

Leitura-5

Funções e ponteiros

Paulo Vieira

Agenda

Funções definição

passagem de argumentos

conversão de tipos

classes de armazenamento

recursividade

Funções definição

Uma função é um módulo de código independente que tem:

- um nome
- executa uma tarefa específica
- pode retornar um valor para o programa que a chama
- o nome da função deve ser único em todo o programa

```
<tipo_dados> <nome_da_funcao> (<lista_parâmetros>){  
    <declarações locais>  
    <instruções>  
}
```

Exemplos de funções

Exemplo 1:

```
int le_digito(void){
    int c;
    do
        getchar();
    while(c<'0' || c>'9');
    return c
}
```

Exemplo 2:

```
void beep(int num_beeps){
    while(num_beeps-- >0);
    putchar('\a');
}
```

Exemplo 3:

```
#include <stdio.h>
float raiz2(float x); // protótipo

void main(){
    float a;
    printf("\nIntroduza um numero:");
    scanf("%f",&a);
    printf("A raiz quadrada de %f é %f",a,raiz2(a));
}

float raiz2(float x){
    float y,z;
    do{
        y=z;
        y=(z+x/z)/2;
    }while(z!=y);
    return y;
}
```

operador de referencia, &.

endereço de memória da variável ch1

valor da variável ch1

valor da variável ch1 em int da tabelas ASCII

RAM memory

		1	2	3	4	5	6	7	8
ch1	0x7fff553c6ac3	0	1	1	0	0	0	0	1
x1	0x7fff553c6ac4	0	0	0	0	0	0	0	0
		0	0	0	0	0	0	0	0
		0	0	0	0	0	0	0	0
		1	1	0	1	0	0	1	0

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

```
int main()
```

```
{
```

```
char ch1='a';
```

```
int x1=210;
```

```
printf("Hello World,%p,%c,%d,%ld,%p,%ld",&ch1,ch1,ch1,sizeof(ch1),&x1,sizeof(x1));
```

```
return 0;
```

```
}
```

endereço de memória de x1, &x1

tamanho em bytes do espaço de memória da variável ch1, 1Byte

tamanho em bytes do espaço de memória da variável x1, 4Bytes

```
Hello World,0x7fff553c6ac3,a,97,1,0x7fff553c6ac4,4
```

o operador de desreferencia, *, ponteiro (apontador)

```
9  #include <stdio.h>
10
11 int main()
12 {
13     int *x1;
14     int x2=210;
15     x1=&x2;
16     printf("Hello World,%p,%d,%p,%d",&x2,x2,x1,*x1);
17
18     return 0;
19 }
20
```

input

Hello World,0x7fff0edc2e9c,210,0x7fff0edc2e9c,210

um ponteiro armazena, como valor, um endereço de memória, esse endereço é chamado o valor do ponteiro

o valor contido no endereço de memória que é valor de um ponteiro chama-se o valor apontado pelo ponteiro

O valor de x1 é o endereço de memória de x2

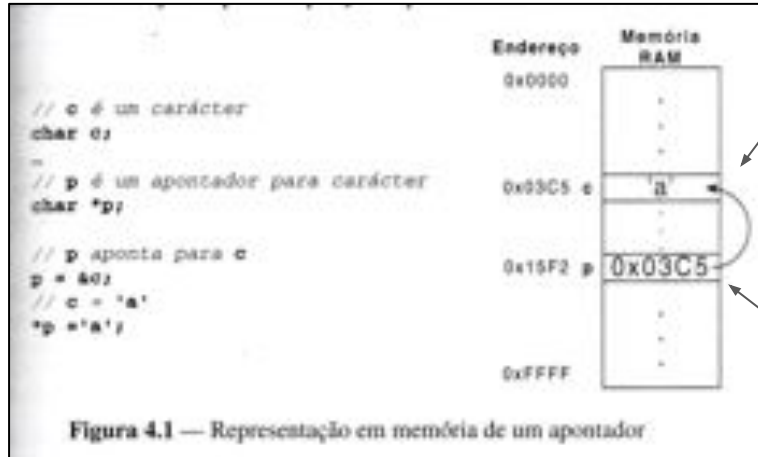
- x1 é um ponteiro
- Para obter o valor apontado por x1 escreve-se *x1, que é 200

ponteiro p, valor do ponteiro p, valor apontado por p

<tipo_de_dados> *p
ou
<tipo_de_dados>* p

p
(é um endereço de memória)

