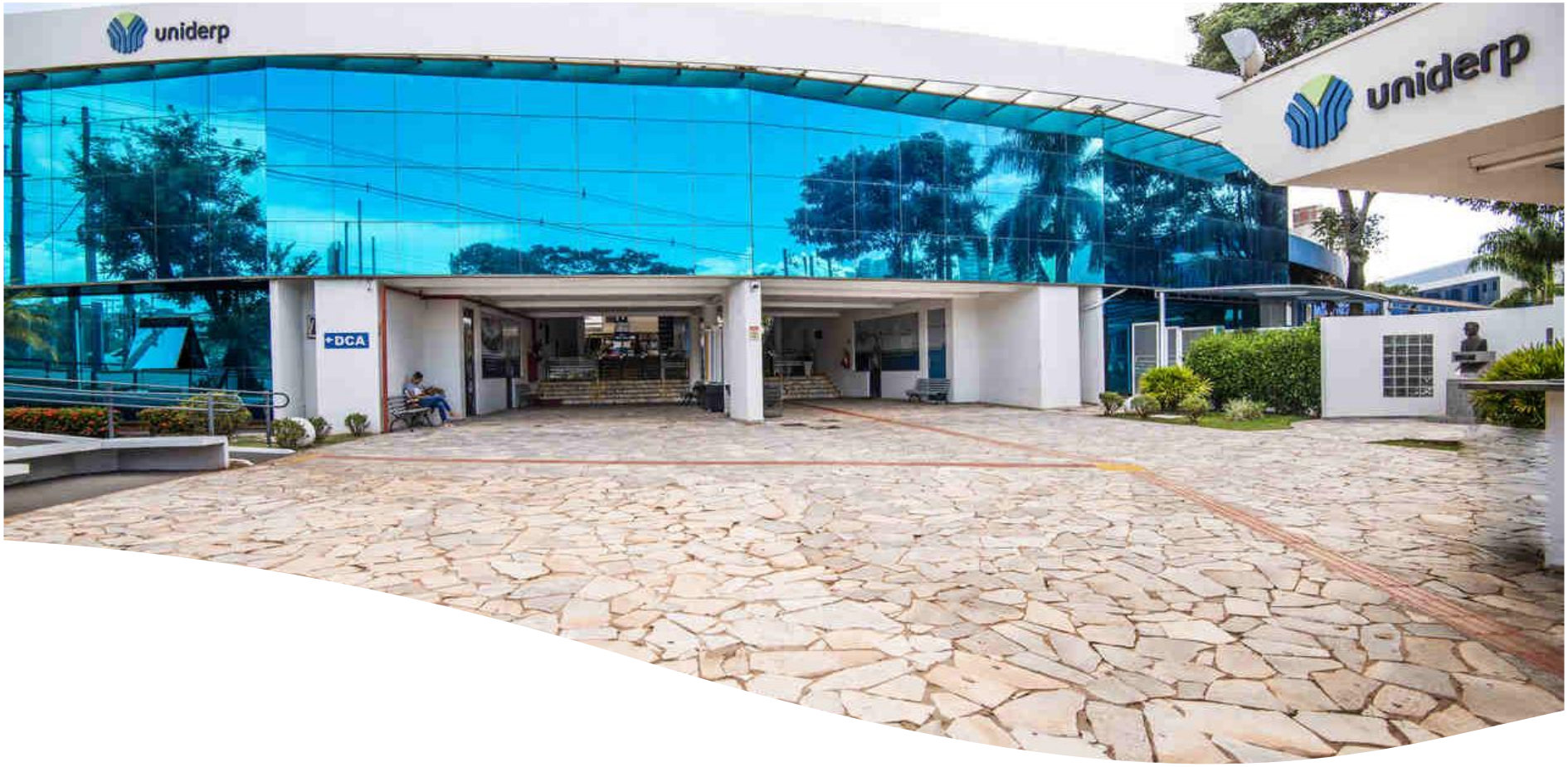

Algoritmos e Estrutura de Dados Avançado

Encontro 03 – Funções

uniderp

Materiais e dúvidas

Nenhum material será enviado via e-mail.

Os materiais serão disponibilizados no **AVA** e no disco virtual: **bit.ly/edauniderp**

Dúvidas, questionamentos, entre outros deverão ser realizados **APENAS** pelo **e-mail** e **AVA.**

Para ingressar no grupo do **Whatsapp** da disciplina acesse o link **linklist.bio/noiza** e selecione sua disciplina.

