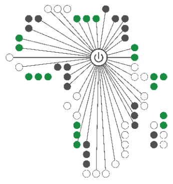

Universidade Virtual Africana

INFORMÁTICA APLICADA: CSI 2303

PROGRAMAÇÃO ESTRUTURADA

Elizabeth Alves Andrade

Prefácio

A Universidade Virtual Africana (AVU) orgulha-se de participar do aumento do acesso à educação nos países africanos através da produção de materiais de aprendizagem de qualidade. Também estamos orgulhosos de contribuir com o conhecimento global, pois nossos Recursos Educacionais Abertos são acessados principalmente de fora do continente africano.

Este módulo foi desenvolvido como parte de um diploma e programa de graduação em Ciências da Computação Aplicada, em colaboração com 18 instituições parceiras africanas de 16 países. Um total de 156 módulos foram desenvolvidos ou traduzidos para garantir disponibilidade em inglês, francês e português. Esses módulos também foram disponibilizados como recursos de educação aberta (OER) em oer.avu.org.

Em nome da Universidade Virtual Africana e nosso patrono, nossas instituições parceiras, o Banco Africano de Desenvolvimento, convido você a usar este módulo em sua instituição, para sua própria educação, compartilhá-lo o mais amplamente possível e participar ativamente da AVU Comunidades de prática de seu interesse. Estamos empenhados em estar na linha de frente do desenvolvimento e compartilhamento de recursos educacionais abertos.

A Universidade Virtual Africana (UVA) é uma Organização Pan-Africana Intergovernamental criada por carta com o mandato de aumentar significativamente o acesso a educação e treinamento superior de qualidade através do uso inovador de tecnologias de comunicação de informação. Uma Carta, que estabelece a UVA como Organização Intergovernamental, foi assinada até agora por dezenove (19) Governos Africanos - Quênia, Senegal, Mauritânia, Mali, Costa do Marfim, Tanzânia, Moçambique, República Democrática do Congo, Benin, Gana, República da Guiné, Burkina Faso, Níger, Sudão do Sul, Sudão, Gâmbia, Guiné-Bissau, Etiópia e Cabo Verde.

As seguintes instituições participaram do Programa de Informática Aplicada: (1) Université d'Abomey Calavi em Benin; (2) Université de Ougagadougou em Burkina Faso; (3) Université Lumière de Bujumbura no Burundi; (4) Universidade de Douala nos Camarões; (5) Universidade de Nouakchott na Mauritânia; (6) Université Gaston Berger no Senegal; (7) Universidade das Ciências, Técnicas e Tecnologias de Bamako no Mali (8) Instituto de Administração e Administração Pública do Gana; (9) Universidade de Ciência e Tecnologia Kwame Nkrumah em Gana; (10) Universidade Kenyatta no Quênia; (11) Universidade Egerton no Quênia; (12) Universidade de Addis Abeba na Etiópia (13) Universidade do Ruanda; (14) Universidade de Dar es Salaam na Tanzânia; (15) Universite Abdou Moumouni de Niamey no Níger; (16) Université Cheikh Anta Diop no Senegal; (17) Universidade Pedagógica em Moçambique; E (18) A Universidade da Gâmbia na Gâmbia.

Bakary Diallo

O Reitor

Universidade Virtual Africana

Créditos de Produção

Autor

Elisabeth Andrade

Par revisor(a)

