UniSALESIANO – Araçatuba – Linguagem Científica

Funções – Passagem de Parâmetro por valor / referência

Introdução

Em C, diferentemente de outras linguagens como Pascal, todas as ações ocorrem dentro de funções. Na linguagem C não há conceito de um programa principal, o que existe é uma função chamada `main` que é sempre a primeira a ser executada.

A forma geral de uma função em C é a seguinte:

```
tipo nome (tipo nome1, tipo nome2, ..., tipo nomeN )
{
    declaração das variáveis
    corpo da função
}
```

O `tipo` na definição da função especifica o tipo do resultado que será devolvido ao final da execução da função. Caso nenhum tipo seja especificado o compilador assume que um tipo inteiro é retornado. O tipo `void` pode ser usado para declarar funções que não retornam valor algum.

Há duas maneiras básicas de terminar a execução de uma função. Normalmente usa-se o comando return para retornar o resultado da função. Portanto, quando o comando

`return` *`expressão`*`;`
for executado, o valor da *expressão* é devolvido para a função que chamou. Quando não há valor para retornar o comando `return` não precisa ser usado e a função termina quando a chave que indica o término do corpo da função é atingido.

O nome da função é qualquer identificador válido. A lista de parâmetros é uma lista, separada por vírgulas, de variáveis com seus tipos associados. É possível que existam funções que não tenham lista de parâmetros, mas ainda assim é necessário que os parênteses sejam usados.

Os parâmetros são valores que a função recebe para realizar as tarefas para as quais foi programada. Por exemplo, uma função que calcule a raiz quadrada de um número do tipo `float`, deve declarar como parâmetro uma variável deste tipo para receber o valor.

É importante notar que diferentemente de declarações de variáveis onde podemos associar vários nomes de variáveis a uma declaração como em

```
int a, dia, mes, i;
```
na lista de parâmetros é necessário associar um tipo a cada variável como no exemplo abaixo:
```
float media (float n1, float n2, float n3)
```

Suponha que uma determinada função, A, deseje usar uma outra função, B. A função A deve colocar no local desejado o nome da função B e a lista de valores que deseja passar. Por exemplo, uma função que deseje usar a função `media` , cujo protótipo foi definido acima, para calcular a média de três valores, `nota1, nota2` e `nota3`, deve escrever no local onde quer que a média seja calculada o seguinte comando:

```
resultado = media(nota1, nota2, nota3);
```

onde `resultado` é a variável que vai receber a média calculada.

É importante notar que os tipos e o número de parâmetros que aparecem na declaração da função e na sua chamada devem estar na mesma ordem e ter tipos equivalentes. Se os tipos são incompatíveis, o compilador não gera um erro, mas podem ser gerados avisos na compilação e resultados estranhos.

Protótipos de Funções

O padrão ANSI extendeu a declaração da função para permitir que o compilador faça uma verificação mais rígida da compatibilidade entre os tipos que a função espera receber e àqueles que são fornecidos.

Protótipos de funções ajudam a detectar erros antes que eles ocorram, impedindo que funções sejam chamadas com argumentos inconsistentes.

A forma geral de definição de um protótipo é a seguinte:

```
tipo nome (tipo nome1, tipo nome2, ..., tipo nomeN);
```

O exemplo mostra a declaração de uma função e seu protótipo.

```
#include<stdio.h>

/* Prototipo da funcao */
int soma (int, int);


/* Funcao Principal */
int main()
{

     int a=5, b=9;
     cout<<" Soma ="<< soma(a,b);
     getch();

         return 0;
}

/* Definicao da funcao */
int soma(int a, int b)
```

```
{

    return a+b;
}
```

Escopo de Variáveis

Variáveis podem ser usadas dentro de uma função particular, e somente dentro desta função, ou pode ocorrer que uma ou mais variáveis precisem ser acessíveis à diversas funções diferentes. Por esta razão temos que definir onde as variáveis de um programa podem ser definidas e a partir deste local inferir onde elas estarão disponíveis.

As variáveis podem ser declaradas basicamente em três lugares: dentro de funções, fora de todas as funções e na lista de parâmetros das funções. As variáveis dentro das funções são chamadas de variáveis locais, as que aparecem fora de todas as funções chamamos de variáveis globais e aquelas que aparecem na lista de parâmetros são os parâmetros formais.

É importante notar que em `C` todas as funções estão no mesmo nível, por isto não é possível definir uma função dentro de outra função.

