Linguagens de Programação
Conceitos e Técnicas

Expressões e Comandos

Expressões

- Uma expressão é uma frase do programa que necessita ser avaliada e produz como resultado um valor
- Elementos
  - Operadores
  - Operandos
  - Resultado

Expressões

- Podem ser
  - Simples
  - Compostas
- Notação
  - Prefixada: operador antes dos operandos

    a = !b;
  - Infixada: operador entre operandos

    a = a + b;
  - Posfixada: operador após operandos

    a = b++;

Operadores

- Aridade
  - unários, binários, ternários, etc
  - eneários: aridade variável
    - número de parâmetros variável em funções C, C++, Java
  - aridade elevada
    - reduz legibilidade e redigibilidade
- Origem
  - Pré-existentes
    - normalmente unários e binários
  - Definidos pelo Programador
    - normalmente funções com qualquer aridade
  - Composição de operadores – ML e APL

Tipos de Expressões

- Literais

  2,72 99 0143 'c' 0x43

- Agregação

```
int c[ ] = {1, 2, 3};
struct data d = {1, 7, 1999};
char * x = {'a','b','c', '\0'};
int b[6] = {0};
char * y = "abc";
```

  - Agregação Estática e Dinâmica

```
void f(int i) {
   int a[] = {3 + 5, 2, 16/4};
   int b[] = {3*i, 4*i, 5*i};
   int c[] = {i + 2, 3 + 4, 2*i};
}
```

Tipos de Expressões

- Expressões de agregação em ADA

```
type data is record
    dia : integer range 1..31;
    mes : integer range 1..12;
    ano : integer range 1900..2100;
end record;
aniversario: data;
data_admissao: data:= (29, 9, 1989);
aniversario := (28, 1, 2001);
data_admissao  := (dia => 5, ano => 1980, mes => 2);
```

Tipos de Expressões

- Aritméticas

```
float f;
int num = 9;
f = num/6;
f = num/6.0;
```

  - + e – unários e binários
- Relacionais
  - usadas para comparar os valores de seus operandos
- Booleanas
  - Realizam as operações de negação, conjunção e disjunção da álgebra de Boole

Tipos de Expressões

- Binárias

```
void main() {
   int j = 10;
   char c = 2;
   printf("%d\n", ~0);          /* imprime -1 */
   printf("%d\n", j & c);       /* imprime  2 */
   printf("%d\n", j | c);       /* imprime 10 */
   printf("%d\n", j ^ c);       /* imprime  8 */
   printf("%d\n", j << c);      /* imprime 40 */
   printf("%d\n", j >> c);      /* imprime  2 */
}
```

Tipos de Expressões

- Condicionais
  - Em ML

```
val c = if a > b  then a - 3 else b + 5
val k = case i of
      1 => if j > 5 then j - 8 else j + 5
    | 2 => 2*j
    | 3 => 3*j
    | _ => j
```

  - Em JAVA

```
max = x > y ? x : y;
par  = z % 2 == 0 ? true : false
```

  - Algumas LPs (tal como ADA) não oferecem expressões condicionais – forçam o uso de comandos condicionais

```
if x > y then max := x; else max := y; end if;
```

Tipos de Expressões

- Chamadas de Funções
  - Operador => nome da função
  - Operandos => parâmetros
  - Resultado => retorno da função
  - Chamada Condicional de Função em ML

```
val taxa =
  (if difPgVenc > 0 then desconto else multa) (difPgVenc)
```

  - Em C

```
double (*p)(double);
p = difPgVenc < 0 ? desconto: multa;
taxa = (*p) (difPgVenc);
```

Tipos de Expressões

- Operadores das LPs denotam funções

Expressão	Representação Prefixada
$a * b$	* (a , b)
c / d	/ (c , d)
$a * b + c / d$	+ (* (a , b) , / (c , d))

Tipos de Expressões

- Algumas Assinaturas de Operadores em JAVA

Operador	Assinatura da Função
!	[boolean → boolean]
&&	[boolean x boolean → boolean]
*	[int x int → int] [float x float → float]

Tipos de Expressões

- Com Efeitos Colaterais

  x = 3.2 * ++c;

  - Podem gerar indeterminismo

    x = 2;

    y = 4;

    z = (y = 2 * x + 1) + y;

  - Funções possibilitam a ocorrência de efeitos colaterais

    fgetc(f);

  - Expressões cujo único objetivo é produzir efeitos colaterais

    delete p; ← Não seria então um comando?