Arlete Maria Vilanculos Ferrão

UVA - Coordenação Académica

Dr. Marilena Cabral

Coordenador Geral Programa de Informática Aplicada

Prof Tim Mwololo Waema

Coordenador do módulo

Jules Degila

Designers Instrucionais

Elizabeth Mbasu

Benta Ochola

Diana Tuel

Equipa Multimédia

Sidney McGregor

Barry Savala

Edwin Kiprono

Kelvin Muriithi

Victor Oluoch Otieno

Michal Abigael Koyier

Mercy Tabi Ojwang

Josiah Mutsogu

Kefa Murimi

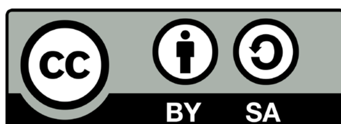
Gerisson Mulongo

Direitos de Autor

Este documento é publicado sob as condições do Creative Commons

Http://en.wikipedia.org/wiki/Creative_Commons

Atribuição http://creativecommons.org/licenses/by/2.5/



O Modelo do Módulo é copyright da Universidade Virtual Africana, licenciado sob uma licença Creative Commons Attribution-ShareAlike 4.0 International. CC-BY, SA

Apoiado por

Projeto Multinacional II da UVA financiado pelo Banco Africano de Desenvolvimento.

Índice

Prefácio 2

Créditos de Produção 3

Direitos de Autor 4

Apoiado por 5

Descrição Geral do Curso 10

Pré-requisitos 10

Materiais 11

Objetivos do Curso 11

Unidades 11

Avaliação 12

Calendarização 13

Leituras e outros Recursos 15

Unidade 0. Avaliação Diagnóstica 17

Introdução à Unidade 17

Objetivos da Unidade 17

Avaliação da Unidade 17

Avaliação 18

Unidade 1. Tipos de dados, operadores e expressões, funções de leitura e escrita 19

Introdução à Unidade 19

Objetivos da Unidade 19

Termos-chave 19

Atividades de Aprendizagem 20

Atividade 1.1 - Tipos de dados básicos em C. 20

Introdução 20

Alguns conceitos básicos: 20

Tipo de instruções: 21

Tipos de dados básicos da linguagem C 21

Declaração e utilização de variáveis 22
Tipos de dados - Constantes 22
Caracteres de formatação especial: 23
Conclusão 23
Avaliação 24
Atividade 1.2 - Operadores e expressões em C 25
Introdução 25
Tipos de Operadores 25
Operadores relacionais 26
Avaliação 28
Expressões 28
Conclusão 28
Atividade 1.3 - Manipulação de operações de leitura e escrita 29
Introdução 29
Funções de entrada e saída 29
Avaliação 30
Conclusão 30
Avaliação da Unidade 31
Resumo da Unidade 31
Avaliação 31
Unidade 2. Estruturas de controlo e funções 35
Introdução à Unidade 35
Objetivos da Unidade 35
Termos-chave 35
Atividades de Aprendizagem 36
Atividade 2.1 - Estrutura de Controlo 36
Introdução 36
Comandos de seleção - Instruções condicionais 36
Comandos de repetição- comandos iteração 39
Resumo dos ciclos 42

Operadores de Incremento e decremento 44

Conclusão 45

Avaliação 45

Atividade 2.2 –Funções e procedimentos 47

Introdução 47

CORPO é onde estão as instruções da função. 47

Características de uma Função 49

Variáveis locais e global em uma função 49

Parâmetros e Argumentos 50

Chamada por valor e chamada por referência 51

Programação modular 52

Breve introdução 52

Tipos de dados abstratos 53

Conclusão 53

Avaliação 53

Resumo da Unidade 54

Avaliação da Unidade 55

Avaliação 55

Unidade 3. Vetores, Ponteiros, Strings e carateres, estruturas, união e enumeração 57

Introdução à Unidade 57

Objetivos da Unidade 57

Termos-chave 57

Atividades de Aprendizagem 58

Atividade 3.1 - Vetores e ponteiros 58

Introdução 58

Definição de vetores e ponteiros 58

Declaração 59

Inicialização de um vetor com tamanho implícito 60

Algumas notas sobre vetores 60

Operações sobre ponteiros 63

Operações Incremento 64

Operações decremento 64

Alocação dinâmica de memória 66

Conclusão 68

Avaliação 69

Atividade 3.2 - Correspondências entre vetor e ponteiros, strings e caracteres . 71