Introdução

- Problemas grandes são divididos em problemas menores
  - Lógica mais simples e clara
  - Facilita a compreensão
  - Modularização
- Subrotinas
  - Em C são as funções
    - Funções de bibliotecas (printf, scanf, sqrt, ...)
    - Funções definidas pelos programadores

Definição

- “Segmento de programa independente que executa alguma tarefa específica”
- “Conjunto de comandos agrupados em um bloco que recebe um nome e através deste pode ser ativado”.

Vantagens

- Para instruções que são acessadas repetidamente em diferentes pontos do programa (e podem receber dados diferentes)
- Reaproveitamento do código e eliminação de redundância
- Facilita o desenvolvimento e o entendimento (evita a construção de blocos de programas muito grandes)

Vantagens

- Facilita a alteração
- Separa o programa em blocos concisos e que possuem um objetivo bem definido (compreendido de forma isolada)
- Permite a construção de uma biblioteca com as rotinas de uso freqüente

Visão Geral

- Um Programa C é formado por uma ou mais funções
- Função obrigatória: main()
- Uma definição de função não pode estar embutida em uma outra
  - As funções são independentes umas das outras
- Uma função é executada quando acessada
- A mesma função pode ser acessada de diferentes partes do programa

Formato Geral

```
tipo_de_retorno  nome_função  (lista_parâmetros)
{
      <comandos>
}
...
nome_função (lista_parâmetros);
variável = nome_função (lista_parâmetros);
```

- Informação
  - Passada para a função através dos parâmetros
  - Retornada através dos parâmetros ou através do comando return (algumas funções não retornam nada, como a printf)

Primeiro Exemplo (exemplo01)

```c
#include <stdio.h>
#include <locale.h>
#include <conio.h>
void esperaZero( ) {
    char tecla;
    do {
        fflush(stdin);
        tecla = getch();
        if (tecla != '0') {
            printf("Nao é zero\n");
        }
    } while (tecla != '0');
}
int main(){
    setlocale(LC_ALL,"");
    esperaZero(); // Chamada da função definida antes
    // ...........
    esperaZero(); // Chamada da função definida antes
    return 0;
}
```

Parâmetros

- Parâmetros ou argumentos são os valores recebidos e/ou retornados por uma função
- Podem ser divididos em três categorias:
  - Entrada: argumentos que têm seu valores estabelecidos fora da função e não podem ser modificados (valores recebidos)
  - Saída: argumentos que têm seus valores estabelecidos dentro da função (valores calculados)
  - Entrada-Saída: argumentos que têm seus valores estabelecidos fora da função, mas que podem ser modificados por ela (valores modificados)

Parâmetros

- Exemplos
  - printf("Mensagem");
  - scanf("%d", &numero);
  - raiz = sqrt(numero);
- Definição
  - Semelhante a declaração de uma variável
  - Entre os parênteses do cabeçalho da função
  - Quando é necessário mais do que um, são separados por vírgula
  - Passados de acordo com a sua posição
  - Nomes parâmetros formais ≠ Nomes parâmetros reais

Parâmetros (exemplo02)

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

void soma(float a, int b) // parâmetros separados por vírgulas
{
    float result; // declaração de variáveis é igual a função main
    result = a+b;
    printf("A soma de %.2f com %d é %.2f", a, b, result);
}

int main() {
    setlocale(LC_ALL,"");
    int a = 10;
    float b = 12.3;
    soma(b,a); // Chamada da função SOMA(12.3,10)
    return 0;
}
```

Localização das Funções

- Declaração X Definição
  - Declaração: protótipo (tipo + nome + parâmetros)
  - Definição inclui a declaração

Localização das Funções (exemplo03)

- Função deve ser declarada antes de ser usada

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

void soma(float, int);

int main() {
    setlocale(LC_ALL,"");
    int a = 10;
    float b = 12.3;
    soma(b,a);
    return 0;
}

void soma(float a, int b)
{
    float result;
    result = a+b;
    printf("A soma de %.2f com %d é %.2f", a, b, result);
}
```

Escopo de Variáveis

- Bloco de código onde a variável é válida
- Variáveis valem no bloco em que são definidas
- Variáveis definidas dentro de uma função recebem o nome de variáveis locais
- Parâmetros formais são válidos apenas dentro da função

Escopo de Variáveis (exemplo04)