Tipos de Expressões

- Referenciamento
  - Usadas para acessar o conteúdo ou retornar referência para variáveis ou constantes

```
*q = *q + 3;
const float pi = 3.1416;
int raio = 3;
float perimetro = 2*pi*raio;
p[i] = p[i + 1];
*q = *q + 3;
r.ano = r.ano + 1;
s->dia = s->dia +1;
t = &m;
```

Referenciamento direto (sem operador)

Operador new de Java/C++ também é de referenciamento (com efeito colateral)!

Tipos de Expressões

- Referenciamento:

Operador	Significado
[]	Acesso a valor ou retorno de referência de elemento de vetor
*	Acesso a valor ou retorno de referência de variável ou constante apontada por ponteiro
.	Acesso a valor ou retorno de referência de elemento de estrutura
->	Acesso a valor ou retorno de referência de elemento de estrutura apontada por ponteiro
&	Retorno de referência a qualquer tipo de variável ou constante

E

Tipos de Expressões

- Categóricas
  - Realizam operações sobre tipos de dados
  - Tamanho do Tipo

```
float * p = (float *) malloc (10 * sizeof (float));
int c [] = {1, 2, 3, 4, 5};
for (i = 0; i < sizeof c / sizeof *c; i++) c[i]++;
```

  - Conversão de Tipo

```
float f;
int num = 9, den = 5;
f = (float)num/den;
```

Tipos de Expressões

- Categóricas
  - Identificação de Tipo

```
Profissao p = new Engenheiro ( );
if (p instanceof Medico)
    System.out.println ("Registre-se no CRM");
if (p instanceof Engenheiro)
    System.out.println ("Registre-se no CREA");
```

Avaliação de Expressões Compostas

- Precedência de Operadores
  - Escolha inadequada pode afetar a redigibilidade

    /* if a > 5 and b < 10 then */

    if (a > 5) and (b < 10) then a := a + 1;
  - Ausência de precedência (SMALLTALK E APL) baixa a redigibilidade
  - Parênteses asseguram a ordem, mas reduzem redigibilidade e impedem otimizações

Avaliação de Expressões Compostas

- Associatividade de Operadores
  - Operadores de Mesma Precedência
  - Normalmente da esquerda para a direita

    x = a + b - c;

    y = a < b < c;
  - Podem existir exceções a essa regra

    x = **p;

    if (!!x) y = 3;

    a = b = c;
  - APL não tem precedência e sempre associa da direita para a esquerda

    X = Y ÷ W - Z

Avaliação de Expressões Compostas

- Associatividade de Operadores
  - Compiladores podem otimizar, mas isso pode causar problemas

    ```
    x = f() + g() + h();
    ```

- Precedência de Operandos
  - Não determinismo em expressões

    ```
    a[i] = i++;
    ```

  - JAVA resolve adotando precedência de operandos da esquerda para direita
  - Garante portabilidade, mas compromete eficiência

Avaliação de Expressões Compostas

- Curto Circuito
  - Situação Potencial

    ```
    z = (x - y) * (a + b) * (c – d);
    ```

  - Usado em Expressões Booleanas

    ```
    int[] a = new int [n];
    i = 0;
    while (i < n && a[i] != v) i++;
    ```

  - JAVA e ADA tem operadores específicos para avaliação com e sem curto circuito

Avaliação de Expressões Compostas

- Curto Circuito
  - Operadores booleanos de C e C++ usam curto circuito
  - Pode-se usar operadores binários *&* e */* pois não usam curto circuito
  - Curto circuito com efeitos colaterais reduz legibilidade

```
if (b < 2*c || a[i++] > c ) { a[i]++; }
```

Comandos

- Objetivo é atualizar variáveis ou controlar o fluxo de controle
- Característicos de LPs imperativas
- Podem ser Primitivos ou Compostos
- Bastam atribuição, seleção e desvio, porém LP fica pouco expressiva

Comandos

- Tipos de comandos [Watt, 1990]:
  - Atribuições;
  - Comandos sequenciais;
  - Comandos colaterais;
  - Comandos condicionais;
  - Comandos iterativos;
  - Chamadas de procedimento;
  - Comandos de desvio incondicional.

