Introdução à Programação

Funções e Escopo de Variáveis

Funções em C

Funções constituem programas em C

Tópicos da Aula

- Hoje vamos detalhar funções em C
  - Importância
  - Escrevendo funções
  - Comando ***return***
  - Passagem de argumentos por valor
  - Execução de uma função
- Depois iremos discutir o escopo de variáveis
  - Conceito de escopo
  - Diferentes tipos de variáveis
    - Local, global e estática

Importância de Funções para Desenvolver Programas

- Desenvolvimento de uma solução se torna mais fácil quando quebramos esta solução em módulos gerenciáveis
- Quando estamos programando, podemos desenvolver módulos separados, onde cada um é responsável por uma certa parte da solução
  - Programação Modular
- Em C, um módulo pode representado por uma **função** ou um **grupo de funções** logicamente relacionadas

Funções em C

Relembrando...

Uma **função** é um conjunto de instruções para realizar uma ou mais tarefas que são agrupadas em uma mesma unidade e que pode ser referenciada por um nome único

- Permitem a construção de programas constituídos de módulos independentes
- São trechos de programas que podem ser ativados
  - Pelo programa principal (função *main*)
  - Por outra função

Estrutura de uma Função

Tipo retornado

Nome

Lista de parâmetros (pode ser vazia)

```
int multiplicacao (int p1, int p2)
{
        int produto;
        produto = p1 * p2;
        return produto;
}
```

Corpo da função

Omitindo o Tipo Retornado

```
multiplicacao (int p1, int p2)
{
      int produto;
      produto = p1 * p2;
      return produto;
}
```

Em C, caso o tipo retornado seja omitido, o compilador assume que a função retorna o tipo **int**

Omitindo o Tipo Retornado

```
multiplicacaoReal (float p1, float p2)
{
      float produto;
      produto = p1 * p2;
      return produto;
}
```

Não vai retornar uma multiplicação de números reais!

Evitar omitir tipo retornado ➔ **dificulta entendimento**

Retorno de uma Função

- Uma função em C pode retornar algum valor, assim como acontece com funções matemáticas
  - Inteiro, real, caractere, etc
- Porém, uma função **não precisa** necessariamente retornar um valor
  - Quando não retorna um valor, dizemos que a função é do tipo ***void***

Funções que Retornam Valores

```
int segundos(int hora, int min) {
      return 60 *(min + hora*60);
}

double porcetagem(double val,double tx) {
      double valor = val*tx/100;
      return valor;
}
```

Funções que retornam valores como resultado usam o comando return

Comando `return`

`return expressão`

- Uma função que não tem valor para retornar tem o tipo de retorno ***void***
  - Neste caso, o uso do comando *return* é opcional

```
void imprimir(int valor) {
   printf("%d",valor);
   return;
}
```

Pode ser omitido

- Para executar este comando o computador:
  - avalia **`expressão`**, obtendo um valor
  - devolve este valor como resultado, **terminando** a execução da função no qual ele se encontra

Considerações sobre Funções

- Uma definição de uma função especifica a seqüência de instruções que serão executadas (**fluxo de controle**) quando esta função for chamada (invocada)