Variáveis Locais

As variáveis locais são aquelas declaradas dentro de uma função. Elas passam a existir quando do início da execução do bloco de comandos ou função onde foram definidas e executado e são destruídas ao final da execução do bloco. Uma variável local só pode ser referenciada, ou seja usada, dentro das funções onde foram declaradas. Outro ponto muito importante é que como as variáveis locais deixam de existir ao final da execução da função, elas são invisíveis para outras funções do mesmo programa. O código que define uma função e os seus dados são particulares a função.

No programa abaixo podemos ver alguns exemplos de variáveis locais. Na função `main` temos cinco variáveis locais: `numero, potencia, continua, para, linha`. A função `Eleva` possui somente a variável `res`, enquanto que `ParaMaiusculas` não possui nenhuma variável local. A única variável nesta função e `c` que é um parâmetro.

```
#define SIM 1
#define NAO 0

float Eleva (float, int);
char ParaMaiuscula (char );

void main(){
     float numero;
     int potencia;
     char continua;
     int para;
```

```
     char linha[80];

     do {
        cout<<"Entre com um numero";
        gets(linha); numero = atof(linha);
        cout<<"Entre com a potencia";
        gets(linha); potencia = atoi(linha);
        cout<<numero<<" Elevado a"<<potencia<<"= "<<Eleva(numero,
potencia));
        cout<<"Continua? [S]im ou [N]ao? ";
        continua = getchar();
        getchar();
        continua = ParaMaiuscula(continua);
        para = continua == 'S'? NAO : SIM;
    } while (!para);
}

float Eleva(float a, int b){
    float res = 1.0;

    for ( ; b>0; b--) res *= a;
    return res;
}

char ParaMaiuscula ( char c ) {
    if ('a' <= c && c <= 'z')
        c = c - 'a' + 'A';
    return c;
}
```

Alguns autores usam o termo *variáveis automáticas* para se referir as variáveis locais. Em `C` existe a palavra chave `auto` que pode ser usada para declarar variáveis locais. No entanto, como todas as variáveis locais são por definição automáticas raramente se usa esta palavra chave.

Observe que um bloco de comandos se inicia em um "{" e termina em um "}". O bloco de comandos mais usado para definir uma variável é a função. Todas as variáveis que serão usadas dentro de um bloco de comandos precisam ser declaradas antes do primeiro comando do bloco. Declarações de variáveis, incluindo sua inicialização, podem vir logo após o abre chaves que inícia um bloco de comandos, não somente o que começa uma função. O exemplo abaixo ilustra este tipo de declaração:

```
#include<stdio.h>
void main(){
     int i;

     for (i=0; i<10; i++){
         int t;
        cin>>t;
         cout<<i*t;
     }
}
```

Existem algumas vantagens em se declarar variáveis dentro de blocos. Como as variáveis somente passam a existir quando o bloco passa a ser executado, o programa ocupa menos espaço de memória. Isto porque se a execução do bloco for condicional a

variável pode nem ser alocada. Outra vantagem é que como a variável somente existe dentro do bloco pode-se controlar melhor o uso da variável, evitando erros de uso indevido da variável.

Variáveis Globais

As variáveis globais são definidas fora de qualquer função e são portanto disponíveis para qualquer função. Este tipo de variável pode servir como uma canal de comunicação entre funções, uma maneira de transferir valores entre elas.

Por exemplo, se duas funções tem de partilhar dados mais uma não chama a outra, uma variável global tem de ser usada.

Parâmetros Formais

As variáveis que aparecem na lista de parâmetros da função são chamadas de parâmetros formais da função. Eles são criados no início da execução da função e destruídos no final.

Parâmetros são valores que as funções recebem da função que a chamou. Portanto, os parâmetros permitem que uma função passe valores para outra. Normalmente os parâmetros são inicializados durante a chamada da função, pois para isto que foram criadas. No entanto, as variáveis que atuam como parâmetros são iguais a todas as outras e podem ser modificadas, operadas, etc, sem nenhuma restrição.

Passagem de Parâmetros por Valor

Parâmetros podem ser passados para funções de duas maneiras: passagem por valor ou passagem por referência.

Na passagem por valor uma cópia do valor do argumento é passado para a função. Neste caso a função que recebe este valor ao fazer modificações no parâmetro não estará alterando o valor original que somente existe na função que chamou.

