VII. Ambientes em Tempo de Execução

Questões sobre a linguagem fonte

Procedimentos

- Uma definição de procedimento associa um identificador a um enunciado(corpo).

Ex: procedure Incrementa (var x: inteiro);
begin x:= x+1; end;

- Uma chamada de procedimento invoca a execução de seu corpo de instruções (ativação).
- Parâmetros são identificadores que realizam a comunicação de valores entre o procedimento que chama e o que é chamado
- Um procedimento é recursivo se uma ativação puder começar antes de uma ativação do mesmo procedimento ter terminado.

Ex:

```
Proc1(x : int)
|  X :=x-1
|  Se x>0 Proc1(x);

Programa: Proc1(2);
```

Pilhas de ativação

O fluxo de controle pode também ser representado por uma pilha, onde o início da ativação corresponde a empilhar o nodo e o fim da ativação corresponde a desempilhar o nodo.

Amarração de nome

Uma declaração “amarra” um nome a uma área de memória até que essa declaração passe a não ser mais válida. O tempo decorrido entre a amarração e seu fim é denominado tempo de vida.

Escopo de uma declaração

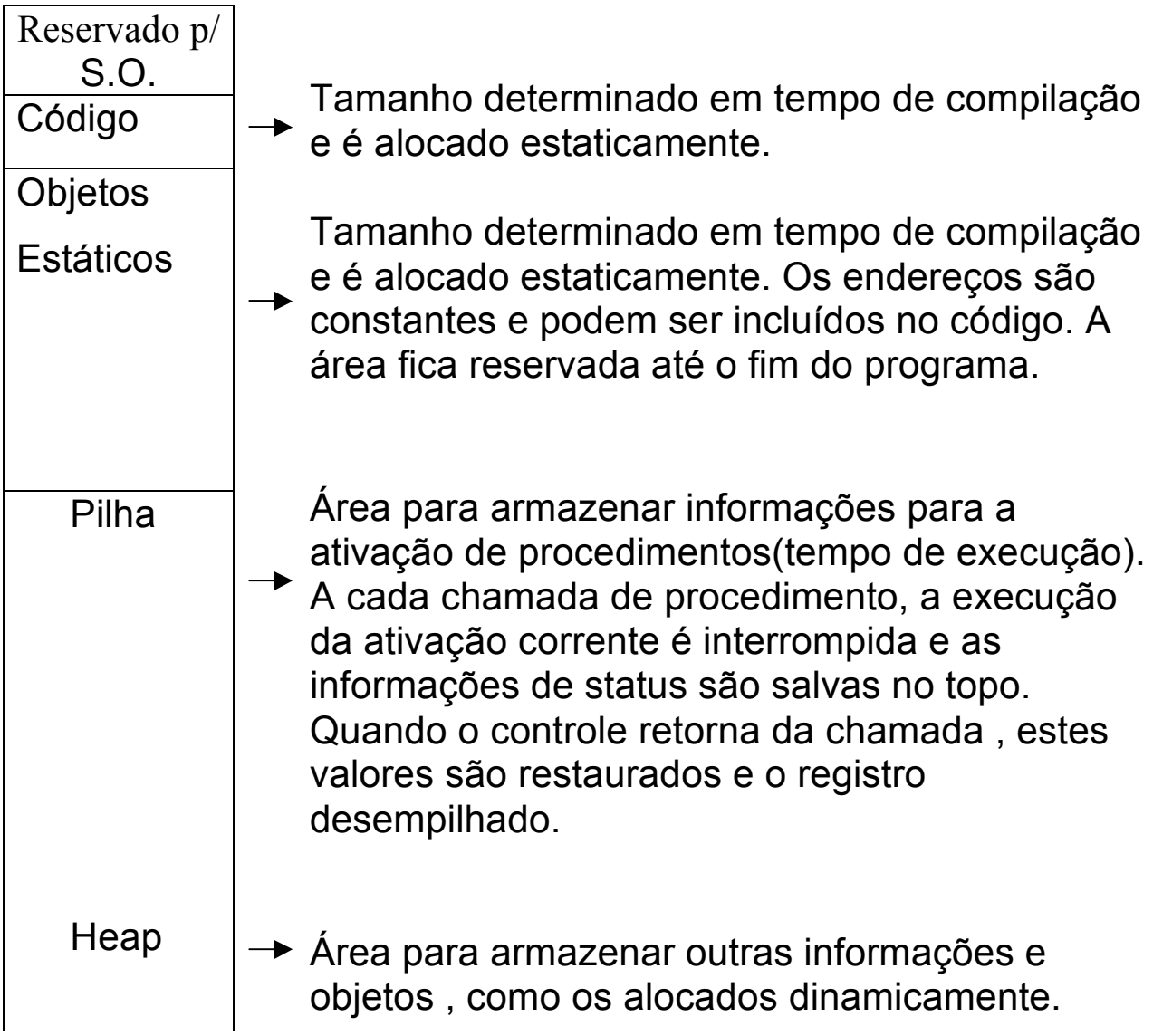
É a porção do programa à qual se aplica uma declaração. Uma ocorrência de um nome é dita local se estiver no escopo de uma declaração do procedimento, caso contrário é dita não-local.