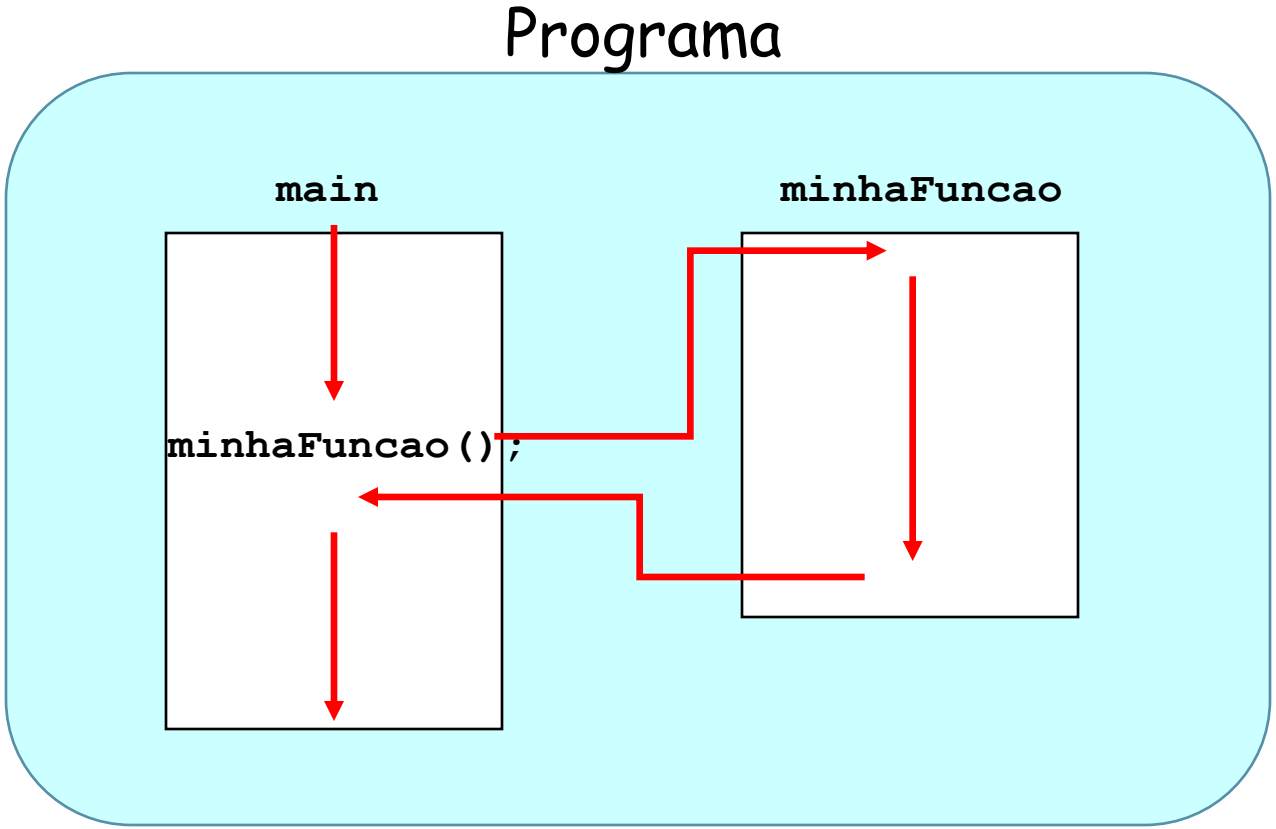
- Quando uma função é chamada, o fluxo de controle do programa pula para a função e executa o código que está nela

Considerações sobre Funções

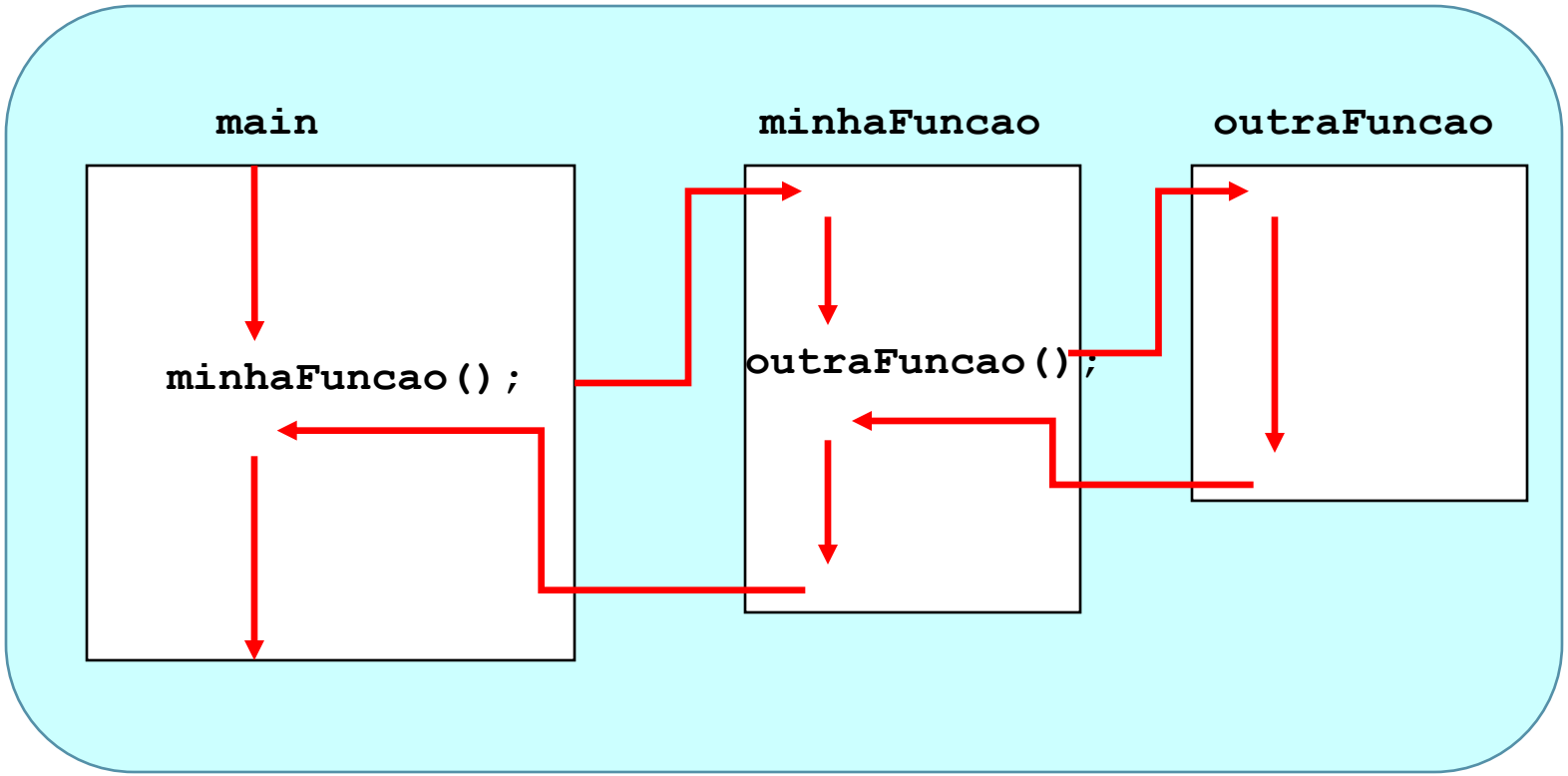
- Quando a função termina de ser executada, o fluxo de controle do programa volta para a instrução logo após a chamada da função

