Estrutura de Dados

Professor: Ricardo Bittencourt Socreppa

Curso: Análise e Desenvolvimento de Sistemas

Funções

Funções

- É um **bloco de código** (sequência de comandos) que pode ser **nomeado** e **chamado** dentro de **um programa**

- Pode ser executado quantas vezes forem necessárias durante a execução do programa.

- A linguagem C já possui algumas funções implementadas:
  - Scanf()
  - Printf()

Definição

- **Razões** para Utilizar Funções:
  - Estruturação dos programas
  - Reutilização de código

- **Estruturação dos programas**
  - São pequenos blocos de código cada um com uma função específica

- **Reutilização de código**
  - A função pode ser utilizada diversas vezes em diverentes lugares

Declaração de uma Função

- **Forma Geral:**

```
tipo_retornado nome_funcao (lista de parametros) {
    sequência de comandos
}
```

- A declaração de ser feita **antes** do código no qual é utilizada, ou seja, **antes** da clausula **main();**

- Pode ser declarada **depois** da **main()** neste caso precisamos declarar o **protótipo** da **função antes** da **main()**

```
tipo_retornado nome_funcao (lista de parametros);
```

Declaração antes da main()

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

int quadrado(int a) {
    return a * a;
}

int main() {
    int n1, n2;
    printf("Digite um numero: ");
    scanf("%d",&n1);
    n2 = quadrado(n1);
    printf("O quadrado de %d eh %d\n",n1,n2);
    system("pause");
    return 0;
}
```

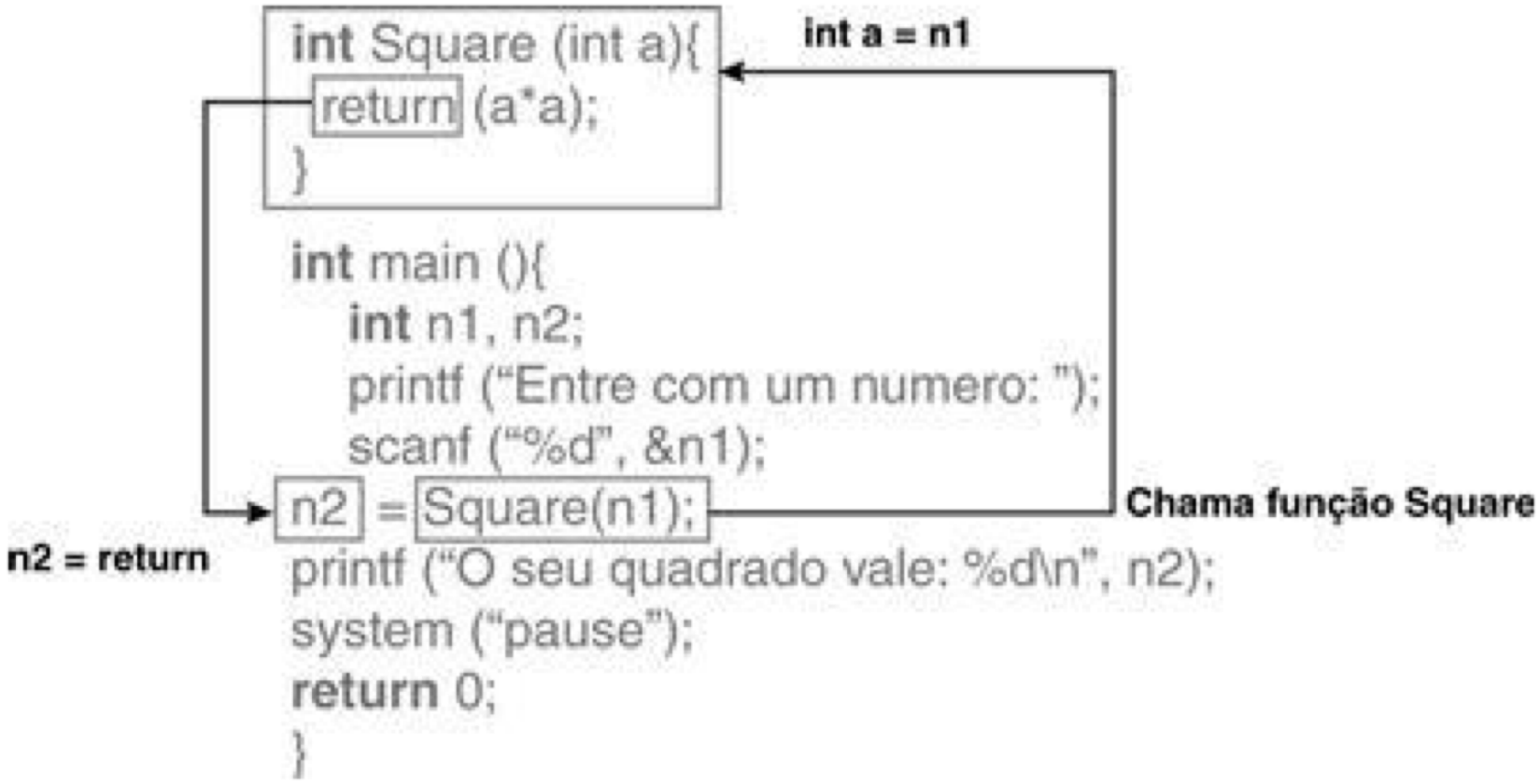
Declaração depois da main()

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

// protótipo da função
int quadrado(int a);

int main() {
    int n1, n2;
    printf("Digite um numero: ");
    scanf("%d",&n1);
    n2 = quadrado(n1);
    printf("O quadrado de %d eh %d\n",n1,n2);
    system("pause");
    return 0;
}
int quadrado(int a) {
    return a * a;
}
```