Introdução 71

Relação entre vetor e apontador 71

Introdução 71

Vetores multidimensional 73

Definição de constantes 73

Matriz de várias dimensões 74

Strings e carateres 74

Inicialização automática de strings 76

Manipulação de strings em C, funções padrão 77

Avaliação 78

Conclusão 78

Atividade 3.3 - Outros tipos de dados: Estrutura, União, enumeração 80

Introdução 80

O que é uma estrutura 80

Definição de novos nomes aos tipos de dados 84

Tipo de dado - União 84

Tipo de dado - enumeração 85

Conclusão 85

Avaliação 86

Avaliação da Unidade 87

Avaliação 87

Resumo da Unidade 87

Trabalho Laboratorial da unidade 3 89

Laboratório 2 91

Laboratório 3 94

Unidade 4. Ficheiros, Testes e depuração de Programa 98

Introdução à Unidade 98

Objetivos da Unidade 98

Termos-chave 98

Atividades de Aprendizagem 99

Atividade 4.1 - Ficheiros em C 99

Introdução 99

Conceito de ficheiro 99

Abertura de ficheiro em C 100

O ponteiro do ficheiro em C 102

Conclusão 103

Avaliação 103

Atividade 4.2 - Teste e depuração de programa 104

Introdução 104

Teste de programa 104

Depuração/Debugging 104

Conclusão 104

Avaliação 105

Resumo da Unidade 105

Avaliação da Unidade 106

Avaliação 106

Leituras e outros Recursos 107

Trabalho Laboratorial da unidade 4 108

Laboratório de depuração de um programa em C 108

Avaliação do Curso. 115

Avaliação 115

Referências do Curso 120

Descrição Geral do Curso

Bem-vindo(a) a Introdução à programação estruturada

A linguagem de programação C tornou-se uma das linguagens de programação mais importante e popular, principalmente por ser flexível, portátil e pela padronização dos compiladores. Esta linguagem foi criada para desenvolvimento de programas estruturados/modulares fazendo com que o resultado seja mais legível e documentado o que estimula bons hábitos a um programador.

Ao longo dos anos, a enfâse na implementação de programas orientou-se à organização de estruturas de dados cada vez mais complexas e especializadas, sendo assim, no fim da década de 1970, início da década de 1980, Larry Constantine propôs a metodologia de desenvolvimento estruturado de programas, também defensora por Edward Yourdon e Glenford Myers, que consistia na decomposição funcional de um programa num conjunto de módulos bem estruturados, sugeriam a modularidade como uma técnica de programação para implementação de programas computacionais mais complexos.

Portanto, este conceito de programação estruturada desenvolvida na linguagem de programação permite que os programas desenvolvidos na linguagem C podem ser divididos em partes separados e unidos no final do programa.

A modularidade é assim uma estratégia muito poderosa para se lidar com um problema complexo em uma aplicação informática. Esta técnica de programação estruturada permite armazenar e organizar dados com o objetivo de facilitar o seu acesso e as possíveis modificações.

Portanto, este módulo permite-lhe adquirir no geral conhecimentos sobre os tipos de dados, as variáveis, operadores e expressões na programação estruturada utilizando a linguagem C, desenvolvendo e aplicando conceitos como: entrada/saída, estruturas de controlo, vetores e matrizes, strings e carateres, ponteiros e alocação dinâmica de memória, manipulação de ficheiros bem como noções de testes e depuração de um programa em C.

Pré-requisitos

Conhecimentos do curso de princípios de programação desenvolvidos no módulo anterior.

Materiais

Os materiais necessários para completar este curso incluem:

- Computador
- Compilador DEV C++ e compiladores online
- Compiladores móveis
- Livros indicados nas referências do curso

Objetivos do Curso

Após concluir este módulo deverás ser capaz de decompor qualquer problema em passos algorítmicos, traduzi-lo em um código fonte e compilá-lo para obter um programa executável. Este tópico é muito importante porque vai introduzi-lo ao conceito de programação estruturada, o que lhe dará a base para todos os outros paradigmas de programação a serem estudados posteriormente e também no aumento da capacidade de resolução de problemas mais complexos.

