Aula 5 – Amarração e Subprogramas

5.1 - Semântica

Enquanto a sintaxe define se os programas estão corretamente escritos em uma linguagem, a semântica define o significado dos programas sintaticamente corretos. Por exemplo, a semântica do C ajuda a determinar que a declaração

int vet[10];

o compilador reserva dez espaços para elementos inteiros, para a variável chamada vet. Os elementos do vetor podem ser referenciados por um índice i, sendo que i>=0 e i<=9, o primeiro espaço reservado será vet[0], na Linguagem C, em Pascal, por exemplo, o índice inicia em 1 e termina em tamanho.

Outro exemplo, a semântica do C define que a instrução

if(a>b) max=a; **else** max=b;

significa que a expressão "a>b" deve ser avaliada e, dependendo do seu valor, um dos dois comandos de atribuição é executado.

Assim, é possível observar que as regras sintáticas mostram como formar o comando e as regras semânticas mostram qual é o efeito do comando.

Alguns conceitos semânticos básicos nas LP são:

- Variáveis: uma variável, que é referenciada por um nome, corresponde a uma região de memória usada para guardar valores que são manipulados pelo programa.

Questões semânticas associadas com variáveis envolvem sua declaração. Algumas propriedades semânticas da declaração são:

- Escopo: em qual(is) parte(s) do programa a variável pode ser acessada
- Tipo: que tipos de dados podem ser armazenados na variável e quais operações podem ser feitas com ela
- Tempo de vida da variável: quando a variável foi criada, isto é, se há uma área de memória reservada, ou alocada para a variável.

- Valores e Referências: envolve a definição de que valor está associado com a variável, se é um valor que denota a localização da memória ou o conteúdo da localização da memória.

- Expressões: possuem algumas regras semânticas para serem escritas envolvendo, principalmente, os tipos de expressões permitidas, de acordo com a Gramática da Linguagem.

5.2 – Abstração e Amarração

Abstração consiste no processo de identificar apenas as qualidades ou propriedades relevantes do fenômeno que se quer modelar.

Usando o modelo abstrato, pode-se concentrar unicamente nas qualidades ou propriedades relevantes e ignorar as irrelevantes. Porém, a determinação do que é relevante depende da finalidade para a qual a abstração está sendo projetada.

A abstração permeia toda a programação. Em particular, apresenta uma dupla relação com as LP. Por um lado, as LP são as ferramentas com as quais os programadores podem implementar os modelos abstratos. Por outro lado, as próprias LP são abstrações do processo subjacente, onde o modelo é implementado.

Na programação, a abstração sugere a distinção que deve ser feita entre "o que" o programa faz e "como" ele é implementado. Por exemplo, quando um procedimento é chamado, pode-se concentrar somente no que ele faz; apenas quando se esta escrevendo o procedimento é que deve-se concentrar em como implementá-lo.

Assim, a abstração pode ser definida com uma entidade que engloba o processamento. Por exemplo, uma abstração de função contém uma expressão a ser avaliada, que quando chamada produzirá um valor resultante; e uma abstração de procedimento contém um comando a ser executado, que quando chamado irá atualizar as variáveis. Em outras palavras, o princípio da abstração diz que é possível construir abstrações sobre qualquer classe sintática.

É importante ressaltar que os programas trabalham com entidades, tais como variáveis, rotinas e comandos. As entidades dos programas, por sua vez, possuem certas propriedades chamadas atributos. Por exemplo, uma variável tem um nome, um tipo e um valor; e uma rotina tem um nome, parâmetros formais de um determinado tipo e certas convenções de passagem de parâmetros. Os valores dos atributos devem ser definidos antes que eles possam ser usados.

A definição do valor de um atributo é conhecida como **amarração**. Para cada entidade, a informação do atributo está contida em um repositório chamado descritor, como demonstra a figura abaixo.

A amarração consiste em um conceito central na definição da semântica da LP. Linguagens de Programação diferem no número de entidades com as quais elas podem lidas, no número de atributos a serem amarrados às entidades, no tempo no qual as amarrações ocorrem, e na estabilidade da amarração, isto é, quando a determinação de uma amarração é fixa ou pode ser alterada.

