DCC024 Linguagens de Programação

2022.1

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

```
def div(dividend, divisor):
  r = dividend / divisor
  print( "Result = ", r)

div(5,1)
div(1,5)
```

Correspondências posicional e nominal

```
def div(dividend, divisor):
  r = dividend / divisor
  print( "Result = ", r)

div(5,1)
div(1,5)


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

Parâmetros opcionais

```
int mult(int a = 1, int b = 2, int c = 3) { return a * b * c; }

int main()
{
  std::cout << mult(4, 5, 6) << std::endl;
  std::cout << mult(4, 5) << std::endl;
  std::cout << mult(5) << std::endl;
  std::cout << mult() << std::endl;
}
```

Parâmetros opcionais em C

```
#include <stdarg.h>
#include <stdio.h>

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

```
#include <stdarg.h>
#include <stdio.h>

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));

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

Parâmetros opcionais em C

```
#include <stdarg.h>
#include <stdio.h>

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));

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

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

Passagem por valor

```
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

```
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

```
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);
}
```

Risco de passagem por referência: *aliasing*

```
void sigsum(int& n, int& ans)
{
  ans = 0;
  int i = 1;
  while (i <= n)
  {
    ans += i;
    i++;
  }
}

int main()
{
  int x = 10;
  int y;
  sigsum(x, y);
  printf("x = %d, y = %d\n", x, y);
```

Risco de passagem por referência: *aliasing*

```
void sigsum(int& n, int& ans)
{
  ans = 0;
  int i = 1;
  while (i <= n)
  {
    ans += i;
    i++;
  }
}

int main()
{
  int x = 10;
  int y;
  sigsum(x, y);
  printf("x = %d, y = %d\n", x, y);

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

Passagem por referência em Python

```
def f(a, L=[]):
  L.append(a)
  print(L)

def g(a):
  a += 1
  print(a)
```

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

x = 0
g(x)
```

Passagem por referência em Python

```
def f(a, L=[]):
  L.append(a)
  print(L)

def g(a):
  a += 1
  print(a)

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

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

x = 0
g(x)


 f1(0)
f1(1)
```

Passagem por referência em Python

```
def f(a, L=[]):
  L.append(a)
  print(L)

def g(a):
  a += 1
  print(a)

def f1(a, L=[]):
  L.append(a)
  print(L)

def f2(a, L=0):
  if L == 0:
    L = []
  L.append(a)
  print(L)
```

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

x = 0
g(x)


 f1(0)
f1(1)


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

```
#define SWAP(X, Y)  \
  {                 \
    int tmp = X;    \
    X = Y;          \
    Y = tmp;        \
  }

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

Expansão de macros

```
int x = 0;

int foo(){
  x++;
  return 1;
}

#define MAX(X, Y) ((X) > (Y) ? (X) : (Y))

int main(){
  int y = MAX(0, foo());
  printf("Max: %d, global x: %d\n", y, x);
}
```

Expansão de macros

```
int x = 0;
int z = 1;

int foo(){
  x++;
  return 1;
}

int bar(){
  return z++;
}

#define MAX(X, Y) ((X) > (Y) ? (X) : (Y))

int main(){
  int y = MAX(bar(), foo());
  printf("Max: %d, global x, z: %d, %d\n", y, x, z);

  y = MAX(bar(), foo());
  printf("Max: %d, global x, z: %d, %d\n", y, x, z);
}
```

Expansão de macros

```
#define SWAP(X, Y)  \
  {                 \
    int tmp = X;    \
    X = Y;          \
    Y = tmp;        \
  }

int main(){
  int a = 2;
  int tmp = 15;
  printf("Before: %d, %d\n", a, tmp);
  SWAP(a, tmp);
  printf("After: %d, %d\n", a, tmp);
}
```

Expansão de macros

```
#define SWAP(X, Y)  \
  {                 \
    int tmp = X;    \
    X = Y;          \
    Y = tmp;        \
  }

int main(){
  int a = 2;
  int tmp = 15;
  printf("Before: %d, %d\n", a, tmp);
  SWAP(a, tmp); -> {int tmp = a; a = tmp; tmp = tmp; };
  printf("After: %d, %d\n", a, tmp);
}
```

Chamada por nome

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

```
void swap(by-name int x, by-name int y){
{
  int tmp = x;
  x = y;
  y = tmp;
}

int main(){
  int a = 2;
  int tmp = 15;
  printf("Before: %d, %d\n", a, tmp);
  swap(a, tmp);
  printf("After: %d, %d\n", a, tmp);
}
```

Chamada por nome

```
void f(by-name int a, by-name int b){
  b = 5;
  b = a;
}

int g(){
  int i = 3;
  f(i + 1, i);
  return i;
}
```

Chamada por necessidade

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

```
int x = 0;
int foo(){
  x++;
  return 1;
}

int max(by-name x, by-name y){ x > y ? x : y; }

int main(){
  int y = max(0, foo());
  printf("Max: %d, global x: %d\n", y, x);
}
```

Avaliação preguiçosa em Haskell

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

Avaliação preguiçosa em Haskell

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

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