Unidades

Unidade 0: Avaliação diagnóstica

Esta unidade permite-lhe verificar os conhecimentos já adquiridos nos módulos de princípios de programação estudados anteriormente e que são necessários recordar antes de iniciares este módulo.

Unidade 1: Tipos de dados, operadores e expressões, funções de leitura e escrita.

Esta unidade apresenta-lhe os conceitos de escrita em linguagem C, operadores e expressões. Além disso, vai introduzi-lo as funções básicas de entradas e saída, nomeadamente a leitura e escrita e os seus formatos, bem como a entrada e saída padrão.

Unidade 2: Estruturas de controlo e funções

Nesta unidade, vais trabalhar a sintaxe de várias instruções de controlo existentes em C, a noção de funções e a estrutura de um programa em C. Breve apresentação sobre programação estruturada.

Unidade 3: Vetores, matrizes, apontadores, strings e caracteres, estruturas, união e enumeração.

Vetores, ponteiros, strings e caracteres serão introduzidas nesta unidade. Tipos de dados compostos a partir de tipos simples serão também detalhados.

Unidade 4: Ficheiros, testes e depuração de programa

Os ficheiros de processamento em geral, os das entradas e saídas serão desenvolvidas nesta unidade que vão ser terminadas com o teste/ técnicas de depuração de um programa.

Avaliação

Avaliações formativas (verificação dos progressos realizados) são incluídas em cada uma das unidades.

Avaliação sumativa (testes e trabalhos finais) são fornecidos no final de cada módulo para o conhecimento das competências e habilidades do módulo.

Avaliação sumativa são administradas a critério da instituição que oferece o curso. A estratégia de avaliação proposta é a seguinte:

1	Trabalhos de casa	20%
2	Projeto em grupo	20%
3	Trabalho Laboratorial	20%
4	Exame final	40%

Calendarização

Unidade	Temas e Atividades	Estimativa do tempo
Unidade 1: Tipos, operadores e expressões, leitura e escrita	Tipos de dados básicos em C: – estrutura geral de um programa – tipos de dados básicos da linguagem C – variáveis de tipos de dados básicos – Conversões de tipos implícito e explícito – Tipos de dados básicos constantes – operadores – tabela de prioridade dos operadores – expressões Manipulação de operações de leitura e escrita	25h
Unidade 2: Instruções de controlo, funções	As instruções de controlo: – instruções de seleção – instruções de repetição As funções em C: – função na linguagem C – variáveis locais e globais – passagem por endereço e passagem por valor – introdução à programação modular	35h

Unidade 3: Vetores, matrizes, ponteiros e cadeias de caracteres, estruturas, união e enumerações	Vetores e ponteiros: – vetor, matrizes e ponteiros (definições e inicialização, matrizes de string/caracteres, matrizes multidimensionais, definições e inicialização de ponteiros, aritméticas de ponteiros) – alocação dinâmica Correspondência entre vetor e ponteiros, strings e caracteres: – relação entre vetor e o ponteiro (matriz unidimensional e matriz bidimensional) – strings (as funções padrão) – Estruturas, União, Enumeração: – estrutura (definição com ou sem nome, utilização dos campos da estrutura, as estruturas e os ponteiros, passagem por endereço de estruturas, redefinição de tipo, operações sobre as estruturas, – união – enumeração	35h
Unidade 4: Ficheiros, testes de programa e depuração	Ficheiros: – criação de um ficheiro – Manipulação de ficheiro a através das funções existentes. – Abertura de um ficheiro – Modo de acesso aos ficheiros – Operações sobre o fecho de ficheiros – Generalização: saída de entradas formatadas – Impressão e leitura de caracteres Testes de programas e depuração: – Detenção de erros – Depuração – Makefiles	25h