Alguns atributos podem ser amarrados no tempo de definição da linguagem, outros no tempo de implementação da linguagem, outros no tempo de tradução do programa (ou tempo de compilação), e outros ainda no tempo de execução do programa. A seguir é apresentada uma lista com exemplos de amarrações:

Amarração	Descrição	Linguagens
No tempo de definição da Linguagem	O tipo inteiro é amarrado no tempo de definição da linguagem ao conjunto de operações algébricas que produzem e manipulam inteiros.	Fortran, Ada, C e C++
No tempo de implementação da Linguagem	Um conjunto de valores é amarrado ao tipo inteiro no tempo de implementação da linguagem. Em outras palavras, a definição da linguagem específica que o tipo deve ser suportado e a implementação da linguagem amarra-o à representação da memória, que por sua vez determina o conjunto de valores que estão contidos no tipo.	Fortran, Ada, C e C++
No tempo de compilação	Fornece uma pré-definição do tipo inteiro, mas permite que o programador redefina-a. Assim, o tipo inteiro que é amarrado à representação em tempo de implementação, mas a amarração pode ser alterada no tempo de compilação.	Pascal
No tempo de execução	As variáveis são amarradas a um valor em tempo de execução, e a amarração pode ser alterada repetidamente durante a execução.	Maioria das LP

Nos três primeiros exemplos da tabela, a amarração é estabelecida antes de tempo de execução e não pode ser alterada depois. Este tipo de amarração é geralmente chamada de **amarração estática**. O termo estática denota tanto o tempo de amarração (que ocorre antes do programa ser executado), como a estabilidade (a amarração não pode ser alterada).

Em oposição, uma amarração estabelecida em tempo de execução é geralmente alterável durante a execução, o que é proposto na última linha da tabela. Este tipo de amarração e geralmente chamada de **amarração dinâmica**.

Existem casos, entretanto, onde a amarração é estabelecida em tempo de execução e não pode ser alterada depois de ter sido estabelecida. Um exemplo consiste nas linguagens que fornecem variáveis constantes (somente de leitura) que são inicializadas com uma expressão a ser avaliada em tempo de execução.

Uma variável é caracterizada por um nome e quatro atributos básicos:

- escopo,
- tempo de vida,
- valor (ou conteúdo) e
- tipo.

O nome é usado para identificar e fazer referência à variável. O escopo de uma variável é o trecho de programa onde a variável é conhecida e, portanto, pode ser usada. A variável é dita visível dentro do seu escopo e invisível fora dele. A amarração de variáveis a escopos pode ser feita estática ou dinamicamente:

- Amarração Estática: escopo é definido por regras da linguagem, em geral, função da estrutura sintática do programa, sendo determinado em tempo de compilação; em outras palavras, cada referência a uma variável é estaticamente amarrada a uma declaração (implícita ou explícita) desta variável;

- Amarração Dinâmica: escopo é definido em tempo de execução; as declarações são válidas até que surja um novo contexto para o mesmo nome da variável; neste caso a desvantagem é que variáveis com

o mesmo nome podem ter tipos distintos durante a execução do programa, o que dificulta a legibilidade da programação.

O tempo de vida de uma variável significa o intervalo de tempo durante o qual uma área de memória usada para guardar o valor da variável, está amarrada a uma variável. A alocação da área de memória pode ser estática, quando realizada antes da execução do programa, ou dinâmica, quando realizada durante a execução.

O valor de uma variável é a interpretação do conteúdo armazenado na área alocada à variável. Esse conteúdo é armazenado de maneira codificada e depende do tipo da variável. Além disso, em algumas LP, o valor de uma variável pode ser uma referência (ponteiro) para um objeto. Normalmente, a amarração entre uma variável e o valor armazenado na área de memória correspondente é dinâmica, sendo feita em tempo de execução por um comando de atribuição. Algumas linguagens, entretanto, permitem o "congelamento" da amarração entre uma variável e seu valor, quando a amarração é estabelecida. A entidade resultante é uma constante definida pelo programador.

O tipo de uma variável é caracterizado pela especificação de uma classe de valores que podem ser associados a uma variável, juntamente com as operações que podem ser usadas para criar, acessar e modificar tais valores.

As LP apresentam muitas diferenças nos atributos que podem ser usados em programas, nos momentos em que ocorre a associação e na estabilidade da amarração, isto é, se a amarração é permanente ou temporária.