Funcionamento da Função

- Independente da função ela segue os **seguintes passos**:
  - O código do programa é executado **até encontrar** a chamada da **função**;
  - O **programa** é **interrompido temporariamente**, para executar a função
  - Se **houver parâmetro** na função os **valores são copiados** para ela.
  - **Comandos** da **função** são **executados**
  - Quando a **função termina** ou **encontra** a palavra **return**, o **programa volta** ao **ponto** que foi **interrompido**
  - Se houver um **valor** no **return** ele volta para a **variável** que foi escolhida para **receber** o **valor**;

Exemplo


int Square (int a){
return (a*a);
}
int a = n1
int main (){
int n1, n2;
printf ("Entre com um numero: ");
scanf ("%d", &n1);
n2 = Square(n1);
Chama função Square
n2 = return
printf ("O seu quadrado vale: %d\n", n2);
system ("pause");
return 0;
}

Parâmetros de uma função

- Basicamente são **valores** que **passados** com o **objetivo** de ser **utilizado** dentro da **função**

- O parâmetros devem ser **separados** por **“,”**

- **Forma Geral:**

```
tipo_retornado nome_funcao (tipo nome, tipo nome2, tipo nome3, …) {
    sequência de comandos
}
```

Funções sem parâmetros

- Dependendo da função, ela pode **não possuir parâmetros**

- **Declaração**:
  - Deixar a lista de parâmetros em branco
    - void imprime()
  - Colocar void entre parênteses
    - void impreme (void)

- Se declarar o **void**, **não** é possível colocar **valor de parâmetro** na **função**

Funções sem parâmetros

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

int imprime(){
    printf("Teste da Funcao \n");
}
    int main() {
    imprime();
    imprime(5);
    imprime(5,'a');
    return 0;
}
```

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

int imprime(void){
    printf("Teste da Funcao \n");
}
    int main() {
    imprime();
    imprime(5);
    imprime(5,'a');
    return 0;
}
```

Retorno da função

- É a **maneira** como a **função devolve** o **resultado** (se existir) para quem a chamou;
- Dado a forma geral de uma função:

```
tipo_retornado nome_funcao (lista de parametros) {
    sequência de comandos
}
```

