como por exemplo principal, as recursões em que não é possível prever quantas chamadas recursivas de uma função coexistirão na memória de um programa.

Outros exemplos: variáveis locais de subrotinas, parâmetros das funções..

Alocação dinâmica em HEAP: as variáveis são alocadas em uma região da memória reservada para criação e destruição aleatória de dados, conhecida como HEAP.

É usada quando existem variáveis criadas durante a execução do programa cujo tempo de vida não se pode restringir à duração de uma subrotina.

Exemplos: nós de uma lista em C(Malloc), objetos em Java...

**d)**

O problema da fuga de memória em linguagens sem gerenciamento automático de memória é conhecido como LEAKAGE.

Ele ocorre quando ocorrem manipulações no heap, ou seja, se inserem elementos no heap através de malloc, porém não ocorrem desalocações dos mesmos durante a execução do programa, geralmente quando não há utilização do free e do return, isso acaba acarretando em áreas que ficam alocadas para sempre no heap.

**e)**

O maior problema que ocorre em algoritmos com coleta de lixo é conhecido como “Pare o Mundo!” ou seja, parar a execução do algoritmo diversas vezes para poder fazer a coleta de lixo no heap, gerando queda de desempenho.

O algoritmo que mais sofre com isso é o Mark-and-sweep, foram feitos outros na tentativa de amenizar a queda no desempenho, como o incremental, geracional e o de paralelização.

Também foram feitas alternativas para contornar esse problema: linguagens com implemento de contagem de referência(contar quantos ponteiros visam o dado) com o smart pointer, como exemplo: C++, Rust.

**4)**

**f)**

Propriedades estáticas são qualquer propriedade que pode ser definida de antemão a partir do código, por exemplo: variáveis que podem ser conhecidas e definidas durante o tempo de compilação antes do programa ser executado.

Propriedades dinâmicas são qualquer propriedade que só pode ser definida durante a execução do programa,por exemplo variáveis que só podem ser definidas, “existir” durante o tempo de execução.

**5)**

**g)**

Uma variável é um elemento na memória do computador destinado a armazenar, reter ou representar um dado(um valor) que pode ser alterado durante a execução do algoritmo.

Para funcionar corretamente, a variável precisa de:

**int exemplo 5;**

-NOME(um identificador que a representa):

var exemplo; //Variável com nome exemplo.

-ENDEREÇO:

A posição em que a variável está na memória.

-VALOR:

Qual valor é armazenado pela variável(pode ser de diferentes tipos).

-TIPO:

Qual tipo essa variável é, diz que tipo de valores serão ocupados por essa variável, em C: int,float,char,double.

Exemplo válido para representar endereço,valor,tipo:

endereço valor

Exemplo int (4 bytes)

|  |  |
| --- | --- |
| 0X0100 | 0X00 |
| 0X0101 | 0X00 |
| 0X0102 | 0X00 |
| 0X0103 | 0X05 |
| 0X0104 | 0X00 |
| 0X0105 | 0X00 |
| 0X0106 | 0X00 |
| 0X0107 | 0X00 |

-TEMPO DE VIDA:

O tempo durante o qual uma variável retém seu valor é conhecido como tempo de vida. O valor de uma variável pode ser alterado durante seu tempo de vida, mas mantém algum valor. Quando uma variável perde o escopo, ela não tem mais um valor.

Por exemplo: uma variável local dentro da subrotina terá o tempo menor que a da subrotina.

O tempo começará com a chamada da subrotina ao qual está inserido e terminará com o fim da subrotina.

-ESCOPO:

Abrangência, alcance de uma variável, é uma região do código em que o vínculo da variável continua ativo.

Por exemplo: uma variável global terá um escopo global, vale para todo o programa.

O escopo de uma variável local é a própria função onde ela foi definida.

**h)**

Uma linguagem com escopamento dinâmico terá um ambiente de referenciamento que irá enxergar variáveis a partir de funções inseridas no ambiente inicial e das funções apontadas pelo pai dinâmico, que é o ponteiro para as stack frames anteriores.

Já uma linguagem com escopamento estático terá um ambiente de referenciamento que irá enxergar variáveis de funções inseridas no ambiente inicial e das funções apontadas pelo pai estático, que é o ponteiro para onde a função atual está inserida.

Então, é importante lembrar que no escopamento dinâmico, o escopo depende do histórico de cada execução e pode ser desejável em apenas alguns poucos casos, enquanto no escopamento estático, o escopo não mais depende do histórico de execução, ele é sempre o mesmo.

**i)ESCOPAMENTO ESTÁTICO**

Ambientes de referenciamento:

1)

X SUB1

Y SUB1

2)

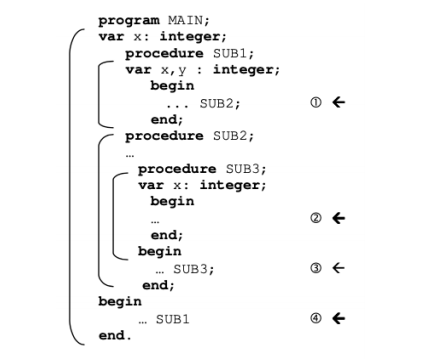
X SUB3

3)

X MAIN

4)

X MAIN



X Y

Váriável visível e onde foi definida:

3’Xs:

-Main (definida dentro do global ou main)

-SUB1(definido dentro da procedure SUB1)

-SUB3(definido dentro da procedure SUB3)

1Y:

-SUB1

(única subrotina que contém uma variável y)

**j) ESCOPAMENTO DINÂMICO**

Ambientes de referenciamento:

1)

X SUB1

Y SUB1

2)

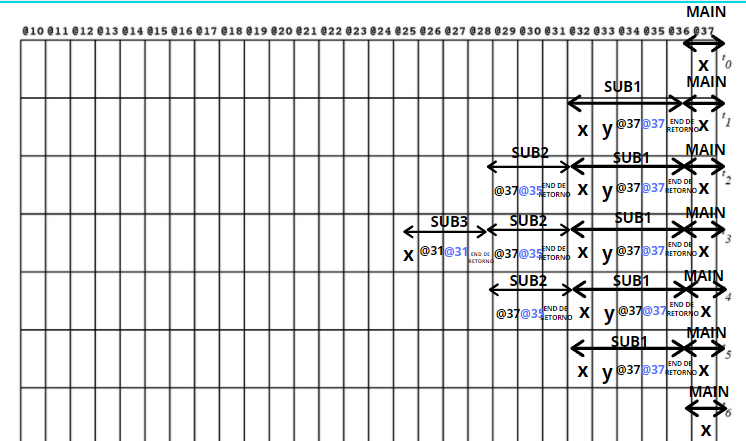
X SUB3

Y SUB1

3)

X SUB1

Y SUB1

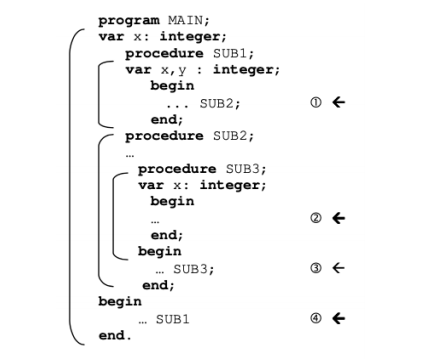
4) X GLOBAL

**4)**

**3)**

**2)**

**1)**



Váriável visível e onde foi definida:

X:

-Main (definida dentro do global ou main)

-SUB1(definido dentro da procedure SUB1)

-SUB3(definido dentro da procedure SUB3)

Y:

-SUB1

(única subrotina que contém uma variável y)