A associação, denominada de amarração ou acoplamento, pode ser feita em diferentes momentos: na própria definição da linguagem, quando alguns tipos de associação são definidos; no momento da compilação, quando o compilador se encarrega de fazer a conversão de um atributo definido no programa em um código objeto; no momento da ligação ou da carga, quando o programa objeto é transformado em um programa executável ou no momento da execução.

Alguns atributos claramente são associados a momentos bem definidos, como por exemplo:

- Associação de variável X tipo: momento da compilação
- Associação de variável X posição de memória: momento da carga (momento da execução)
- Associação de variável X valor: momento da execução

Sob o ponto de vista de execução de um programa, somente interessam as amarrações feitas antes de iniciar a execução, conhecidas como amarrações precoces e aquelas que somente são realizadas durante o processo de execução, conhecidas como amarrações tardias.

Dependendo do momento em que é realizada a amarração, esta pode ser estática ou dinâmica. A amarração é estática quando a associação é invariante durante todo o processo de execução do programa. Ao contrário, quando a associação pode mudar durante o processo de execução, a amarração é dita dinâmica.

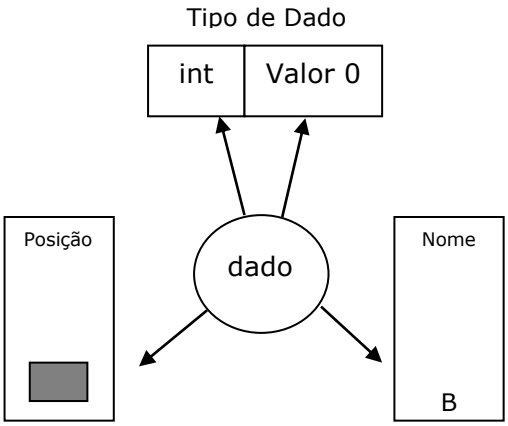
Atributos	Entidades

`program exemplo1;`
`const`
`  Max = 20;`
`type`
`  vetor: array [1..Max] of real;`
`var`
`  valor, i: integer;`
`  vet1, vet2: vetor;`
 	
`Begin`
`  for i := 1 to Max do`
`   begin`
`  vet2[i] := i;`
`  readln (vet1[i]);`
`   end;`
`classifica (vet1);`
`End.`
	Atributos de um programa:
	- Variáveis: nome, tipo, valor, posição (endereço)
	- Subprogramas: nome, parâmetros, modos de passagem de parâmetros
	- Comandos/Expressões: ações
	- Operadores: símbolos e operações
	
	Amarração: associação entre atributos definidos pelo programador e entidades de execução.

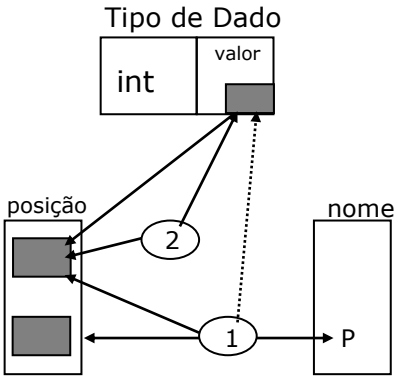
Momento da Amarração	Estabilidade da Amarração
- Definição da LP (tipos de dados e operações) - Implementação da LP (domínio de valores) - Compilação do programa (nomes e tipos) - Ligação e carga do programa (posição de memória) - Execução do programa (valores de variáveis)	Estática: - LP de tipagem forte - Amarração de nome e tipo - Operadores - Chamadas de procedimentos Dinâmica: - Tipos variantes - Amarração de valor e posição - Mensagens (polimorfismo)

Atributos de Variáveis	Amarração de Posição
Variáveis podem ser caracterizadas pelos atributos de: - Duração (tempo de vida) - Nome (identificador) - Posição (endereço) - Valor (conteúdo) - Tipo (domínio e operações) - Alcance e visibilidade (acesso)	- Uma variável pode ter diferentes endereços durante a execução - Um nome de variável pode ter diferentes endereços em diferentes lugares de um programa - Aliases: dois nomes de variáveis podem ser usados para acessar a mesma posição

Amarração: exemplos de associação



Exemplo para a declaração
int B = 0;



Exemplo de
int *P;

5.3 – Exercícios

1. Considerando o seguinte trecho de código em C, liste os tipos de amarração ocorridos para cada linha de código nos tempos de compilação e execução:

```
int a;
scanf ("%i",&a);
int c = a * 5;
c = c * a;
a = c;
```

2. Liste todos os tipos de amarração que ocorrem no seguinte trecho de código em C:

```
#define TAM 30
float x = 3.2;
char str[TAM];
gets(str);
x = x * TAM;
```

5.4 Abstração Procedimental

Uma sub-rotina/Função/Procedimento é um escopo nomeado parametrizado.