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

void FUNC1(){
    int B = -100;
    printf("Valor de B dentro da função FUNC1: %d\n", B);
}
void FUNC2(){
    int B = -200;
    printf("Valor de B dentro da função FUNC2: %d\n", B);
}

int main() {
    setlocale(LC_ALL,"");
    int B = 10;
    printf("Valor de B: %d\n", B);
    B += 10;
    FUNC1();
    printf("Valor de B: %d\n", B);
    B += 10;
    FUNC2();
    printf("Valor de B: %d\n", B);
    return 0;
}
```

Escopo de Variáveis

- Resultado:

Valor de B: 10

Valor de B dentro da função FUNC1: -100

Valor de B: 20

Valor de B dentro da função FUNC2: -200

Valor de B: 30

Valor de retorno de uma função

- Uma função pode retornar (=devolver) um valor
  - Neste caso deve ser especificado o tipo do valor retornado (<tipo_de_retorno>), que pode ser int, float, etc.
  - O valor de retorno é especificado pelo comando return
  - O valor retornado normalmente é atribuído a uma variável
  - Caso a função não retorne nada, ela deve ser declarada como void
- Exemplos:
  - Função sqrt retorna um double
  - Função getch retorna um int

Passagem de Parâmetros

- Os parâmetros são passados para uma função de acordo com a sua posição
- Os parâmetros formais de uma função se comportam como variáveis locais (criados na entrada e destruídos na saída)
- Podem ser passados argumentos para funções de duas maneiras
  - Passagem por Valor
  - Passagem por Referência

Passagem de Parâmetros

- Passagem por Valor (exemplo05)
  - Copia o valor do argumento no parâmetro formal
  - Alterações feitas nos parâmetros da função não têm efeito nas variáveis usadas para chamá-la

```
1  #include <stdio.h>
2  int multiplica (int x) {
3      x = x * x;
4      return (x);
5  }
6  int main() {
7      int t=10;
8      printf("%d - %d", multiplica(t), t);
9      return 0;
10 }
```

Passagem de Parâmetros

- Passagem por Referência (exemplo06)
  - Não é passado o valor de uma variável, mas sim o seu endereço
  - Alterações feitas nos parâmetros da função afetam as variáveis usadas para chamá-la

```
1  #include <stdio.h>
2  void multiplica (int &x) {
3      x = x * x;
4  }
5  int main() {
6      int t=10;
7      multiplica(t);
8      printf("Valor de t: %d", t);
9      return 0;
10 }
```

Passagem de Parâmetros

- Passagem por Referência (exemplo07)

```c
#include <stdio.h>
// Define que o parâmetro é uma referência à outra variável
void zera(float *a){
    // utiliza o operador de referência para alterar o conteúdo da variável
    *a = 0;
}

int main(){
    float f;
    f = 20.7;
    zera(&f);  // Passa o endereço da variável f para a função
    printf("Valor final: %.2f", f);
}
```

Passagem de Parâmetros

- A passagem de vetores por parâmetro é sempre por referência. Isto significa que não se deve, na chamada da função, passar o endereço do vetor. Isto ocorre pois, por convenção, o nome do vetor já representa o endereço inicial deste na memória.
- No cabeçalho da função que recebe um vetor, há duas formas de definir o parâmetro que é um vetor.
- Na primeira declara-se o vetor como se faz normalmente na linguagem. O exemplo a seguir mostra o uso desta abordagem.

Passagem de Parâmetros (exemplo08)

```c
#include <stdio.h>

void zeraVet(float V[], int qtd){
    int i;
    for(i=0;i<qtd;i++)
       V[i] = 0.0;
}

int main(){
    int i;
    float Vet[10];

    zeraVet(Vet,10);  // Passa o nome do vetor como parâmetro
    for(i=0;i<10;i++)
        printf("%d ", Vet[i]);
    return 0;
}
```

Passagem de Parâmetros

- Na segunda abordagem, pode-se declarar o parâmetro como um ponteiro para o tipo de dado que forma o vetor. No caso do exemplo a seguir, como o vetor foi declarado com um vetor de float, o parâmetro da função deve ser float *V.
- No código da função, o acesso aos dados do vetor é feito da mesma maneira quando se declara o vetor explicitamente (exermplo09).

```
void zeraVet(float *V, int qtd){
    int i;
    for(i=0;i<qtd;i++)
        V[i] = 0.0;
}
```

Exercícios

1) Sabendo que o programa principal lê um número, escreva uma função em C/C++ que mostre na tela a tabuada deste número digitado e retorne ao programa principal o valor da somatória das parcelas da tabuada.

2) Faça um programa que possua uma função que receba dois números a e b, em seguida, faça a troca destes dois números. Dica: a e b devem ser passados por referência e impressos fora da função.

Exercícios

3) Faça um programa que possua uma função que receba os valores a, b e c passados por valor, receba também dois valores x1 e x2 passados por referência. Em seguida, calcule e retorne as duas raízes da equação do segundo grau nas variáveis x1 e x2.

Encerramento

DÚVIDAS, CRÍTICAS E SUGESTÕES, ENVIAR PARA:

noiza@anhanguera.com


uniderp