Modularização de Programas
Funções

Thierson Couto Rosa
thierson@inf.ufg.br

Instituto de Informática - INF/UFG

Lidando Com Problemas Complexos

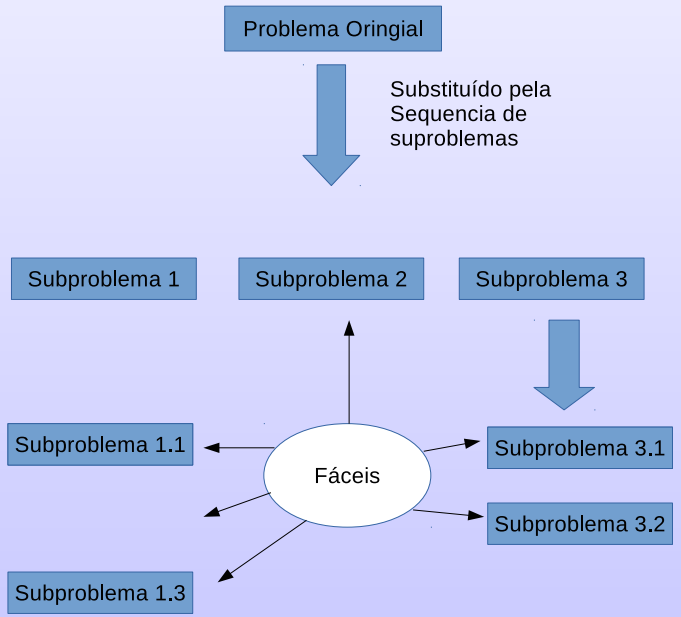
- A grande maioria dos problemas para os quais há uma solução computacional são problemas complexos.
- No nosso contexto, um problema complexo é um problema cuja solução envolve a realização de várias tarefas para resolve-lo (não significa que o problema seja necessariamente difícil).
- Uma abordagem comumente adotada é aquela conhecida como "Dividir para conquistar"ou ainda, "Técnnica dos refinamentos Sucessivos".

Técnica de Dividir para Conquistar

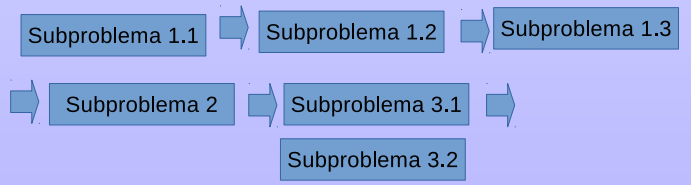
Ou Técnica dos Refinamentos Sucessivos.

- Tentar decompor (ou refinar) um problema em subproblemas menores, ou constituintes do problema original.
- Se um subproblema ainda for complexo, aplicar novamente a esse subproblema a técnica dos refinamentos sucessivos.
- A técnica pode ser aplicada repetidamente a cada subproblema até que restem apenas subproblemas considerados simples de serem resolvidos
- O uso da técnica pode levar a uma decomposição hierárquica do problema original, como exemplifica a figura a seguir.

Ilustração da Técnica de Dividir para Conquistar



Resolver o problema original implica em resolver os subproblemas
Na seguinte ordem:



Exemplo de Problema

Leia três valores reais (A, B e C) e verifique se eles formam ou não um triângulo. Em caso positivo, calcule o perímetro do triângulo e imprima a mensagem:

$$\text{Perimetro} = \text{XX.X}$$

Caso os valores não formem um triângulo, calcule a área do trapézio que tem A e B como base e C como altura, mostrando a mensagem:

$$\text{Area} = \text{XX.X}$$

Entrada: A entrada é formada por uma linha contendo três valores decimais separados um do outro por um espaço em branco.
Saída: A saída deve conter em uma única linha a frase apropriada.
Para que os três valores: A, B e C formem um triângulo as três condições abaixo devem ser satisfeitas: $|b-c| < a < b+c$; $|a-c| < b < a+c$; $|a-b| < c < a+b$;
A área de um trapézio é computada como Área$= \frac{(A+B)*C}{2}$.

Esboço da Solução

A solução Quebra o Problema em Subproblemas

- Ler os três valores a, b, c;
- Verificar se forma um triângulo
  - Se formarem
    - Computar o perímetro do triângulo;
    - Imprimir o perímetro do triângulo;
  - Senão
    - Computar a área do trapézio;
    - Imprimir a área do trapézio;

Refinamento da Solução

- Quais dos subproblemas listados acima são fáceis para nós? Quais requerem maior detalhamento?
- as leituras, as impressões e os cálculos do perímetro e área conseguimos codificar diretamente na linguagem C.
- O problema de verificar se a, b, c formam um triângulo requer maior detalhamento.

Encaminhamento da Solução

- Como sabemos resolver os demais problemas, nos concentramos no problema de detectar se os três valores formam um triângulo.
- Esquecemos temporariamente do problema original e nos dedicamos a esse problema.
- Quando terminarmos de resolver separadamente esse problema, reunimos todas as soluções dos subproblemas e temos a solução do problema original.