O escopo contém declarações de variáveis locais e instruções. Nomeado é ser referenciado na chamada e parametrizado são os argumentos que podem ser passados para o escopo.

Os subprogramas têm como características gerais:

- Possuem um único ponto de entrada (através de seus argumentos e parâmetros);

- A unidade chamadora é suspensa durante a execução dos subprogramas (no momento da chamada do subprograma o compilador passa a executar apenas as instruções contidas no subprograma);

- O controle retorna ao chamador quando o subprograma é encerrado (ao término das instruções do subprograma, é retornada a chamadora e executadas as instruções nela contidas).

Podem retornar um valor, chama-se Função. É o caso da linguagem C, que sempre devolve um valor para a sua chamadora (no caso, outra função – que pode ser a main ou outra). Podem também serem chamadas através de expressões.

Pode não retornar um valor, é chamada de "procedure" (Ada e Pascal), "subroutine" (Fortran) e "function/methods" (C++, Java). Tem como característica, além de não retornar um valor, também não podem ser utilizadas em expressões.

Uma Função é composta de um nome, de argumentos e parâmetros. O nome é uma variável atribuída a identificação da função, por esse nome é feito o chamamento da função dentro do programa. Já os argumentos são as expressões que aparecem em uma chamada. Os parâmetros são os identificadores que aparecem na declaração. Observe o exemplo em Linguagem C, abaixo:

```
#include<stdio.h>
#include<conio.h>
#define TAM 10

int Funcao (int w){
 int i;
 i = 2 * w;
 return (i);
}


int main(){
  int i,x;
  for(i=0;i<TAM;i++){
    x = Funcao;
    printf("\nx = %i\n",x);
   }
getch();
}
```

Subprogramas são trechos de programa que realizam uma tarefa específica. Podem ser chamados pelo nome a partir do programa principal ou de trechos de outros subprogramas, até mesmo ele próprio (chamada recursiva).

Pode ser de dois tipos:

- Funções: Retornam um valor em seu nome
- Procedimentos: Não retornam valor

A Linguagem C possui apenas o subprograma do tipo Função.

Declaração:

<tipo_do_valor_retornado> <nome_da_função> (<parâmetros_formais>)

Chamada:

<nome_da_função> (<parâmetros_reais>);

Parâmetros Formais: são aqueles passados na declaração da função. É onde informamos quais as variáveis que a função irá receber quando chamada e quais os seus tipos (são informados com uma

declaração de variáveis). Esses parâmetros são considerados como variáveis locais a função. Se a função não precisa receber nenhum parâmetros colocamos apenas parênteses.

Parâmetros Reais: são aqueles passados na chamada da função. É quando informamos quais os valores que os parâmetros terão dentro da função. Se a função não espera receber nenhum parâmetro, na sua chamada colocamos o abre e fecha parênteses vazio.

Exemplo:

```
#include<stdio.h>
#include<stdlib.h>
#include<conio.h>
#define TAM 10

int CriaVetor(int vet[]){
 int i;
 for(i=0;i<TAM;i++)
   vet[i]=rand()%101;
 }

int ImprimeVetor(int vet[]){
 int i;
 for(i=0;i<TAM;i++){
  printf("%i\t",vet[i]);
  }
 }

int menu(){
 int op,vetor[TAM];
 do{
  printf("\nDigite a opcao desejada:\n");
  printf("1 - Criar vetor\n2 - Imprimir vetor\n3 - Sair\n");
  scanf("%i",&op);
  switch(op){
   case 1:{CriaVetor(vetor); break;}
   case 2:{
     printf("\nImpressao do Vetor\n\n");
     ImprimeVetor(vetor);
     break;}
   case 3:{printf("\nMenu Encerrado!\n");break;}
   default: printf("\nOpcao Invalida!\n");
   }
  }while(op!=3);
 }
```