DCC024 Linguagens de Programação
2022.2

Passagem de parâmetros

Haniel Barbosa

UFMG | ICEx
DEPARTAMENTO DE
CIÊNCIA DA COMPUTAÇÃO

UFMG

Passagem de parâmetros

- Durante este curso temos lidado com declaração e chamada de funções

- Quais são diferentes formas em que a passagem de parâmetros para funções pode ser implementada?

Alguns conceitos

- Parâmetros formais e parâmetros reais

```
void f(int x, int y) {}
...
f(1,0);

LetFun("f", "x", ..., Call(Var "f", IConst 1))
```

Alguns conceitos

- Parâmetros formais e parâmetros reais

```
void f(int x, int y) {}
...
f(1,0);

LetFun("f", "x", ..., Call(Var "f", IConst 1))
```

- Como associar os parâmetros formais aos parâmetros reais?
  - Posicional
  - Nominal

Correspondências posicional e nominal

```
1  def div(dividend, divisor):
2    r = dividend / divisor
3    print( "Result = ", r)
4
5  div(5,1)
6  div(1,5)
```

Correspondências posicional e nominal

```
1  def div(dividend, divisor):
2    r = dividend / divisor
3    print( "Result = ", r)
4
5  div(5,1)
6  div(1,5)
```

```
1  div(dividend=1,divisor=5)
2  div(divisor=1,dividend=5)
```

Parâmetros opcionais

```
1  int mult(int a = 1, int b = 2, int c = 3) { return a * b * c; }
2
3  int main()
4  {
5    std::cout << mult(4, 5, 6) << std::endl;
6    std::cout << mult(4, 5) << std::endl;
7    std::cout << mult(5) << std::endl;
8    std::cout << mult() << std::endl;
9  }
```

Parâmetros opcionais em C

```c
int foo(size_t nargs, ...){
  int sum = 0;
  va_list ap;
  va_start(ap, nargs);
  for (int i = 0; i < nargs; ++i)
    sum += va_arg(ap, int);
  va_end(ap);
  return sum;
}
int main(){
  printf("%d\n", foo(0)); printf("%d\n", foo(1, 5)); printf("%d\n", foo(2, 5, 7
  printf("%d\n", foo(3, 5, 7, 9, 10));
}
```

Parâmetros opcionais em C

```c
int foo(size_t nargs, ...){
  int sum = 0;
  va_list ap;
  va_start(ap, nargs);
  for (int i = 0; i < nargs; ++i)
    sum += va_arg(ap, int);
  va_end(ap);
  return sum;
}
int main(){
  printf("%d\n", foo(0)); printf("%d\n", foo(1, 5)); printf("%d\n", foo(2, 5, 7
  printf("%d\n", foo(3, 5, 7, 9, 10));
}
```

```c
printf("%d\n", foo(10, 5, 7, 9, 11));
```

Parâmetros opcionais em C

```c
int foo(size_t nargs, ...){
  int sum = 0;
  va_list ap;
  va_start(ap, nargs);
  for (int i = 0; i < nargs; ++i)
    sum += va_arg(ap, int);
  va_end(ap);
  return sum;
}
int main(){
  printf("%d\n", foo(0)); printf("%d\n", foo(1, 5)); printf("%d\n", foo(2, 5, 7
  printf("%d\n", foo(3, 5, 7, 9, 10));
}
```

```c
printf("%d\n", foo(10, 5, 7, 9, 11));
```

```c
printf("%d\n", 0);
printf("%d %d\n", 0);
printf("%d %d\n", 0, 1, 2);
```

Passagem por valor

```c
void swap(int x, int y)
{
  int aux = x;
  x = y;
  y = aux;
}

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

Passagem por valor

```c
void swap(int* x, int* y)
{
  int aux = *x;
  *x = *y;
  *y = aux;
}

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

Passagem por referência

```
1  void swap(int& x, int& y)
2  {
3    int aux = x;
4    x = y;
5    y = aux;
6  }
7
8  int main()
9  {
10   int a = 2;
11   int b = 3;
12   printf("%d, %d\n", a, b);
13   swap(a, b);
14   printf("%d, %d\n", a, b);
15 }
```

Risco de passagem por referência: aliasing

```
1  void sigsum(int& n, int& ans)
2  {
3    ans = 0;
4    int i = 1;
5    while (i <= n)
6    {
7      ans += i;
8      i++;
9    }
10 }
11
12 int main()
13 {
14   int x = 10;
15   int y;
16   sigsum(x, y);
17   printf("x = %d, y = %d\n", x, y);
```