- O **tipo_retornado** estabelece o tipo do valor que a função vai devolver;
- **Tipos de retorno:**
  - **Básicos**: int, float, char, double, void e ponteiros;
  - **Tipos definidos pelo usuário**: struct, array

Função sem retorno (void)

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

void imprime(int a){
    int i;
    for(i = 0; i <= a; i++){
        printf("Linha %d \n",i);
    }
}

int main() {
    imprime(5);
    return 0;
}
```

Função com retorno

- O comando **return** é utilizado para retorna o valor da função

- **Forma Geral:**

```
return expressao;
```

- A **expressão** deve ser **compatível** com o **tipo de retorno declarado** na **função**

- Quando **chega** no comando **return** a **função** é **encerrada imediatamente**

Função com retorno

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

int soma(int x, int y){
    return x + y;
}

int main() {
    int a,b;
    printf("Digite a:");
    scanf("%d",&a);
    printf("Digite b:");
    scanf("%d",&b);
    printf("A soma eh %d",soma(a,b));
    return 0;
}
```

Tipos de passagem de parâmetros

- Na linguagem C podemos passar 4 tipos de parâmetros em uma função, são elas:
  - Passagem de Valor
  - Passagem por Referência
  - Passagem de Arrays
  - Passagem de Estruturas

Passagem de Valor

- Este tipo de passagem é o **padrão** para todos os **tipos básicos**:
  - int
  - float
  - double
  - Char

- Qualquer **modificação** que a função fizer nos **parâmetros** ocorre **somente dentro** da **função**

Passagem de Valor

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

void soma_mais_um(int n){
    n++;
    printf("Dentro da Funcao %d \n",n);
}
int main() {
    int x = 5;
    printf("Antes da Funcao %d \n",x);
    soma_mais_um(x);
    printf("Depois da Funcao %d \n",x);
    system("pause");
    return 0;
}
```

Passagem por Referência

- É quando o **valor passado** por **parâmetro** a **função modifica** o **valor original passado** a ela.
- Exemplo:
  - scanf();
- Na passagem de parâmetros por referência, **não** se **passam valores** das **variáveis**, **mas** os **endereço** da **variável** na **memória**.
- Para **passar** um **parâmetro** por **referência**, usa-se o **operador "*"** na **frente** do **nome do parâmetro** na declaração da função
- Na **chamada da função**, é necessário utilizar o **operador "&"** na **frente** do **nome da variável** que será passada por referência

Passagem por Referência

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

void soma_mais_um(int *n){
    *n = *n + 1;
    printf("Dentro da Funcao %d \n",*n);
}

int main() {
    int x = 5;
    printf("Antes da Funcao %d \n",x);
    soma_mais_um(&x);
    printf("Depois da Funcao %d \n",x);
    system("pause");
    return 0;
}
```

Passagem por Referência

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

void troca(int *a, int *b){
    int temp;
    temp = *a;
    *a = *b;
    *b = temp;
    printf("Dentro da Funcao %d e %d \n",*a, *b);
}
int main() {
    int x = 2;
    int y = 3;
    printf("Antes da Funcao %d e %d \n",x, y);
    troca(&x,&y);
    printf("Depois da Funcao %d e %d \n",x, y);
    system("pause");
return 0;
}
```

Passagem de Arrays

- Para passar um array como parâmetro, além de **passar o array**, é necessário passar um **segunda variável** informando o **tamanho** do **array** que esta sendo **passado**
- Os array são **sempre** passados por **referência**
- A **passagem** por **referência evita copia desnecessária** de **grande quantidades de dados** para **outras áreas da memória**, afetando o desempenho do programa
- Forma Geral:
  - void imprime (int *m, int n);
  - void imprime (int m[], int n);
  - void imprime (int m[5], int n);
- Na chamada da função **passamos o nome do array sem colchetes**, **sem índice e sem operador “&”**

Passagem de Arrays

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

void imprime(int *n, int m){
    int i;
    for (i =0; i < m; i++){
    printf("%d \n", n[i]); }
}

int main() {
    int v[5] = {1,2,3,4,5};
    imprime(v,5);
    system("pause");
    return 0;
}
```