Modularizaçao da Solução

- A abordagem acima fez com que a solução do problema original fosse quebrada em módulos de soluções, cada uma resolvendo um subproblema.
- Se utilizamos a função principal (main()) para resolver o problema original como um todo, então seria interessante se cada subproblema ainda complicado ou complexo pudesse ter sua solução explicada através de um subprograma.
- As linguagens de programação possuem mecanismos para escrever subprogramas, também denominados *sub-rotinas, módulos, procedimentos* ou *funções*.
- Na linguagem C esse mecanismo é denominado *função*

Vantagens da Modularização de Programas

- A modularização permite lidar com a complexidade do problema original, isto é, a quantidade de situações diferentes que um problema original apresenta.
- Quando modularizamos a solução podemos nos concentrar na solução de um problema menor ou menos complicado a ser resolvido pelo módulo.
- Quando um subproblema se repete várias vezes dentro do problema original, podemos escrever um único sub-programa que resolve todas as instâncias daquele subproblema e chamar esse subprograma sempre que o subproblema aparecer.

Funções na linguagem C

Declaração de uma Função

- A declaração de uma função deve ocorrer antes da função principal **int** main().
- O corpo de uma função na linguagem C obedece ao seguinte formato:
  tipo nome da função (*lista de parâmetros*){
  lista de declarações de variáveis;
  lista de comandos;
  return(expressão)
  }

Funções na linguagem C

Componentes da Declaração

- O tipo da função pode ser qualquer tipo válido em C ou tipos criados pelo programador (veremos mais tarde).
- O nome de uma função deve obedecer às mesmas regras de formação de nomes de variáveis
- Uma função pode não retornar nenhum valor associado ao seu nome. Neste caso podemos usar **void** como tipo da função.
- A lista de parâmetros pode ser vazia, ou ter um número finito de declarações de parâmetros. Cada declaração é separada da outra por uma vírgula.

Funções na linguagem C

Componentes da Declaração - Cont.

- Após a lista de parâmetros segue um bloco, que é delimitado por "{" e"}".
- Dentro de do bloco pode haver dois componentes:
  - listas de declarações de variáveis locais (pode não existir)
  - lista de comandos (pode não existir);
- Quando a função tem um tipo, ela dever retornar uma expressão daquele tipo, através do comando `return()`, que deve estar na lista de comandos.

Funções na linguagem C

Exemplo

```
1  int etriangulo(double x, double y, double z){
2          if((x > abs(y–z)) && (x < (y + z) ) &&
3                          (y > abs(x–z)) && (y < (x + z) ) &&
4                          (z > abs(x–y)) && (z < x+ y ) ){
5                          return(1);
6                  }
7                  return(0);
8  }
```

Repare que a lista de parâmetros difere da lista de declaração de variáveis. Cada parâmetro vem precedido de seu tipo.

Funções na linguagem C

Exemplo

- A função anterior recebe como parâmetros três valores double.
- A função decide se os três parâmetros forma um triângulo.
- Em caso positivo a função retorna 1. Em caso negativo, retorna zero.
- Nesse exemplo não houve a necessidade de se declarar variáveis locais no bloco da função (o que está entre "{" e "}").
- Uma função pode chamar outra função. No exemplo, a função `etriangulo` chama a função matemática `abs` que calcula o valor absoluto de uma expressão.

Chamada à função

Exemplo

```
1  int main(){
2          double a,b,c;
3          scanf("%lf %lf %lf", &a, &b, &c);
4          if (etriangulo(a,b,c)){
5                  //Computar e imprimir perimetro
6                  printf("Perimetro = %.1lf\n", a+b+c);
7          }
8          else{
9                  //Imprimir e computar area do trapezio
10                 printf("Area = %.1lf\n", (b+a)*c/2.0);
11         }
12 }
```

Chamada de função

- A chamada a uma função é uma expressão cujo tipo deve ser compatível com o tipo da função e cujo valor é valor retornado pela função, caso a função retorne algum valor.
- No caso do exemplo, podemos chamar a função como parte da expressão do if na linha 4 no código do slide anterior.
- Quando há uma chamada a uma função, a função que está em execução (`main()` no caso) é interrompida e a execução é transferida para o código da função chamada.
- Quando a função chamada termina a execução da função que chamou é retomada imediatamente após a chamada.

Chamada de função

- Se a função chamada retorna algum valor, a chamada é substituída pelo valor retornado. No caso do exemplo, a chamada à função `triangulo` é substituída por zero ou um.
- Quando uma função é chamada podemos passar expressões como parâmetros para a função chamada.
- As expressões são avaliadas durante a chamada e seus valores são **copiados** para os parâmetros correspondentes da função chamada.

Chamada de função

- No caso da chamada da linha 4, as três expressões correspondem, respectivamente, aos valores das três variáveis `a,b,c`.
- Na função `etriangulo` ocorre chamadas à função `abs()` do C. Nesse caso passamos expressões aritméticas como parâmetros para a função `abs()`.

Passagens de Parâmetros

Passagem por Valor

- Na passagem por valor, os parâmetros da função correspondem a cópias dos valores correspondentes passados como parâmetros na chamada.
- Se os valores dos parâmetros da função são modificados pela função, esses valores não afetam os argumentos correspondentes da chamada, porque apenas as cópias são alteradas e não os dados nos argumentos da chamada.
- A passagem por valor é útil somente quando os parâmetros da função são parâmetros de entrada, isto é, eles não são alterados pela função ou se são, essa alteração não necessita ser visualizada no código que realiza a chamada.