Leituras e outros Recursos

As leituras e outros recursos deste curso são:

Livros

- Backes, A. (2013). Linguagem C: Completa e Descomplicada. Campus. Elsevier 1ª edição.
- Cormen, T. H…[et al.] (2002). Algoritmos: Teoria e Prática. Rio de Janeiro. Elsevier 2ª edição.
- Damas, L. (1999). Linguagem C. Rio de Janeiro. FCA 10ª edição.
- Damas, L. (2001). Linguagem C. Lisboa. FCA.
- Damas, L. (2014). Linguagem C. Lisboa. FCA 24ª edição.
- Guerreiro, P. (2006). Elementos de programação com C. FCA.
- Holzner, S. (1991). C Programming. Brady.
- Jamsa, K. (1989). Microsoft C – Secrets, Shortcuts and Solutions, Microsoft.
- Jackson, M. A. (1975). Principles of Program Design. Academic Press, Inc. Orlando, FL, USA.
- Mizrahi, V. V. (s/d). Treinamento em Linguagem C: curso completo, módulo I. São Paulo. Makron Books.
- Pereira, S. L. (s/d). Linguagem C. Disponível em http://www.ime.usp.br/~slago/slago-C.pdf
- Kernighan, B. W. & Ritchie, D. (1988). The C Programming Language -the ANSI edition, Prentice-Hall
- Rocha, A. A (2014). Estruturas de dados e algoritmos em C. Lisboa. FCA 3ª edição.
- Rocha, A. A. (2006). Introdução á programação usando C. FCA.
- Schildt, H. (1995). C – The Complete Reference, McGraw-Hill. 3rd edition.
- Seymour, L. (1986). Data structures. McGraw Hill Companies.
- Schildt., H. (1995). C Completo e Total. São Paulo. São Paulo. Makron Books 3ª edição.
- Tenenbaum, A. A., Langsam, Y., Augenstein. M.J. (1995). Estruturas de dados usando C. São Paulo. Makron Books.
- Vasconcelos, J. B. & Carvalho, V. J. (2005). Algoritmia e estruturas de dados: Programação nas linguagens C e JAVA. Lisboa. Centro Atlântico.

Sebentas/tutorias/Apontamentos Aulas

- Dias, A. M. (2017). Introdução à Programação. FCT Nova Lisboa.
- Endereços e Ponteiros. Disponível em http://www.ime.usp.br/~pf/algoritmos/aulas/pont.html
- Operadores. Site da eletrónica de programação. Disponível em: http://www.li.facens.br/eletronica
- Programação estruturada. Site da eletrónica de programação. Disponível em: http://www.li.facens.br/eletronica
- Sampaio, I. & Sampaio, A. (s/d). Sebenta Linguagem C. ISEP.
- Santos. J. & Baltarejo, P. (2006). Apontamentos de Programação em C/C++. Porto. Departamento de Engenharia Informática. Disponível em: http://www.ebah.pt/content/ABAAAgZ4UAH/sebenta-cpp-03-2006
- Sá, M. R. T (s/d). Apostila de Introdução à Linguagem C, Versão 2.0 Unicamp apostilas. Disponível em: http://www.ufjf.br/petcivil/files/2009/02/Apostila-de-Introdu%C3%A7%C3%A3o-%C3%A0-Linguagem-C.pdf
- Sebenta da disciplina de Introdução à programação. Escola Superior de Tecnologia de Setúbal. 2004-2005.
- Sousa, J. F. (s/d). Introdução Manipulação de arquivos em C Estrutura de Dados II. Disponível em: http://www.ufjf.br/jairo_souza/files/2009/12/3-Arquivos-Manipula%C3%A7%C3%A3o-de-arquivos-em-C.pdf
- Vilela, V. V. (1999). 300 ideias para programar computadores. Brasília.