*p



valor
apontado
por p, *p='a'

valor do
ponteiro p
ou valor
de p,
p=0x03C5

microcontroladores

simuladores de microcontroladores

<https://www.oshonsoft.com/downloads.php>

atmega-datasheet, [ATMEGA328 pdf](#), [ATMEGA328 description](#), [ATMEGA328 datasheets](#), [ATMEGA328 view :: ALLDATASHEET ::](#)

2- Passagem de argumentos

Argumentos: Lista de variáveis, constantes, ou expressões colocadas entre os parênteses da função aquando da chamada desta.

parâmetros: lista de declarações de variáveis aquando da definição da função

Exemplo:

chamada da função:

troca(2,3)

cabeçalho da função: void troca(int x,int y);

No C a passagem dos argumentos é feita por valor

```
#include <stdio.h>
void troca(int x,int y);
```

parametros

```
int main()
{
    int a=2, b=3;
    troca(2,3);
    printf("Hello World a=%d, b=%d",a,b);
    return 0;
}
```

Hello World a=2, b=3

argumentos

```
void troca(int x,int y){
    int z=0;
    z=x;
    x=y;
    y=z;
}
```

parametros

passagem de argumentos por valor. A alteração de variáveis an função não altera o valor destas na zona de código que invoca a função

2- Passagem de argumentos

passagem de variáveis por referência (é possível)

```
#include <stdio.h>
void troca(int *x,int *y);
```

```
int main()
{
    int a=2, b=3;
    troca(&a,&b);
    printf("Hello World a=%d, b=%d",a,b);
    return 0;
}
```

endereços de
memória da
variável b

Hello World a=3, b=2

apontadores para int,
operador de desreferencia

```
void troca(int *x,int *y){
    int z=0;
    z=*x; // a z é atribuído o conteúdo do endereço x
    *x=*y; // a x é atribuído o conteúdo do endereço y
    *y=z; // a y é atribuído o conteúdo de z
}
```

passagem de variáveis por referência (as alterações das variáveis na função são válidas tb na zona de código onde a função é chamada)

Strings | Conversão de tipo

atoi() - função que converte uma string num inteiro

conversões automáticas

```
#include <stdio.h>
#include <stdlib.h>
void main ()
{
    char* a;
    a="IPG-Guarda";
    printf("%d,%s",atoi("12345"),*
&a);
}
```

ponteiros

scanf com
ponteiros no
necessita usar &

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

```
int main() {
    char *s;
    printf("enter the string : ");
    scanf("%s", s);
    printf("you entered %s\n", *&s);
    return 0;
}
```

strings (ponteiro de char)

scanf (em que o argumento
são ponteiros)

printf - ponteiro

As Storage classes

Uma storage class define: o escopo (visibilidade) e o tempo de vida de uma variável e/ou função no interior de um programa em C

Precedem o tipo que modificam

Existem 4 tipos diferentes de Storage classes em C:

- auto
- register
- static
- extern

A auto Storage Class

Erro de compilação, auto só declara variáveis locais

A auto storage class é a classe de armazenamento padrão para todas as variáveis locais

São definidas duas variáveis na mesma classe de armazenamento.

variáveis armazenadas na RAM

'auto' só pode ser usado no interior de funções, ou seja, variáveis locais.