Noção estática (P.F.)	**Contraparte dinâmica (exec.)**
Definição de um procedimento	Ativação de um procedimento
Declaração de um nome	Amarração de um nome a uma área de memória
Escopo de uma declaração	Tempo de vida de uma amarração

C	**Pascal**	**Significado**
int V	v: integer	Declara variável inteira e reserva espaço
int * P	P: ^integer	Declara apontador para inteiro e reserva espaço
p = &v	P:= @v	Atribui a ‘p’ o endereço reservado a ‘v’
*p = 10	P^ :=10	Armazena o valor 10 na posição de memória apontada por ‘p’

Organização de Memória

Exemplo de Organização (Alocação de R.A. em pilha)



Registro de Ativação

Quando um procedimento é chamado, um conjunto de informações deve ser salvo em memória, na área estática , pilha , ou heap . O registro tem os seguintes campos (normalmente):

Valor Retornado	→	No caso de funções, guardar o valor de retorno ao procedimento chamador.
Parâmetros reais	→	Contém os valores dos parâmetros no procedimento chamado.
Elo de controle	→	Apontador para o R.A. do procedimento chamador
Elo de acesso	→	Apontador para dados locais de outro reg. ativação.
Estado salvo	→	Contém os valores dos registradores que precisam ser restaurados ao fim do tempo de vida (ex: PC).
Dados locais	→	Objetos locais ao procedimento.
Temporários	→	Valores temporários resultantes da avaliação de expressões no procedimento.

Considerações sobre os dados locais

Endereços de memória -> vamos considerar que a memória é organizada em byte contíguos.

Tamanho dos objetos -> quantidade de bytes necessários para armazenamento. Depende da máquina e do tipo de dado.

Disposição -> reservadas à medida em que suas declarações são examinadas pelo compilador.

Estratégias para alocação de memória

Alocação de memória estática

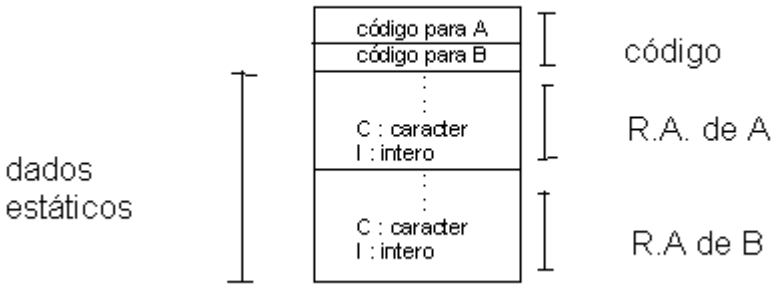
Objetos alocados estaticamente têm seus tamanhos e endereços determinados em tempo de compilação. Os endereços são obtidos através de um valor de deslocamento, a partir de uma extremidade do registro de ativação. Neste caso, os registros de ativação ficam alocados na área de dados estáticos.

O Fortran 77 é um exemplo de linguagem que usa esta estratégia. Todas as ativações de um procedimento utilizam um único registro de ativação.