Unidade 0. Avaliação Diagnóstica

Introdução à Unidade

Esta unidade permite-lhe verificar os conhecimentos que são necessários recordar antes de iniciares este curso. Podes fazer a avaliação da unidade antes de realizar as atividades de aprendizagem para ajudá-lo a refrescar os seus conhecimentos de informática e de princípios de programação anteriormente estudados nos outros módulos.

Objetivos da Unidade

Após a conclusão desta unidade, deverá ser capaz de:

1. Recordar e aplicar os seus conhecimentos básicos de informática e de princípios de programação.

Avaliação da Unidade

Verifique a tua compreensão!

Teste diagnóstico

Instruções

Responda as questões colocadas a seguir, a avaliação tem a duração de 2h.

Critérios de avaliação

A pontuação total é de 20 pontos distribuída em cada uma das questões apresentadas a seguir.

Avaliação

Questões de diagnóstico (20 pontos)

1. O que é um computador?
2. O que é um algoritmo? Apresenta um exemplo clássico?
3. O que entendes por programação?
4. Descreva as fases de programação? Responda à questão através da apresentação de um exemplo prático ou de um esquema com legenda?
5. Faça uma pesquisa na internet e apresenta 4 linguagens de programação encontrada? Faça uma tabela com duas colunas, uma com a descrição de cada linguagem e a outra com a sua respetiva classificação a nível da geração?
6. Qual a importância da função "main ()" num programa em C?
7. Faça uma revisão sobre funções e procedimentos e apresenta as diferenças entre uma função e um procedimento?
8. Escreva a sintaxe de uma função do tipo int?
9. Faça programa utilizando uma função, por exemplo uma função que soma dois números inteiros?
10. Qual a diferença entre vetor (matriz unidimensional) e uma matriz multidimensional?

Unidade 1. Tipos de dados, operadores e expressões, funções de leitura e escrita

Introdução à Unidade

Esta unidade apresenta uma breve introdução sobre a programação estruturada e os conceitos de tipos de dados e operadores e expressões. Para além destes conceitos, vai introduzi-lo as funções básicas de entradas e saídas, nomeadamente as funções de leitura e escrita. Permitirá ainda criar pequenos programas simples na linguagem de programação C.

Objetivos da Unidade

Após a conclusão desta unidade, deverá ser capaz de:

1. Compreender a programação estruturada
2. Definir variável
3. Utilizar diferentes tipos de dados
4. Criar uma expressão/instrução correta em C.
5. Efetuar as operações de entrada e saída.
6. Escrever programas em C.

Termos-chave

Variável: símbolo denotado por um identificador que inclui um valor e um tipo.

Expressão: elemento de sintaxe que combina um conjunto de operadores e operandos e que tem um valor.

Programa: conjunto de operações a serem executadas por uma máquina.

Atividades de Aprendizagem

Atividade 1.1 - Tipos de dados básicos em C

Introdução

Esta atividade apresenta os tipos básicos de dados da linguagem C. Permite-lhe identificar os tipos de dados básicos da linguagem C e definir variáveis e constantes de diferentes tipos.

Detalhes da atividade

Estrutura básica de um programa em C

Antes de iniciar os detalhes desta atividade, propomos apresentar alguns conceitos importantes como variáveis e tipos e dados básicos em C e recordar a estrutura geral de um programa em C. Veja a sintaxe a seguir:

```
#include<stdio.h>

        main()
        {
           instruções;
        }
```

Alguns conceitos básicos:

Apresentação: o formato do texto é livre e não tem nenhum significado para o compilador (é apenas importante para a legibilidade do programa).

Identificadores: eles são feitos a partir do alfabeto 'a-z, A-Z, 0-9, _ "e começa com uma letra ou um sublinhado" _ ").

Case sensitive: maiúsculas são diferentes de minúsculas em C. Por exemplo a variável MEDIA é diferente da variável media, são dois elementos diferentes na linguagem C.