Risco de passagem por referência: aliasing

```
1  void sigsum(int& n, int& ans)
2  {
3    ans = 0;
4    int i = 1;
5    while (i <= n)
6    {
7      ans += i;
8      i++;
9    }
10 }
11
12 int main()
13 {
14   int x = 10;
15   int y;
16   sigsum(x, y);
17   printf("x = %d, y = %d\n", x, y);
```

```
1 x = 10;
2 sigsum(x, x);
3 printf("x = %d, y = %d\n", x, x);
```

Passagem por referência em Python

```
1  def f(a, L=[]):                    lst = []
2    L.append(a)                      f(0, lst)
3    print(L)                         f(1, lst)
4
5  def g(a):                          x = 0
6    a += 1                           g(x)
7    print(a)
```

Passagem por referência em Python

```
1  def f(a, L=[]):
2    L.append(a)
3    print(L)
4
5  def g(a):
6    a += 1
7    print(a)
```

```
lst = []
f(0, lst)
f(1, lst)

x = 0
g(x)
```

```
1  def f1(a, L=[]):
2    L.append(a)
3    print(L)
```

```
 f1(0)
f1(1)
```

Passagem por referência em Python

```
1  def f(a, L=[]):
2    L.append(a)
3    print(L)
4
5  def g(a):
6    a += 1
7    print(a)
```

```
lst = []
f(0, lst)
f(1, lst)

x = 0
g(x)
```

```
1  def f1(a, L=[]):
2    L.append(a)
3    print(L)
```

```
 f1(0)
f1(1)
```

```
1  def f2(a, L=0):
2    if L == 0:
3      L = []
4    L.append(a)
5    print(L)
```

```
f2(0)
f2(1)
```

Passagens com avaliações preguiçosas

- Nas passagens de parâmetros que vimos até o momento a atribuição de valores a parâmetros formais é sempre feita
  - Com isso os parâmetros reais são sempre avaliados no momento da chamada
  - *Eager evaluation* (avaliação gulosa)
- Alternativamente, passagens de parâmetro podem adiar a atribuição de valores a parâmetros formais
  - Apenas quando os parâmetros formais são usados avalia-se os parâmetros reais
  - *Lazy evaluation* (avaliação preguiçosa)
  - Questões de quantas vezes avaliar os parâmetros reais ou em que escopo tornam-se relevantes

Expansão de macros

```
1  define SWAP(X, Y)  \
2    {                 \
3      int tmp = X;    \
4      X = Y;          \
5      Y = tmp;        \
6    }
7
8  int main()
9  {
10   int a = 2;
11   int b = 3;
12   printf("%d, %d\n", a, b);
13   SWAP(a, b);
14   printf("%d, %d\n", a, b);
15 }
```

Expansão de macros

```
1  int x = 0;
2
3  int foo(){
4    x++;
5    return 1;
6  }
7
8  #define MAX(X, Y) ((X) > (Y) ? (X) : (Y))
9
10 int main(){
11   int y = MAX(0, foo());
12   printf("Max: %d, global x: %d\n", y, x);
13 }
```

Expansão de macros

```
1  int x = 0;
2  int z = 1;
3
4  int foo(){
5    x++;
6    return 1;
7  }
8
9  int bar(){
10   return z++;
11 }
12
13 #define MAX(X, Y) ((X) > (Y) ? (X) : (Y))
14
15 int main(){
16   int y = MAX(bar(), foo());
17   printf("Max: %d, global x, z: %d, %d\n", y, x, z);
18
19   y = MAX(bar(), foo());
20   printf("Max: %d, global x, z: %d, %d\n", y, x, z);
21 }
```

Expansão de macros

```
1  define SWAP(X, Y)  \
2    {                 \
3      int tmp = X;    \
4      X = Y;          \
5      Y = tmp;        \
6    }
7
8  int main(){
9    int a = 2;
10   int tmp = 15;
11   printf("Before: %d, %d\n", a, tmp);
12   SWAP(a, tmp);
13   printf("After: %d, %d\n", a, tmp);
14 }
```

Expansão de macros

```
1  define SWAP(X, Y)  \
2    {                 \
3      int tmp = X;    \
4      X = Y;          \
5      Y = tmp;        \
6    }
7
8  int main(){
9    int a = 2;
10   int tmp = 15;
11   printf("Before: %d, %d\n", a, tmp);
12   SWAP(a, tmp); -> {int tmp = a; a = tmp; tmp = tmp; };
13   printf("After: %d, %d\n", a, tmp);
14 }
```