Tipos de Comandos

- Atribuição
  - = versus :=

    i := !i + 1

    if (a = 10) a += 3;
  - Simples

    a = b + 3 * c;
  - Múltipla

    a = b = 0;
  - Condicional

    (if a < b then a else b) := 2;

Derreferenciamento obrigatório em APL explicita o fato de i representar dois conceitos diferentes.

Tipos de Comandos

- Atribuição
  - Composta

    a += 3;

    a *= 3;

    a &= 3;
  - Unária

    ++a;

    a++;

    --a;

    a--;
  - Expressão

    while ((ch = getchar ( )) != EOF) { printf("%c", ch); }

Tipos de Comandos

- Seqüenciais
  - Blocos

```
{
    n = 1;
    n += 3;
    if (n < 5) {
        n = 10;
        m = n * 2;
    }
}
```

Tipos de Comandos

- Colaterais

```
a = 0;
a = 3, a = a + 1;

val altura = 2
and largura = 3
and comprimento = 5

and volume = altura * largura * comprimento
```

Tipos de Comandos

- Condicionais
  - Seleção de Caminho Condicionado

```
if (x < 0) { x = y + 2; x++; }
```

  - Seleção de Caminho Duplo

```
if (x < 0) { x = y + 2; x++; } else { x = y; x--; }
```

    - Problema com Marcadores

```
if ( x == 7 )
    if ( y == 11) {
        z = 13;
        w = 2;
    }
else z = 17;
```

Comandos Condicionais

- Seleção de Caminho Duplo
  - Necessidade de Uso de Marcadores

```
if ( x == 7 ) {
    if ( y == 11 ) {
        z = 13;
        w = 2;
    }
} else z = 17;
```

Comandos Condicionais

- Seleção de Caminho Duplo
  - ADA requer marcador de final de comando

```
if x > 0 then
    if y > 0 then
        z := 0;
    end if;
else
    z := 1;
end if;
```

Comandos Condicionais

- Seleção de Caminhos Múltiplos

```
switch (nota) {
    case 10:
    case 9: printf ("Muito Bom!!!");
        break;
    case 8:
    case 7: printf ("Bom!");
        break;
    case 6:
    case 5: printf ("Passou...");
        break;
    default: printf ("Estudar mais!");
}
```

Comandos Condicionais

- Caminhos Múltiplos com ifs Aninhados

```
if (rendaMes < 1000)
   iR = 0;
else if (rendaMes < 2000)
   iR = 0.15 * (2000 - rendaMes);
else
   iR = 0.275 * (rendaMes - 2000) +
        0.15 * (2000 – rendaMes);
```

  - MODULA-2, ADA e FORTRAN-90 têm elsif

Tipos de Comandos

- Iterativos
  - Número Indefinido de Repetições
    - Pré-teste

```
f = 1;
y = x;
while ( y > 0) {
    f = f * y;
    y--;
}
```

    - Pós-teste

```
f = 1;
y = 1;
do {
    f = f * y;
    y++;
} while (y <= x);
```

Comandos Iterativos

- Problema com Pré-Teste e Pós-Teste
  - Pré-teste

```
s = 0;
printf ("n: ");
scanf ("%d", &n);
while (n > 0) {
    s +=n;
    printf ("n: ");
    scanf ("%d", &n);
}
```

  - Pós-teste

```
s = 0;
do {
    printf ("n: ");
    scanf ("%d", &n);
    if (n > 0) s+=n;
} while (n > 0);
```

Comandos Iterativos

- Número Definido de Repetições
  - Em MODULA-2

```
s := 0;
FOR i := 10 * j TO 10 * (j + 1) BY j DO
   s := s + i;
END;
```

- Consenso:
  - Valores da Variável de Controle Conhecidos Antes do Primeiro Ciclo e Fixos
  - Realização do Teste Antes da Execução do Corpo

Número Definido de Repetições

- Variação no Escopo da Variável de Controle
  - ADA e JAVA restringem ao corpo
  - ADA não permite alteração da variável de controle no corpo da repetição
  - FORTRAN, PASCAL e C tratam como variável ordinária
  - C++ permite que escopo comece no comando