Comentários:

```
// em uma linha
/* em várias linhas */
```

Tipo de instruções:

Uma instrução: é uma única palavra (expressão, chamadas de funções, instruções de controlo) ou uma instrução composta.

Instrução simples: é uma instrução que termina sempre por;

Instruções compostas: instruções que estão contidas em { }

Tipos de dados básicos da linguagem C

void: tipo vazio, usado principalmente para especificar as funções sem argumento ou nenhum retorno.

char: tipo que permite armazenar um único carater. (Ex: 'A'.)

int: armazenam valores que pertencem ao conjunto de números naturais (sem parte fracionária) positivos e negativos. (Ex: 2, -345, +115, 0)

float/double: são utilizadas para armazenar valores numéricos com parte fracionária. São também frequentemente denominadas reais ou de vírgula flutuante, (Ex: 3.14, 0.0000024514, 1.0)

long double: mais recente tipo de dado que pode representar números com partes decimais de grande precisão, se a máquina permitir.

Signed/Unsigned: por defeito uma variável do tipo inteiro admite valores inteiros positivos e negativos. Por exemplo, se um inteiro for armazenado, em 2 bytes os seus valores podem variar entre -32768 e 32768. Caso se deseje que a variável contenha apenas valores positivos, deverá ser declarada com o prefixo unsigned (Ex: int idade // ou unsigned idade // a idade de um indivíduo não pode ser negativa).

Tipo de variável	Número Bytes	Valor mínimo	Valor máximo
int	2	-32 768	32 768
Shot int	2	-32 768	32 768
Long int	4	-2 147 483 648	2 147 483 648
Unsigned int	2	0	65 535
Unsigned short int	2	0	65 535
Unsigned long int	4	0	4 294 967 295

Figura 1: Resumo tipos variáveis e valores: Fonte (Damas, 2014, pág. 50)

Declaração e utilização de variáveis

Conceito

Uma variável é um nome que é dado a uma determinada posição de memória para conter um valor de um determinado tipo. O valor contido numa variável, pode variar ao longo da execução de um programa.

Declaração de uma variável

A definição de uma variável indica ao compilador qual o tipo de dado que fica atribuído ao nome que nós indicamos para essa variável.

Para a sua declaração utilizamos a seguinte sintaxe: variáveis deve-se proceder com tipo de variável e a seguir o nome, conforme a sintaxe a seguir: tipo [var1, var2, ..., varN];

Exemplo:

```
int a, b;      // a e b são variáveis do tipo inteiro
char cha;    // cha é uma variável do tipo char
float soma; // soma é uma variável do tipo float
```

Uma variável pode ser inicializada quando declarada, conforme o exemplo a seguir:

```
int a = 0;
int b = 2 * 15;
char cha = 'a';
```

Variável Local: uma variável é chamada de local se for declarada dentro de uma função ou bloco.

Variável Global: uma variável é chamada de variável global se for declarada fora da função.

Estes conceitos serão reforçados na atividade sobre funções em C.

Tipos de dados - Constantes

Constantes de tipo inteiro: são criados a partir de números e, naturalmente, expressa em base dez (10). Podem também ser expressas em base de oito (octal), quando o primeiro dígito é um 0 ou base de dezasseis (hexadecimal) quando os dois primeiros caracteres são 0x ou 0X.

Tipos sem precisão: 0342 (oct.), 0X0FF (hex.), 123 (déc.), -280 (decimal).

Longo: 120L, 0364l, 0x1faL , etc.

Constantes de tipo real: tem o padrão de tipo double e pode ser seguido de 'f', 'F' ou 'l', 'L' para se referir a um valor do tipo float ou long double.

Notação com casa decimal: 121.34,

Notação exponencial: 12134e-2

Notação do tipo float: 121.34f

Notação do tipo long double: 121.34l

Constantes de tipo string: os caracteres são escritos entre aspas simples. 'U' '5' '' '. Os únicos caracteres imprimíveis que não podem ser representados desta forma são a barra invertida e apóstrofo, que são, respetivamente, designados por \\ e \.