Ex:

```
Program A
| character C
| integer I
End

Character Function B|
| character C
| integer I
End
```



Alocação de Memória de Pilha

Os registros de ativação são empilhados e desempilhados à medida em que as ativações são iniciadas e terminadas, respectivamente. Cada ativação de um procedimento X reserva uma área na pilha, para seu registro de ativação. Essa área é liberada ao fim da ativação, com a perda dos valores armazenados.

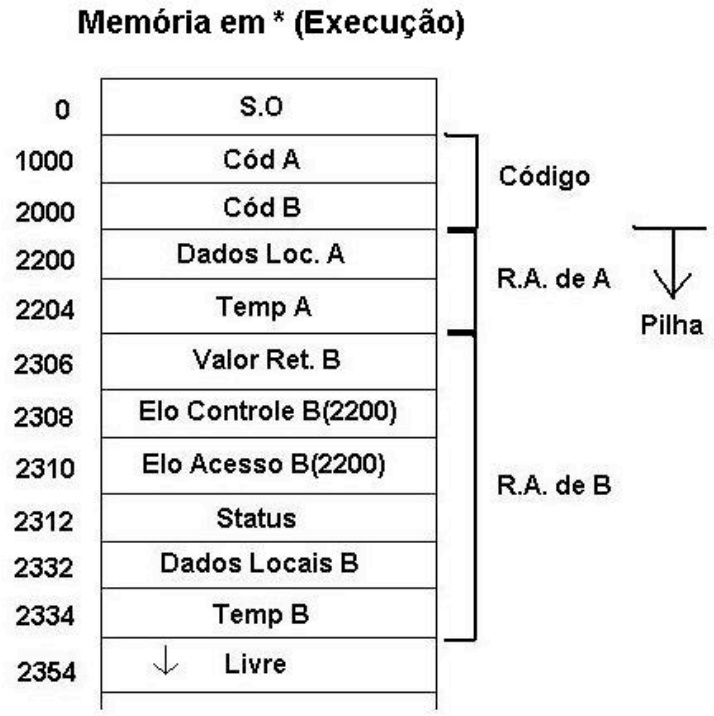
Ex: Máquina com 64K endereços de bytes. Sistema Operacional utiliza 1000 bytes.

```
Prog A
Var i, j: integer
        .
        .
        .
        i := 1;
        j := B;
End;
```

Tam Código A = 1000,
temp = 102 bytes

```
Func B : Integer
Var i: integer
                .
                *
        return(i);
End;
```

Tam Código B = 200,
temp = 20 bytes
Estado salvo = 20 bytes



Referências Ocas

Referir-se a uma área de memória que já foi liberada (retorno da ativação) é um erro de lógica.

Ex:
```
main()
{ int *p;
  p = A();  ← referência Oca
}

int *A()
{ int i = 23;
  return(&i);
}
```

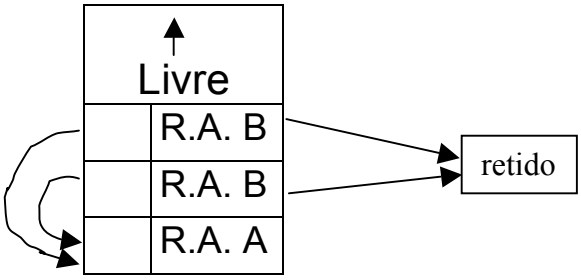
Alocação de Memória de Heap

Aloca blocos de memória contígua, à medida do necessitado, para registros de ativação ou outros objetos (Alocação Dinâmica). Estes blocos podem ser liberados em qualquer ordem. Após o fim da ativação, os dados da área reservada não são perdidos. A liberação desta área deve ser requisitada por instruções do programa.

Ex:

```
Proc A;
|   B;
|   B;
|    *
```

Memória de Heap em *



"C"	Pascal	Descrição
int *p	p:^integer	declara ponteiro de int.
p = malloc(sizeof(int))	new(p)	reserva área de memória, de conteúdo indefinido. O endereço da área fica em p.
*p = 10	p^:= 10	armazena 10 na área apontada por p.
free(p)	Dispose(p)	libera área de memória Os conteúdos da área e de p ficam indefinidos

Acesso aos nomes não locais

- Um bloco é uma parte do código que contém suas próprias declarações de dados locais.

