Linguagens de Programação

Rotinas, Modos e Tipos de Passagem de Parâmetros

Carlos Bazilio
bazilio@ic.uff.br
http://www.ic.uff.br/~bazilio/cursos/lp

Passagem de Parâmetros

- Os parâmetros são um canal de comunicação entre uma rotina chamadora e uma chamada
- Na rotina chamadora, o parâmetro é tecnicamente chamado de argumento ou parâmetro real
- Na rotina chamada, o parâmetro é chamado de parâmetro formal
- Tipos comuns de passagem de parâmetros:
  - Valor, Referência, Valor/Resultado, Nome

Passagem de Parâmetros Valor

- Neste modo, na transferência de execução da rotina chamadora para a chamada é realizada uma cópia do valor do parâmetro real para o formal
- A rotina chamada “não consegue modificar o valor” do parâmetro real passado
- Este modo fornece apenas valores de entrada para uma dada rotina chamada
- Exemplo: passagens de parâmetro em Java

Passagem de Parâmetros Referência

- Neste modo, na transferência de execução da rotina chamadora para a chamada o parâmetro formal passa a ser um “sinônimo” do parâmetro real
- Com isso, qualquer modificação do parâmetro formal altera o parâmetro real
- Exemplos: os operadores & (em C) e var (em Pascal) precedendo a declaração de parâmetros numa função

Passagem de Parâmetros Referência

- Usualmente, este modo é preferido em relação ao modo por valor por causa da possível operação custosa de clonar um objeto
  - Na passagem do valor, o valor do argumento é copiado para o parâmetro da função

Passagem de Parâmetros Resultado

- Modo dual à passagem por valor
- Ou seja, este modo permite o retorno de valores de saída atualizados por uma dada rotina chamada
- O valor inicial do argumento passado é desconsiderado

Passagem de Parâmetros Valor/Resultado

- Combina os modos valor e resultado
- Difere da passagem por referência, pois as áreas de memórias para argumento e parâmetro são distintas entre as rotinas chamadora e chamada
- A execução entre este modo e o de referência poderia ser diferente, por exemplo, num programa *multithreaded*

Passagem de Parâmetros Nome

- Tipicamente utilizado quando queremos passar um argumento que será posteriormente avaliado
- Exemplo a seguir, em Algol, extraído do link abaixo: http://en.wikipedia.org/wiki/Jensen%27s_Device
- Imagine que queiramos computador a seguinte fórmula:

$$H_{100} = \sum_{i=1}^{100} \frac{1}{i}$$

Passagem de Parâmetros Nome

- Em Algol, isto pode ser feito da seguinte forma:

```
begin
   integer k;
   real procedure sum (i, lo, hi, term);
      value lo, hi;
      integer i, lo, hi;
      real term;
      comment term passado por nome, assim como i;
   begin
      real temp;
      temp := 0;
      for i := lo step 1 until hi do
         temp := temp + term;
      sum := temp
   end;
   comment observe a correspondência entre a notação matemática e a
chamada à função sum;
   print (sum (k, 1, 100, 1/k))
end
```

Passagem de Parâmetros Nome

- Os argumentos *i* e *1/i* são passados por nome, o que permite a expressão ser avaliada apenas dentro da função, quando é referenciada
- Para a função *sum* computar outras fórmulas, basta passar a equação correspondente
  - i e 1/(1 – i*i)
  - k e k*k
  - j e f(j)

Tipos de Valores Passados por Parâmetro

- Valores, Endereços, Funções, ...
- Valores de 1a classe:
  - Estes podem ser associados à uma variável, podem ser passados como parâmetro e podem ser retornados como resultado de uma função
- 2a classe
  - Estes podem ser associados à uma variável, podem ser passados como parâmetro
- 3a classe

Outras Características de Passagens por Parâmetro

- Algumas linguagens permitem que o número de parâmetros de uma função seja variável
- Por exemplo, observe os argumentos passados para as funções *scanf()* e *printf()*
- Em C isto é obtido terminando a lista de parâmetros com o parâmetro '...'

Outras Características de Passagens por Parâmetro

- Algumas linguagens permitem a definição de valores padrão (*default*) na passagem de parâmetros
- Desta maneira, quando algum argumento não é informado na chamada da função, o valor padrão é utilizado

Outras Características de Passagens por Parâmetro

```
#include "stdio.h" \ #include "stdlib.h" \ #include "string.h"
#include "stdarg.h"

float frac (int k = 0) { // Valor default: somente em C++
    return ++k;
}
int frac2(int k, ...) { // Número de argumentos variável
    va_list args;
    va_start(args, k);
    int proxi = va_arg(args, int);
    int proxi2 = atoi(va_arg(args, char *));
    va_end(args);
    return k + proxi + proxi2;
}
main() {
    printf("%3.1f\n", frac(20));
    printf("%d\n", frac2(5, 15, "25.0"));
}
```

Rotinas Genéricas

- Diversos mecanismos (estruturas, funções, etc) tem seus conceitos definidos independente do tipo de dado manipulado
  - Listas, Pilhas, Filas, …
  - Ordenação, Pesquisa, …
- Linguagens usualmente disponibilizam mecanismos genéricos para a implementação destes conceitos
- Em C, por exemplo, isto é obtido através do tipo *(void *)*

Rotinas Genéricas

```
#include "stdio.h"
#include "stdlib.h"

main() {
    ListaGen* l = NULL;
    char *nome = "vinicius";
    l = insere(l, nome, imprimeString);
    int *x = (int *)malloc(sizeof(int));
    (*x) = 10;
    l = insere(l, x, imprimeNumero);
    exibe(l);
}
```

Rotinas Genéricas

```
#include "stdio.h"
#include "stdlib.h"

typedef struct lista_gen {
    void* info;
    void (* impressao)(void *); // Campo que representa uma função
    struct lista_gen* prox;
} ListaGen;

ListaGen* insere(ListaGen *l, void* p, void impr(void *));
void imprimeString (void *p);
void imprimeNumero (void *p);
```

Rotinas Genéricas

```
...
ListaGen* insere(ListaGen *l, void* p, void impr(void *)) {
    ListaGen* n = (ListaGen *) malloc(sizeof(ListaGen));
    n->info = p; n->impressao = impr; n->prox = l;
    return n;
}
void imprimeString (void *p) {
    printf("%s\n", (char *)p);
}
void imprimeNumero (void *p) {
    printf("%d\n", *((int *)p));
}
void exibe(ListaGen *l) {
    ListaGen *laux;
    for (laux = l; laux != NULL; laux=laux->prox) {
     laux->impressao(laux->info);
    }
}
```