Chamada por nome

- Parâmetros formais são funções para as avaliações dos parâmetros reais
- Parâmetros reais são avaliados no escopo de chamada

```
1  void swap(by-name int x, by-name int y){
2  {
3    int tmp = x;
4    x = y;
5    y = tmp;
6  }
7
8  int main(){
9    int a = 2;
10   int tmp = 15;
11   printf("Before: %d, %d\n", a, tmp);
12   swap(a, tmp);
13   printf("After: %d, %d\n", a, tmp);
14 }
```

Chamada por nome

```
1  void f(by-name int a, by-name int b){
2    b = 5;
3    b = a;
4  }
5
6  int g(){
7    int i = 3;
8    f(i + 1, i);
9    return i;
10 }
```

Chamada por necessidade

- Como chamada por nome mas utiliza memorização (*memoization*)

```
1  int x = 0;
2  int foo(){
3    x++;
4    return 1;
5  }
6
7  int max(by-name x, by-name y){ x > y ? x : y; }
8
9  int main(){
10   int y = max(0, foo());
11   printf("Max: %d, global x: %d\n", y, x);
12 }
```

Avaliação preguiçosa em Haskell

```
1  fib m n = m : (fib n (m+n))
```

Avaliação preguiçosa em Haskell

```haskell
fib m n = m : (fib n (m+n))
```

```haskell
getIt [] _ = 0
getIt (x:xs) 1 = x
getIt (x:xs) n = getIt xs (n-1)

getNthFib n = getIt (fib 0 1) n

getNthFib 4 = ...
```

Em resumo...

- Tipos de parâmetros:
  - *Formais*: aqueles declarados na lista de parâmetros da função
  - *Reais*: aqueles passados para a função, que substituirão os parâmetros formais

- Correspondência entre parâmetros reais e formais
  - *Posicional*: correspondência feita de acordo com a posição na chamada da função. Exemplo: quase todas as linguagens de programação.
  - *Nominal*: parâmetros são anexados a nomes, que os identificam durante a invocação da função. Exemplo: Python (que também tem posicional).

Em resumo...

- Tipos de passagem de parâmetros: avaliação gulosa (*eager*)
  - *Chamada por valor*: o parâmetro formal é como uma variável local no escopo da função, inicializada com o valor do parâmetro real no momento de chamada da função.
  - *Chamada por referência*: o parâmetro formal é um *alias* para o parâmetro real. Qualquer modificação afeta ambos, igual e simultaneamente.
  - Outros:
    - *Chamada por resultado*: o parâmetro formal é como uma variável local, porém não inicializada. Antes que a função retorne, o valor atual do parâmetro formal é copiado para o parâmetro real.
    - *Chamada por valor e resultado*: combinação de passagem por valor e por resultado: parâmetro formal como variável local, inicializada com o valor do parâmetro real, e, antes que a função termine, copia-se o valor atual do parâmetro formal para o real.

Em resumo...

- Tipos de passagem de parâmetros: avaliação preguiçosa (*lazy*)
  - *Chamada por expansão de macros*: o corpo da macro é executado no escopo de chamada. Parâmetros formais são substituídos pelos parâmetros reais e reavaliados a cada ocorrência utilizada, no escopo daquela ocorrência na macro.
  - *Chamada por nome*: parâmetros formais são substituídos pelos parâmetros reais e reavaliados, no escopo de chamada, a cada ocorrência utilizada.
  - *Chamada por necessidade*: como por nome mas o parâmetro real é avaliado somente na primeira ocorrência utilizada do correspondente parâmetro formal. O resultado é armazenado para otimizar subsequentes usos. A avaliação só é feita até onde é necessário pela chamada.

Gulosa vs Preguiçosa

- Em que condições vocês acham que vale a pena trocar uma chamada gulosa por uma preguiçosa?

Gulosa vs Preguiçosa

- Em que condições vocês acham que vale a pena trocar uma chamada gulosa por uma preguiçosa?
  - É frequente o uso do parâmetro formal no método invocado?

- Como automaticamente determinar se vale a pena fazer a troca?

Gulosa vs Preguiçosa

- Em que condições vocês acham que vale a pena trocar uma chamada gulosa por uma preguiçosa?
  - É frequente o uso do parâmetro formal no método invocado?
- Como automaticamente determinar se vale a pena fazer a troca?
  - Análise estática do fluxo de execução

Gulosa vs Preguiçosa

- Em que condições vocês acham que vale a pena trocar uma chamada gulosa por uma preguiçosa?
  - É frequente o uso do parâmetro formal no método invocado?

- Como automaticamente determinar se vale a pena fazer a troca?
  - Análise estática do fluxo de execução
  - Análise da frequência com que os caminhos otimizáveis são considerados
    - Dinamicamente, via *profiling*
    - Simbolicamente, via automatização de raciocínio lógico