

MC-102 — Aula 10

Funções

Instituto de Computação – Unicamp

29 de Agosto de 2012

Roteiro

- 1 Protótipo de funções
- 2 Escopo de Variáveis: variáveis locais e globais
- 3 Exemplo Maior
- 4 Exercícios

Definindo funções depois do `main`

Até o momento, aprendemos que devemos definir as funções antes do programa principal, mas o que ocorreria se declarássemos depois?

Declarando funções depois do main

```
#include <stdio.h>

int main () {
    float a = 0, b = 5;
    printf ("%f\n", soma (a, b));
    return 0;
}

float soma (float op1, float op2) {
    return (op1 + op2);
}
```

Dependendo do compilador, ocorre um erro de compilação!

Declarando uma função sem defini-la

- Para organizar melhor um programa, e podermos implementar funções em partes distintas do arquivo fonte, utilizamos **protótipos de funções**.
- Protótipos de funções correspondem a primeira linha da definição de uma função contendo: tipo de retorno, nome da função, parâmetros e **por fim um ponto e vírgula**.

```
tipo nome (tipo parâmetro1, ..., tipo parâmetroN);
```

- O protótipo de uma função deve aparecer antes do seu uso.
- Em geral coloca-se os protótipos de funções no início do seu arquivo do programa.

Protótipo de Funções

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

```
float soma (float op1, float op2);
```

```
int main () {  
    float a = 0, b = 5;  
    printf ("%f\n", soma (a, b));  
    return 0;  
}
```

```
float soma (float op1, float op2) {  
    return (op1 + op2);  
}
```

Protótipo de Funções

```
#include <stdio.h>

float soma (float op1, float op2);
float subtr (float op1, float op2);

int main () {
    float a = 0, b = 5;
    printf ("%f\n %f\n", soma (a, b), subtr(a, b));
    return 0;
}

float soma (float op1, float op2) {
    return (op1 + op2);
}

float subtr (float op1, float op2) {
    return (op1 - op2);
}
```

Variáveis locais e variáveis globais

- Uma variável é chamada **local** se ela foi declarada dentro de uma função. Nesse caso, ela existe somente dentro daquela função e após o término da execução da mesma, a variável deixa de existir.

Variáveis Parâmetros também são variáveis locais

- Uma variável é chamada **global** se ela for declarada fora de qualquer função. Essa variável é visível em todas as funções. Qualquer função pode alterá-la e ela existe durante toda a execução do programa.

Em geral o programa é organizado da seguinte forma:

```
#include <stdio.h>
#include <outras bibliotecas>
```

Protótipos de funções

Declaração de Variáveis Globais

```
int main(){
    Declaração de variáveis locais
    Comandos;
}

int fun1(Parâmetros){ //Parâmetros também são locais
    Declaração de variáveis locais
    Comandos;
}

int fun2(Parâmetros){ //Parâmetros também são locais
    Declaração de variáveis locais
    Comandos;
}

...
...
```

Escopo de variáveis

- O **escopo** de uma variável determina de quais partes do código ela pode ser acessada.
- A regra de escopo em C é bem simples:
 - ▶ As variáveis globais são visíveis por todas as funções.
 - ▶ As variáveis locais são visíveis apenas na função onde foram declaradas.

Escopo de variáveis

```
#include<stdio.h>

void fun1();
int fun2(int local_b);

int global;

int main() {
    int local_main;
    /* Neste ponto são visíveis global e local_main */
}

void fun1() {
    int local_a;
    /* Neste ponto são visíveis global e local_a */
}

int fun2(int local_b){
    int local_c;
    /*Neste ponto são visíveis global, local_b e local_c*/
}
```

Escopo de variáveis

- É possível declarar variáveis locais com o mesmo nome de variáveis globais.
- Nesta situação, a variável local “esconde” a variável global.

```
#include <stdio.h>

void fa();

int nota = 10;

int main(){
    nota = 20;
    fa();
}

void fa() {
    int nota;
    nota = 5;
    /* Neste ponto nota é a variável local. */
}
```

Outro exemplo:

```
#include <stdio.h>

void fun1();
void fun2();

int x;
int main(){
    x = 1;
    fun1();
    fun2();
    printf("%d\n", x);
}

void fun1(){
    x = x +1;
    printf("\n%d",x);
}

void fun2(){
    int x = 3;
    printf("\n%d",x);
}
```

O que será impresso ?

```
#include <stdio.h>

void fun1();
void fun2();

int x = 1;
int main(){
    int x=1;
    fun1();
    fun2();
    printf("%d\n", x);
}

void fun1(){
    x = x + 1;
    printf("\n%d",x);
}

void fun2(){
    int x = 4;
    printf("\n%d",x);
}
```

O que será impresso ?