Passagem de Parâmetros por Referência

Na passagem por referência o que é passado para a função é o endereço do parâmetro e portanto a função que recebe pode através do endereço modificar o valor do argumento na função que chamou,

Para a passagem de parâmetros por referência é necessário o uso de ponteiros. Este assunto será discutido no próximo capítulo e portanto neste capítulo estaremos usando somente funções com passagem por valor.

```
#include <iostream>

using namespace std;



void trocar (int &a, int &b) {

   int aux;

   aux = a;

   a = b;

   b = aux;

}

 main () {

   int var1 = 10, var2 = 50;

   cout << "O valor de var1 e " << var1 << endl;

   cout << "O valor de var2 e " << var2 << endl;

   cout<<"\n\n\n";

   trocar (var1, var2);

   cout << "O valor de var1 e " << var1 << endl;

   cout << "O valor de var2 e " << var2 << endl;

   cout<<"\n\n\n";

   system("pause");

}
```

O Comando return

O comando `return` é usado para retornar o valor calculado para a função que chamou. Qualquer expressão pode aparecer no comando, que tem a a seguinte forma geral:

```
return expressão
```

A função que chamou é livre para ignorar o valor retornado. Além disso a função pode não conter o comando e portanto nenhum valor é retornado e neste caso a função termina quando o último comando da função é executado. Quando o comando `return` não existe o valor de retorno é considerado indefinido. As funções que não retornam valores devem ser declaradas como do tipo `void`.

É importante observar que funções que são declaradas com um tipo válido podem ser incluídas em qualquer expressão válidas em `C`.

Passagem de Vetores e Matrizes

Matrizes são um caso especial e excessão a regra que parâmetros são passados sempre por valor. Como veremos mais adiante, o nome de um vetor corresponde ao endereço do primeiro elemento do array, Quando um vetor é passado como parâmetro, apenas o endereço do primeiro elemento é passado.

Existem basicamente três maneiras de declarar um vetor como um parâmetro de uma função. Na primeira ele é declarado como tem sido apresentado em todos os exemplos até agora. O exemplo mostrado abaixo mostra um programa que usa uma função para descobrir quantas vezes um caracter ocorre em um vetor. Observe que a dimensão do vetor foi declarada explicitamente.

```
#include<stdio.h>
#include<conio.h>
#define DIM 80

char conta (char v[], char c);

void main(){
     char linha[DIM];
     char c;
     int maiusculas[26], minusculas[26];

     puts("Entre com uma linha");
     gets (linha);
     for (c='a'; c<='z'; c++)
         minusculas[c-'a'] = conta(linha, c);
     for (c='A'; c<='Z'; c++)
         maiusculas[c-'A'] = conta(linha, c);
     for (c='a'; c<='z'; c++)
         if (minusculas[c-'a'])
            printf("%c apareceu %d vezes\n", c, minusculas[c-'a']);
     for (c='A'; c<='Z'; c++)
         if (maiusculas[c-'A'])
            printf("%c apareceu %d vezes\n", c, maiusculas[c-'A']);
    getch();
```

```
}

char conta (char v[DIM], char c){
     int i=0, vezes=0;

     while (v[i] != '\0')
           if (v[i++] == c) vezes++;
     return vezes;
}
```

Uma outra maneira, leva em conta que apenas o endereço do vetor é passado. Neste modo o parâmetro é declarado como um vetor sem dimensão. Isto é perfeitamente possível porque a função somente precisa receber o endereço onde se encontra o vetor. Além disso C não confere limites de vetores e portanto a função precisa do endereço inicial do vetor e uma maneira de descobrir o final do vetor. Esta maneira pode ser, por exemplo, uma constante, ou o caracter '\0' em um vetor de caracteres.

O exemplo mostra este modo de passar vetores com um programa que inverte o conteúdo de um vetor.

```
#include<stdio.h>
#include<conio.h>
#define DIM 6
void Le_vetor (int v[], int tam);
void Imprime_vetor (int v[], int tam);
void Inverte_vetor (int v[], int tam);

void main(){
     int v[DIM];

     Le_vetor(v, DIM);
     Imprime_vetor (v, DIM);
     Inverte_vetor (v, DIM);
     Imprime_vetor (v, DIM);
     getch();
}

void Le_vetor (int v[], int tam){
     int i;

     for (i=0; i<tam; i++){
         printf("%d = ? ", i);
         scanf("%d", &v[i]);
     }
}

void Imprime_vetor (int v[], int tam){
     int i;

     for (i=0; i<tam; i++)
         printf("%d = %d\n", i, v[i]);
}

void Inverte_vetor (int v[], int tam){
     int i, temp;

     for (i=0; i<tam/2; i++){
         temp = v[i];
```

```
            v[i] = v[tam-i-1];
            v[tam-i-1] = temp;
        }
    }
```