Ex.:
```
{
        int i
. }
```

- Um bloco pode estar aninhado dentro de outro bloco.

```
{  {  }  }
```

- Um bloco pode ser considerado um procedimento sem parâmetros, sendo empilhado e desempilhado após a ativação.

Regra do Aninhamento mais interno

1) O escopo de uma declaração inclui o bloco em que foi declarada.
2) Se um nome **X** não foi declarado no bloco **B** onde aparece, esta ocorrência pertence ao escopo do bloco **A** mais internamente aninhado a **B** que possui declaração de **X**.
3) O escopo de um nome **X** não abrange um bloco aninhado que contenha outra declaração para **X**. Isto é chamado “buraco” no escopo de uma declaração.

```
main ()
{
| int a = 0,  (1)
|     b = 0;  (2)
|     {
|     |     int b = 1;  (3)
B0 |      {
|     |      |
|     B1  B2   int a = 2;  (4)
|     |      |      printf("%d,%d\n",a,b);
|     |      }
|     |
|     |      {
|     |      |
|     |     B3    int b = 3;  (5)
|     |      |      printf("%d,%d\n",a,b);
|     |      }
|     |      printf("%d,%d\n",a,b);
|     }
|     printf("%d,%d\n",a,b);
}
```

Declaração	Escopo
1	B0 - B2
2	B0 - B1
3	B1 - B3
4	B2
5	B3

Saída: 2,1
0,3
0,1
0,0

Regras de Escopo

Indicam a forma de tratamento das referencias a nomes não-locais. Se dividem em regra de escopo léxico (ou estático) e regra de escopo dinâmico.

Na R.E Léxico, determina-se o escopo das declarações a partir do exame do código estático. Na R.E dinâmico é preciso considerar de onde foi chamada a ativação do procedimento (tempo de execução) para se saber a qual escopo os nomes não-locais fazem referencia.

EX:

```
Programa A
      var i: integer;

            procedure B:
            begin  writeln(i); end;

            procedure C:
            var i: integer;
            begin i:=1; B; end;

      begin
          i := 0; B; C;
      end
```

Saída: se RE estático:0,0
se RE dinâmico:0,1

Regras de escopo para “C” e Pascal

- Ambos utilizam R.E léxico.
- Pascal pode ter procedimentos aninhados, “C” não.
- Um nome não-local em “C” deve ser global(estático).
- Pascal não tem variáveis estáticas (globais).

Transmissão de parâmetros

Definições:

- Parâmetro real: argumento transmitido de um procedimento chamador a um procedimento chamado.
- Parâmetro formal: identificador que aparece na definição do procedimento chamado e é tratado como nome local.

Transmissão de Parâmetro por Valor

O parâmetro real da chamada é avaliado e seu conteúdo passado para o procedimento chamado. O parâmetro formal é considerado um nome local e não afeta diretamente os valores no registro de ativação do procedimento chamador.

Ex: Program A;

```
var a,b,c : integer;
procedure B (a,b : int);
|    a := 2;
|    c := a + b;

a := 0; b := 1; c:= 2;
B (a,b);
Writeln (a,b,c);
```

saída: 0, 1, 3

Ex:

```
void A (*int a; int b)
{ *a := b }

main ()
{   int a=1, b=2, c=3;
     A(&c, a);
     printf( %d, %d,%d\n", a, b, c);
}
```

saída: 1, 2, 1

Transmissão de Parâmetro por Referência

O endereço do parâmetro real é avaliado e passado para o procedimento chamado, ficando inalterado até o fim da ativação. O parâmetro formal serve como referência indireta ao parâmetro real, logo, o conteúdo do parâmetro real pode ser alterado.

Ex:

```
Prog A;
   var      b: integer;
            c: ^integer;
   Proc X ( var y: integer; var z: ^integer);
   |  y := 1;
   |  new (z); z^:=2;
begin
      b := 0;
      c := &b;                  saída
      writeln (b,c^);             0, 0
      X (b,c);
      Writeln (b,^c);             1, 2
end.
```