- Uma chamada a uma função pode ou não retornar um valor
  - Depende da definição da função

Fluxo de Controle de uma Função



Fluxo de Controle de uma Função



Parâmetros e argumentos

- Os **parâmetros** são nomes que aparecem na **declaração** de uma função

```
void imprimir(int valor)
```

- Os **argumentos** são expressões que aparecem na expressão de **invocação** da função

```
imprimir(10);
```

Parâmetros de uma Função

- Quando uma função é chamada por outra, os argumentos da chamada são copiados para os parâmetros (formais) presentes no assinatura da função

```
int main() {
  float valor = media (30,40);
}
```

```
float media (float num1, float num2)
{
   float result = (num1 + num2)/2;
   return result;
}
```

Passagem de Argumentos

C permite a **passagem de argumento por valor**: o valor da expressão é avaliado primeiro e depois passado para a função chamada

Passagem de Argumento por Valor

```
void incrementa(int x)   {
   x = x + 1;
   printf("%d",x);
}
```

Comunicação de dados entre funções é feita através de passagem de argumentos

```
int main()  {
      int y = 1;
      printf("%d",y);
      incrementa(y);
      printf("%d",y);
      ...
}
```

saída:1

saída:2

saída:1

não altera o valor de y

Ordem de Definição das Funções

Onde uma função deve ser definida?

- Antes da função ***main***

  OU

- Depois da função ***main*** desde que se declare sua assinatura antes da ***main***

Definindo a Função Antes da *main*

```
#include <stdio.h>

int segundos(int hora, int min) {
    return 60 *(min + hora*60);
}

int main()  {
    int  minutos,hora, seg ;
    printf("Digite a hora:minutos\n");
    scanf ("%d:%d",&hora,&minutos) ;
    seg = segundos(hora,minutos);
    printf("\n%d:%d tem %d segundos.",hora,minutos,seg);
    return 0 ;
}
```

Definição antes da main

Definindo a Função Depois da *main*

```
#include <stdio.h>

int segundos(int hora, int min);

int main()  {
    int  minutos,hora, seg ;
    printf("Digite a hora:minutos\n");
    scanf ("%d:%d",&hora,&minutos) ;
    seg = segundos(hora,minutos);
    printf("\n%d:%d tem %d segundos.",hora,minutos,seg);
    return 0 ;
}

int segundos(int hora, int min) {
    return 60 *(min + horas*60);
}
```

Deve-se declarar antes a assinatura da função - Modo alternativo
`int segundos (int,int)`

Definição depois da main

Escopo de Variáveis

- O escopo de uma variável define a área do programa onde esta variável pode ser referenciada
- Variáveis que são declaradas fora das funções (inclusive da função *main*), podem ser referenciadas por todas as funções do programa
  - São chamadas de **variáveis globais**
- Variáveis que são declaradas dentro de uma função só podem ser referenciadas dentro desta função
  - São chamadas de **variáveis locais**