'auto' permite criar blocos de código para valores de variáveis

código corrigido

```
#include<stdio.h>
#include<string.h>
#include<stdlib.h>
auto int x=0;
int main()
{
    int number = 5;
    {
        int number = 20;
        printf("inner number: %d", number);
    }
    printf("\n");
    printf("outer number: %d", number);
    return 0;
}
```

```
#include <stdio.h>

int main()
{
    auto int number = 5;
    {
        auto int number = 20;
        printf("inner number: %d", number);
    }
    printf("\n");
    printf("outer number: %d", number);
    return 0;
}
```

run

```
inner number: 20
outer number: 5

...Program finished with exit code 0
Press ENTER to exit console.
```

A register Storage Class

A register storage class é usada para definir variáveis locais que devem ser armazenadas em um registro em vez de na RAM.

```
{  
    register int miles;  
}
```

A variável tem um tamanho máximo igual ao tamanho do registro (geralmente uma palavra) e não pode ter o operador unário '&' aplicado a ela (pois não tem uma localização na memória).

```
/* store integer variable "i" in RAM, register, or other location as compiler sees fit */  
int i;  
  
/* suggests storing integer variable "i" in a CPU register or other fast location */  
register int i;
```

O registro deve ser usado apenas para variáveis que requerem acesso rápido, como contadores.

O definir 'registro' não significa que a variável será armazenada em um registro. Isso significa que PODE ser armazenado em um registro dependendo do hardware e das restrições de implementação.

A static Storage Class

A static storage class instrui o compilador a manter uma variável local existente durante o tempo de vida do programa, em vez de criá-la e destruí-la cada vez que entra e sai do escopo (bloco de código onde ela é válida, onde é definida).

Portanto, tornar as variáveis locais static permite que elas mantenham seus valores entre as chamadas de função.

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

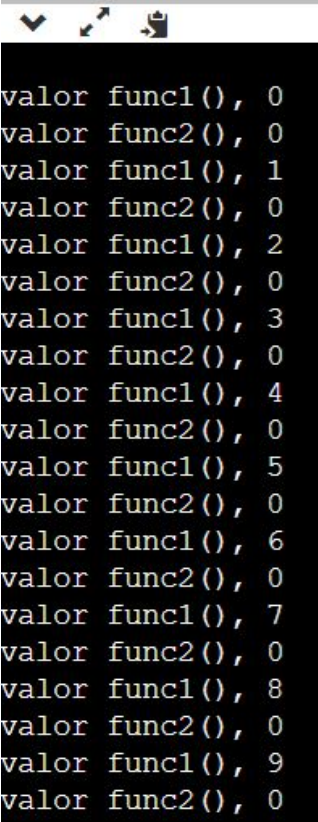
```
int fun1();
int fun2();
int main(){
    for(int i=0;i<10;i++){
        printf("\nvalor func1(), %d",fun1());
        printf("\nvalor func2(), %d",fun2());
    }
}
```

esta linha onde static
está declara só é
executada uma vez

```
int fun1(){
    static int count1=0;
    return count1++;
}
```

```
int fun2(){
    int count2=0;
    return count2++;
}
```

output



```
valor func1(), 0
valor func2(), 0
valor func1(), 1
valor func2(), 0
valor func1(), 2
valor func2(), 0
valor func1(), 3
valor func2(), 0
valor func1(), 4
valor func2(), 0
valor func1(), 5
valor func2(), 0
valor func1(), 6
valor func2(), 0
valor func1(), 7
valor func2(), 0
valor func1(), 8
valor func2(), 0
valor func1(), 9
valor func2(), 0
```

A extern Storage Class

A extern storage class é usada para fornecer uma referência de uma variável global que é visível para TODOS os arquivos de programa.

Quando se usa 'extern', a variável não pode ser inicializada, no entanto, mas aponta o nome da variável em um local de armazenamento que foi definido anteriormente.

The extern modifier é mais comumente usado quando há dois ou mais arquivos partilhando as mesmas variáveis globais ou funções.