Número Definido de Repetições

Comando for de C

```
dif = 0;
for (i = 0; i < n; i++) {
    if (a[i] % 2 == 0) dif += a[i];
    if (a[i] % 3 == 0) dif -= a[i];
}

for (i = 10 * j, s = 0; i <= 10 * (j + 1); s += i++);

for (i = 0, s = 0; i <= n && s < 101 && a[i] > 0 ; )
    s += a[i++];

for (;;);
```

Número Definido de Repetições

- Pode não se Restringir a Tipos Primitivos Discretos

```
@dias = ("Dom", "Seg", "Ter", "Qua","Qui", "Sex", "Sab");
foreach $dia (@dias) {
    print $dia
}
```

- Maioria das LPs não Oferece
- JAVA e C++ Oferecem Iteradores Associados a Coleções
- A partir da versão 5, Java oferece o for-each.

Tipos de Comandos

- Chamada de Procedimentos
  - Objetivo é Atualizar Variáveis
- Desvios Incondicionais
  - Somente Comandos de Entrada e Saída Única Podem Ser Restritivos em Algumas Situações
  - Entrada Única e Saídas Múltiplas é Positivo
  - Entradas Múltiplas é Negativo
  - Tipos
    - Desvios Irrestritos
    - Escapes

Desvio Irrestrito

- Conhecido como Comando goto
- Pode Ser Nocivo a Boa Programação
- Algumas LPs o Eliminaram (MODULA-2)
- É Importante em Algumas Situações
- Também é Usado para Propagação de Erros em LPs sem Tratamento de Exceções
  - Visual Basic: On Error GoTo X

Desvio Irrestrito

- É necessário em alguns casos
  - Sem goto

```
achou = 0;
for (i = 0; i < n && !achou; i++)
   for (j = 0; j < n && !achou; j++)
      if ( a[i] == b[j] ) achou = 1;
if (achou) printf ("achou!!!");
else printf ("não achou!!!");
```

  - Com goto

```
for (i = 0; i < n; i++)
   for (j = 0; j < n; j++)
            if ( a[i] == b[j] )
                  goto saida;
saida:
if (i < n) printf ("achou!!!");
else printf ("não achou!!!");
```

Escapes

- Desvios Incondicionais Estruturados
- Não Podem Criar ou Entrar em Repetições
  - break

```
s = 0;
for(;;) {
    printf ("n: ");
    scanf ("%d", &n);
    if (n <= 0) break;
    s+=n;
}
```

  - continue

```
i = 0;
s = 0;
while(i < 10) {
    printf ("n: ");
    scanf ("%d", &n);
    if (n < 0) continue;
    s+=n;
    i++;
}
```

Escapes

- Associação com Iterações Rotuladas pode ser útil
- ADA e JAVA oferecem

```
saida:
for (i = 0; i < n; i++) {
   for (j = 0; j < n; j++) {
      if ( a[i] < b[j] ) continue saida;
      if ( a[i] == b[j] ) break saida;
   }
}
if (i < n) printf ("achou!!!");
else printf ("não achou!!!");
```

Escapes

- Podem Interromper a Execução de Subprogramas e Programas

```
void trata (int erro) {
    if (erro == 0) {
        printf ("nada a tratar!!!");
        return;
    }
    if (erro < 0) {
        printf ("erro grave – nada a fazer!!!");
        exit (1);
    }
    printf("erro tratado!!!");
}
```

Escapes

- JAVA Não Tem Usado goto
  - Embora seja palavra reservada
  - Combinação tem sido suficiente
    - Escapes
    - Escapes rotulados
    - Tratamento de Exceções

Considerações

- LP Fica Empobrecida Quando Não Oferece Tipos de Expressões ou Comandos Listados
- Expressões e Comandos Adicionais Podem Não Acrescentar Nada
  - Entrada e Saída em COBOL e FORTRAN versus Chamadas de Procedimento
- Existem Interseções entre Expressões e Comandos
  - LPs Orientadas a Expressão (ALGOL-68 e ML)
  - C é Orientada a Expressão se Considerar Fluxo de Controle Retornando void