Escopo de Variáveis

- Pode existir uma variável local a uma função com mesmo nome e tipo de uma variável global, neste caso ao se referir ao nome da variável dentro da função, estar-se-á acessando a variável local

```
#include <stdio.h>

int   numero = 10;

int main(){

      int numero = 4;

      printf("%d", numero);
}
```

Declaração de variável local

Referência à variável local

Será impresso o valor 4

Variáveis Globais

- Podem ser usadas em qualquer parte do código
- Se não inicializadas explicitamente, **C inicializa com valores padrões**
  - 0 para tipos numéricos
- Existem durante todo o ciclo de vida do programa (ocupando memória)

Variáveis Globais

- Normalmente são declaradas no início do programa ou em arquivos do tipo header (*.h)
- São declaradas uma única vez
- Deve-se evitar o uso abusivo delas
  - Pode penalizar o consumo de memória
  - Pode dificultar a legibilidade do código

Uso de Variáveis Globais

```
#include <stdio.h>

int minutos,hora;

int segundos() {
    return 60 *(minutos + horas*60);
}

int main()   {
    int  seg ;
    printf("Digite a hora:minutos\n");
    scanf ("%d:%d",&hora,&minutos) ;
    seg = segundos();
    printf("\n%d:%d tem %d segundos.",hora,minutos,seg);
    return 0 ;
}
```

Todas as funções "enxergam" as variáveis minutos e hora

Comunicação de dados entre funções agora é feita através de variáveis globais

Variáveis Locais

Têm a mesma capacidade de armazenamento que as variáveis globais mas

- São declaradas dentro de uma função
- Só existem durante a execução da função
  - Não ocupam a memória durante toda a execução do programa
- Não são inicializadas automaticamente
- Só são visíveis dentro da função
  - Outras funções não podem referenciá-las

Variáveis Locais

- Caso uma função declare uma variável local, esta é **criada a cada execução** da função

```
int funcao( )
{
   int a= 100;
   a = a + 23;
   return a;

}
```

Sempre retorna 123

Modificador *static*

- Caso a variável local venha com o modificador static, a variável é criada uma única vez
  - Armazena seu valor em várias execuções da mesma função
  - Evita uso de variáveis globais

```
int funcao( )
{
   static int a= 100;
   a = a + 23;
   return a;

}
```

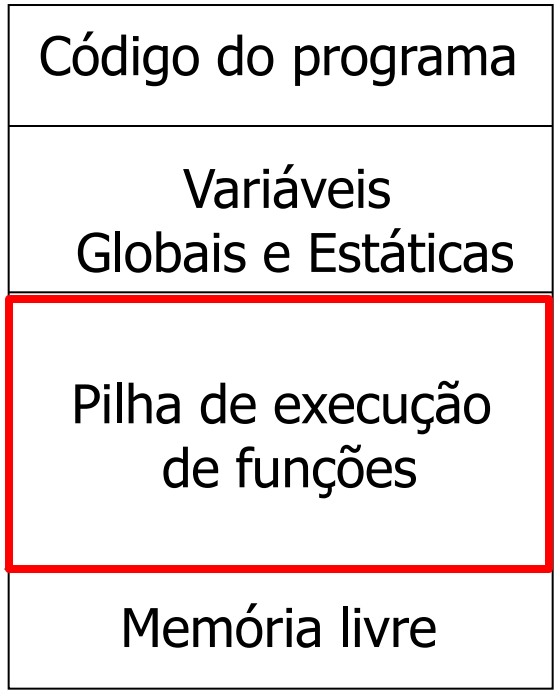
Inicializa apenas uma vez

1ª vez que função for chamada retorna 123

2ª vez retorna 146

Armazenamento das Variáveis

Onde são armazenados na memória os diferentes tipos de variáveis?



Pilha de execução armazena variáveis locais das funções

Quando acaba a execução da função, espaço ocupado pelas suas variáveis é liberado

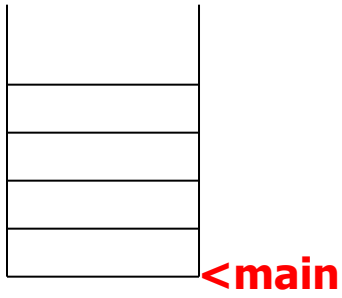
Pilha de Execução de Funções

Considere o seguinte código:

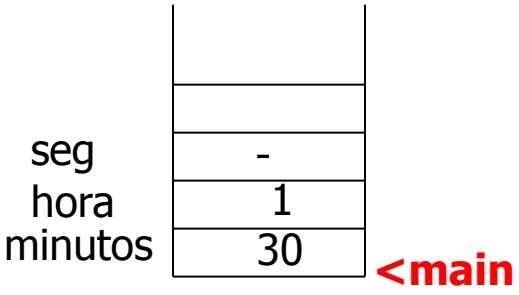
```
#include <stdio.h>

int segundos(int hora, int min) {
    int seg;
    seg = 60 *(min + hora*60);
    return seg;
}

int main()  {
    int  minutos = 30,hora = 1, seg ;
    seg = segundos(hora,minutos);
    printf("\n%d:%d tem %d segundos.",hora,minutos,seg);
    return 0 ;
}
```

Pilha de Execução de Funções

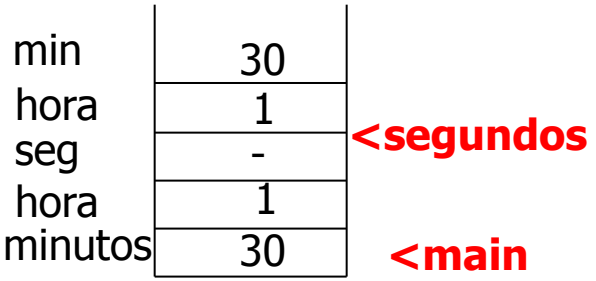
1 – Início do programa: pilha vazia



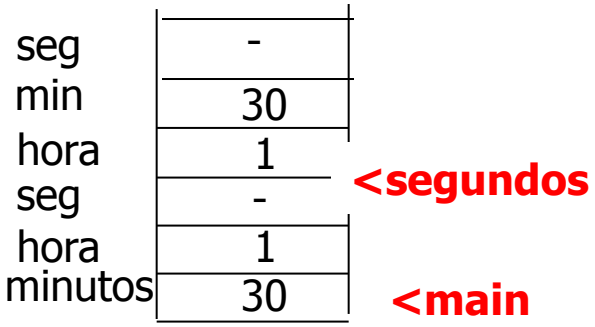
2 – Declaração de variáveis: minutos, hora,seg



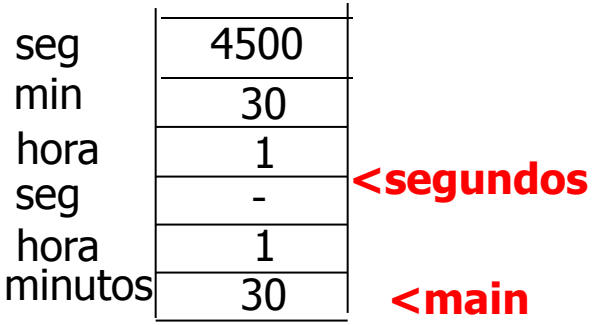
3 – Chamada da função: cópia do argumento



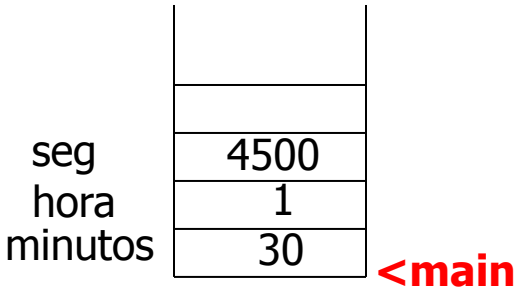
4 – Declaração da variável local: seg



5 – Final da avaliação da expressão



6 – Retorno da função: desempilha



Macros Semelhantes a Funções

- Pré-processador e Macros
  - Diretiva de Definição com Parâmetros
    - São chamadas de Macros
    - Exemplo

```
#define MAX(a,b) ((a) > (b) ? (a):(b))

int main() {
 float v =  4.5 ;
 float c =  MAX(v,3.0) ;
```

O compilador verá:

```
int main() {
 float v = 4.5 ;
 float c = ((v)>(3.0)?(v):(3.0)) ;
```

Macros Semelhantes a Funções

- Pré-processador e Macros
  - Macros definidas incorretamente
  - Uso de macros deve ser feito com cautela!

```
#define DIF(a,b) a - b
int main(){
  printf("%d",4 * DIF(5,3));
  return  0 ;
}
```

Saída é 17 e não 8

```
#define PROD(a,b)( a * b )
int main(){
  printf("%d",PROD(3 + 4,2));
  return 0;
}
```

Saída é 11 e não 14

Resumindo ...

- Funções
  - Importância
  - Escrevendo funções
  - Comando ***return***
  - Passagem de argumentos por valor
  - Execução de uma função
- Escopo de variáveis
  - Conceito de escopo
  - Diferentes tipos de variáveis
    - Local, global, estática