First File: main.c

```
#include <stdio.h>

int count ;
extern void write_extern();

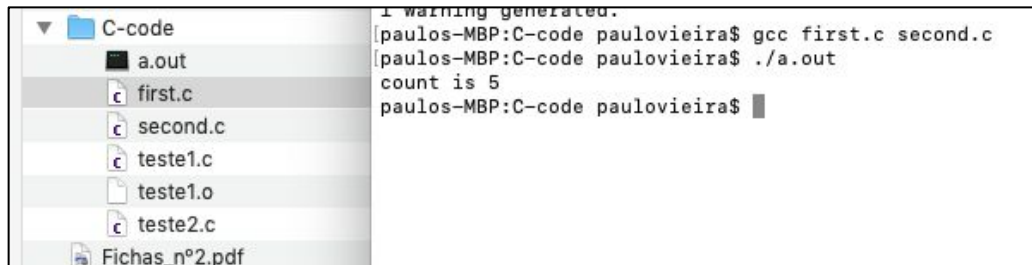
main() {
    count = 5;
    write_extern();
}
```

Second File: support.c

```
#include <stdio.h>

extern int count;

void write_extern(void) {
    printf("count is %d\n", count);
}
```



The screenshot shows a terminal window with a file explorer on the left. The file explorer displays a directory named 'C-code' containing files: 'a.out', 'first.c', 'second.c', 'teste1.c', 'teste1.o', and 'teste2.c'. The terminal window on the right shows the following commands and output:

```
I warning generated.
paulos-MBP:C-code paulovieira$ gcc first.c second.c
paulos-MBP:C-code paulovieira$ ./a.out
count is 5
paulos-MBP:C-code paulovieira$
```

Recursividade

recursividade ocorre quando uma função se chama si própria directa ou indirectamente

A recursividade indireta acontece quando uma função chama outra que por sua vez chama a primeira função

Exemplo 1: recursividade, x!

```
#include <stdio.h>
#include <math.h>
unsigned long fact(int n);

int main()
{
    int x;
    printf("Introduza um numero:");
    scanf("%d",&x);
    printf("%ld",fact(x));

    return 0;
}

unsigned long fact(int n){
    if(n>1) return n*fact(n-1);
    else return 1;
}
```


Recursividade


Exemplo 2

```
#include <stdio.h>
#include <math.h>
int mdc(int n1,int n2);
```

Recursivo, mdc

```
int main()
{
    int z=0,x,y;
    printf("Introduza dois valores:");
    scanf("%d %d",&x,&y);
    if(y>x) z=mdc(y,x%y);
    else z=mdc(x,y%x);
    printf("o mdc entre %d e %d é %d",x,y,z);
    return 0;
}
```

```
int mdc(int n1,int n2){
    return (n2>0? mdc(n2,n1%n2): n1);
}
```



```
if(n2>0) return mdc(n2,n1%n2);
else return n;
```

```
#include <stdio.h>
#include <math.h>
int mdc(int n1,int n2);
```

Não recursivo,
mdc

```
int main()
{
    int z=0,x,y;
    printf("Introduza dois valores:");
    scanf("%d %d",&x,&y);
    if(y>x) z=mdc(y,x%y);
    else z=mdc(x,y%x);
    printf("o mdc entre %d e %d é %d",x,y,z);
    return 0;
}
```

```
int mdc(int n1,int n2){
    int temp;
    while(n2>0){
        temp=n2;
        n2=n1%n2;
        n1=temp;
    }
    return n1;
}
```

Recursividade. Exemplo 2 (o factorial de um número)

```
#include <stdio.h>
#include <math.h>
unsigned long fact(int n);

int main()
{
    int x;
    printf("Introduza um numero:");
    scanf("%d",&x);
    printf("%ld",fact(x));

    return 0;
}

unsigned long fact(int n){
    if(n>1) return n*fact(n-1);
    else return 1;
}
```

FIM