Recursividade

Programação I
2021.2022

Teresa Gonçalves
tcg@uevora.pt

Departamento de Informática, ECT-UÉ

Como programar?

Perceber o problema

Dados e Resultados

Pensar numa solução

Dividir o problema em problemas mais simples

Resolver os problemas mais simples

Resolver o problema mais complexo

Implementar a solução

Utilizar funções para estruturar a resolução dos sub-problemas

Testar a solução

Fazer vários testes

Escolher valores que produzam comportamentos diferentes do programa

Sumário

Funções

Recursividade

Funções (revisão)

Função

Sequência de instruções com nome que realiza uma computação

Uma função

- Tem um nome
- Recebe argumentos
- Devolve um resultado
- É executada sempre que o seu nome é invocado

Definição vs. Utilização

Definição de uma função

Especifica

- o nome e parâmetros da função
- a sequência de instruções a executar

Possui

- Cabeçalho: nome, parâmetros
- Corpo: instruções a executar

Utilização (invocação)

Executa a sequência de instruções especificada na definição

- Utilizando os argumentos indicados na invocação

Notas

- A definição cria a função
- As instruções só são executadas quando a função é invocada
- Uma função tem de ser definida antes de ser invocada

Fluxo de execução

A invocação de uma função provoca um desvio no fluxo de execução

- Salta para o corpo da função
- Executa as instruções lá existentes
- Regressa, retomando o ponto onde tinha ficado

Valor de retorno

Tipo

`int, float`

`char`

Procedimento

Função que não retorna valor

Tipo especial: void

Exemplo

```
void intFrac( float x ){
    int i;
    float f;
    i = (int) x;
    f = x-(float)i;
    printf("A parte inteira: %d\n", i);
    printf("A parte fracionaria: %f", f);
    return;
}
```

Recursividade

Recursividade

Corpo da função == sequência de instruções

- Atribuições
- Outras instruções
- Invocação de funções

Recursividade

- Quando o corpo da função contém a **invocação à própria função**

Exemplo – contagem decrescente

```
void contagemD(int n){
  while(n>=0){
    printf("%d\n",n);
    n=n-1;
  }
}
```

```
void contagemDRec(int n){
  printf("%d\n",n);
  if(n>0){
    contagemDRec(n-1);
  }
}
```

contagemDRec

```
void contagemDRec(int n){
  printf("%d\n",n);
  if(n>0){
    contagemDRec(n-1);
  }
}
```

```
contagemDRec(3)
  print(3)
  contagemDRec(2)
    print(2)
    contagemDRec(1)
      print(1)
      contagemDRec(0)
        print(0)
      (termina)
    (termina)
  (termina)
(termina)
```

Critério de paragem

É essencial

Caso contrário existem infinitas invocações sucessivas!

Que esgotam os recursos de memória do computador...

Exemplo

```
void contagemDRec(int n){
  printf("%d\n",n);
  contagemDRec(n-1);
}
```

Factorial (definição iterativa)

n! = n*(n-1)*(n-2)*...*3*2*1

```
int factorial(int n){
  int res;
  res = 1;
  while(n>0){
     res = res*n;
     n = n-1;
  }
  return res;
}
```

Factorial (definição recursiva)

1! = 1

n! = n*(n-1)!

```
int factorial(int n){
  if(n==1)
    return 1;
  else
    return n*factorial(n-1);
}
```

Solução recursiva

Sequência de invocações da mesma função...
... mas com argumentos diferentes

factorial(n) → factorial(n-1) → factorial(n-2) → ...

Critério de paragem

n==1

Trata parte do problema e "junta-a" com o resultado das restantes invocações

n*resultado_da_funcao

Fibonacci

fib(0)=1

fib(1)=1

fib(n)=fib(n-1)+fib(n-2)

```
int fib(int n){
  int f;
  if(n<2)
    return 1;
  else {
    f=fib(n-1)+fib(n-2);
    return f;
  }
}
```

fib(5)

```
fib(5)
   fib(4)
      fib(3)
         fib(2)
            fib(1)    # devolve 1
            fib(0)    # devolve 1
         fib(1)    # devolve 1
      fib(2)
         fib(1)    # devolve 1
         fib(0)    # devolve 1
   fib(3)
      fib(2)
         fib(1)    # devolve 1
         fib(0)    # devolve 1
      fib(1)    # devolve 1
```

Características de uma função recursiva

Tem um caso base simples

que tem a solução e devolve um valor (condição de paragem)

Às vezes existe mais do que um caso base!

Tem uma forma de aproximar o problema ao caso base

descartar parte do problema para obter um problema mais simples

Tem uma chamada recursiva

que passa um problema mais simples à mesma função!

Pensar recursivamente

Ver a solução como uma versão mais pequena do mesmo problema

- Caso(s) base
  - Identificar o caso base e o que faz
  - Devolver o valor correto para o caso base
- A função recursiva é reduzida a uma condição **if-else**
  - o caso base devolve um valor
  - o caso não base chama recursivamente a mesma função com um parâmetro ou conjunto de dados **mais pequeno** que se **aproxima** do caso base

Caso base

- problema mais simples que a função pode resolver

Exemplos

Função soma()

Devolve a soma dos primeiros n inteiros

Parâmetros: 1

```
int n
```

Tipo resultado

```
int
```

Cabeçalho

```
int soma(int n)
```

Implementação soma()

Versão iterativa

```
int soma (int n) {
    int res=0;
    while (n>=1) {
        res = res+n;
        n = n-1;
    }
    return res;
}
```

Versão recursiva

```
int soma (int n) {
    if (n==1)
        return 1;
    return n+soma(n-1);
}
```

Função multiplo()

Devolve o i-ésimo múltiplo de n

Parâmetros: 2

```
int i
int n
```

Tipo resultado

```
int
```

Cabeçalho

```
int multiplo(int i, int n)
```

Implementação multiplo()

Versão iterativa (1)

```
int multiplo (int i, int n){
    return i*n;
}
```

Versão iterativa (2)

```
int multiplo (int i, int n){
    int res=0;
    while (i>=1) {
        res = res+n;
        i = i-1;
    }
    return res;
}
```

Implementação multiplo()

Versão iterativa

```
int multiplo (int i, int n){
    int res=0;
    while (i>=1) {
        res = res+n;
        i = i-1;
    }
    return res;
}
```

Versão recursiva

```
int multiplo (int i, int n){
    if (i==1)
        return n;
    return n+multiplo(i-1,n);
}
```