Exemplo Maior

- Em uma das aulas anteriores vimos como testar se um número é primo:

```
divisor = 2;
eprimo=1;
while(divisor<=candidato/2) {
    if(candidato % divisor == 0){
        eprimo=0;
        break;
    }
    divisor++;
}
if(eprimo)
    printf(" %d, ", candidato);
```

Exemplo Maior

- Depois usamos este código para imprimir os n primeiros números primos:
- Veja no próximo slide.

Exemplo Maior

```
int main(){
    int divisor=0, n=0, eprimo=0, candidato=0, primosImpr=0;
    printf("\n Digite numero de primos a imprimir:");
    scanf("%d",&n);
    if(n>=1){
        printf("2, ");
        primosImpr=1;
        candidato=3;
        while(primosImpr < n){
            divisor = 2;
            eprimo=1;
            while( divisor <= candidato/2 ){
                if(candidato % divisor == 0){
                    eprimo=0;
                    break;
                }
                divisor++;
            }
            if(eprimo){
                printf("%d, ",candidato);
                primosImpr++;
            }
            candidato=candidato+2;//Testa proximo numero
        }
    }
```

Exemplo Maior

- Refazer código com alteração: Se o número de primos a ser impresso é negativo usaremos o valor absoluto deste.
- Podemos criar uma função que testa se um número é primo ou não (note que isto é exatamente um bloco logicamente bem definido).
- Vamos criar também uma função que retorna o valor absoluto de um número.
- Depois fazemos chamadas para estas funções.

Exemplo Maior

```
#include <stdio.h>

int ePrimo(int candidato); //retorna 1 se candidato e primo; e 0 caso contrário
int valorAbs(int x); //retorna valor absoluto de x

int main(){
    int divisor=0, n=0, eprimo=0, candidato=0, primosImpr=0;

    printf("\n Digite numero de primos a imprimir:");
    scanf("%d",&n);
    n = valorAbs(n);
    if(n >= 1){
        printf("2, ");
        primosImpr = 1;
        candidato = 3;
        while(primosImpr < n){
            if( ePrimo(candidato) ){
                printf("%d, ",candidato);
                primosImpr++;
            }
            candidato=candidato+2;
        }
    }
}
```

Exemplo Maior

```
int valorAbs(int x){
    if(x < 0)
        return -1*x;
    else
        return x;
}

int ePrimo(int candidato){
    int divisor, eprimo;
    divisor = 2;
    eprimo=1;
    while( divisor <= candidato/2){
        if(candidato % divisor == 0){
            eprimo=0;
            break;
        }
        divisor++;
    }
    if(eprimo)
        return 1;
    else
        return 0;
}
```

Exemplo Maior

- O código é mais claro quando utilizamos funções.
- Também é mais fácil fazer alterações.
- Exemplo: queremos otimizar o teste de primalidade, e para tanto não vamos testar todos os divisores $2, \dots, (\textit{candidato}/2)$.
 - ▶ Testar se número é par maior que 2 (não é primo).
 - ▶ Se for ímpar, testar apenas os divisores ímpares $3, 5, 7, \dots$
- O uso de funções facilita modificações no código. Neste caso altera-se apenas a função **ePrimo**.

Exemplo Maior

Função **ePrimo** é alterada para:

```
int ePrimo(int candidato){
    int divisor, eprimo;

    if( (candidato>2) && (candidato % 2 == 0) )//se for par > 2
        return 0;

    divisor = 3;
    eprimo=1;
    while( divisor <= candidato/2 ){
        if(candidato % divisor == 0){
            eprimo=0;
            break;
        }
        divisor = divisor + 2; //divisores impares
    }
    if(eprimo)
        return 1;
    else
        return 0;
}
```

Exercício

- Escreva uma função em C, para computar a raiz quadrada de um número positivo. Use a idéia abaixo, baseada no método de aproximações sucessivas de Newton. A função deverá retornar o valor da vigésima aproximação.

Seja Y um número, sua raiz quadrada é raiz da equação

$$f(x) = x^2 - Y.$$

A primeira aproximação é $x_1 = Y/2$. A $(n + 1)$ -ésima aproximação é

$$x_{n+1} = x_n - \frac{f(x_n)}{f'(x_n)}$$