Passagem de Array Multidimensional

- Para passar um **array multidimensional** é necessário **especificar todas as dimensões exceto** a **primeira**

- Forma Geral:
  - void imprime (int m[][5], int n);
  - void imprime (int m[][12][4], int n);

- Um **array bidimensional (matriz)** pode ser entendido como um **array de arrays**

Passagem de Array Multidimensional

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

void imprime_matriz(int m[][2], int n){
    int i,j;
    for (i =0; i < n; i++){
        for (j =0; j < 2; j++){
            printf("%d \n", m[i][j]);
        }
    }
}
int main() {
    int mat[3][2] = {{1,2},{3,4},{5,6}};
    imprime_matriz(mat,3);
    system("pause");
    return 0;
}
```

Passagem de Estruturas

- Como já vimos anteriormente, estruturas são como **um conjunto de variáveis**

- Pode-se passar por **duas formas distintas**:
  - **Toda a estrutura**
    - Basta declarar na lista de parâmetros a estrutura
  - **Apenas determinados campos da estrutura**
    - Declarando uma variável do mesmo tipo da variável da estrutura

Apenas determinados campos da estrutura

```c
#include <stdio.h>
#include <stdlib.h>
struct ponto {
    int x,y;
};

void imprime(int p){
    printf("Valor = %d\n",p);
}

int main() {
    struct ponto p1 = {10,20};
    imprime(p1.x); imprime(p1.y);
    system("pause");
    return 0;
}
```

Apenas determinados campos da estrutura por referência

```c
#include <stdio.h>
#include <stdlib.h>
struct ponto {
    int x,y;
};
void soma_imprime(int *n){
    *n = *n + 1;
    printf("Valor = %d\n",*n);
}
int main() {
    struct ponto p1 = {10,20};
    soma_imprime(&p1.x);
    soma_imprime(&p1.y);
    system("pause");
    return 0;
}
```

Toda a Estrutura

```c
#include <stdio.h>
#include <stdlib.h>
struct ponto {
    int x,y;
};
void imprime(struct ponto p){
    printf("x = %d\n",p.x);
    printf("y = %d\n",p.y);
}
int main() {
    struct ponto p1 = {10,20};
    imprime(p1);
    system("pause");
    return 0;
}
```

Toda a Estrutura

- Necessário colocar () na variável, por exemplo:
  - (*n)

```
#include <stdio.h>
#include <stdlib.h>
struct ponto {
    int x,y;
};
void atribui(struct ponto *p){
    (*p).x = 10;
    (*p).y = 20; }
int main() {
    struct ponto p1;
    atribui(&p1);
    printf("x = %d\n",p1.x);
    printf("y = %d\n",p1.y);
    system("pause");
    return 0;
}
```

Operador seta

- O operador **seta** (->) **substitui** o uso do **conjunto** dos **operadores “*”** e **“.”** no **acesso** de uma **estrutura passada** por **referência** para uma **função**

```c
#include <stdio.h>
#include <stdlib.h>
struct ponto { int x,y; };
void atribui(struct ponto *p){
    p->x = 10;
    p->y = 20;
}
int main() {
    struct ponto p1;
    atribui(&p1);
    printf("x = %d\n",p1.x);
    printf("y = %d\n",p1.y);
    system("pause");
    return 0;
}
```

Exercícios

1. Escreva uma função que recebe dois números e retorno o maior deles
2. Faça uma função que receba um número inteiro de 1 a 12 e imprima o mês e sua quantidade de dias de acordo com número digitado.
3. Elabore uma função que receba três números inteiros por parâmetro, representando a hora, minuto e segundo. A função deve retornar esse horário convertido em segundos
4. Escreva uma função que recebe um array de 10 elementos e retorno a sua soma.
5. Escreva uma função que recebe um array contado a nota de 10 alunos e retorne a média dos alunos.