Caracteres de formatação especial:

O símbolo \ é utilizado para retirar o significado especial que um carácter tem.

n nova linha

t tabulação horizontal

f quebra de página

v tabulação vertical

a sinal de alerta

b retorno

Constantes de tipo cadeia de caracteres/string: uma string é uma sequência de caracteres entre aspas (que termina em memória pelo caráter '\ 0').

Exemplo:

"universidade de cabo verde!"

Conclusão

Nesta atividade todos os tipos básicos de dados foram apresentados, os outros tipos de dados mais complexos serão apresentados nas unidades seguintes. Uma pequena avaliação permite-lhe compreender estes conceitos. Vamos exercitar!

Avaliação

1. Declare três variáveis do tipo: inteiro, real e caracter e atribui valores aos mesmos.
2. Apresente uma cadeia de caracteres com formatação (com parágrafos e tabulações).
3. Indique quais das seguintes declarações de variáveis estão corretas:

   y int;

   Int;

   3ab;

   _sim;

   float a, b;

   char ch1=ch2='A'
4. Faça um programa simples em C, onde aplicas estes conceitos, por exemplo um programa que calcula e mostra o perímetro de uma circunferência.

Atividade 1.2 - Operadores e expressões em C

Introdução

Uma vez já visto os tipos de dados básicos, nesta atividade vais realizar as possíveis operações sobre os mesmos. Serão apresentados os operadores e os seus diferentes tipos, precisamos então entender quando devemos utilizar os operandos e expressões em C.

Detalhes da atividade

Tipos de Operadores

Em C, os operadores podem ser classificados de acordo com o número de operandos. Podemos ter operadores aritmético, operadores de incremento e decremento e operadores relacionais.

Operadores aritméticos

A linguagem C possui um conjunto de operadores, alguns são mais usados do que outros, como é o caso dos operadores aritméticos.

São eles:

=	Atribuição
+	Soma
-	Subtração
*	Multiplicação
/	Divisão
%	Resto da divisão

Exemplos:

```
Num=20;    // atribuído o valor 20 à variável num
17%5;      // valor é 2, pois quando dividimos 17 por 5 teremos o resto 2
```

Operadores de incremento e decremento

-- e ++ permitem decrementar e incrementar as variáveis de tipo inteiro (ou ponteiros). Incrementa de 1 seu operando e decrementa de 1 seu operando respetivamente.

p -- é equivalente a p = p - 1

p++ é equivalente a p = p + 1

Exemplos:

Pós-incrementação: y=val++; ⇔ {y=val; val = val + 1; }

Pré-incrementação: y=++val; ⇔ {val = val + 1; y=val;}

Exemplo:

```
n=5;
x=n++;
printf (“x=%d n=%d”, x, n);
```

A saída será x=5 e n=6 porque na primeira linha atribuímos o valor 5 à variável n, na segunda linha o valor n é atribuído a x e depois n é incrementada de 1, tornando seu valor 6.

Precedência

Os operadores de incremento e decremento tem precedência maior do que os operadores aritméticos.

Exemplo:

a*b++

é equivalente a ter:

(a)*b++

Operadores relacionais

Os operadores relacionais são usados para fazer comparações. São eles:

>	Maior
>=	Maior ou igual
<	Menor
<=	Menor ou igual
==	Igualdade
! =	Diferente

Precedência

Os operadores aritméticos têm maior precedência do que os operadores relacionais.

Exemplo:

```
main()
{
Printf ("A resposta e %d", 4+1<4);
}
```

Saída: A resposta é 0.

Tamanho do operador

O operador sizeof atribui o tamanho em bytes da variável. Pode também atribuir o tamanho de um tipo, o tipo deve estar entre parênteses.

sizeof(t) = tamanho da tabela t; sizeof(t) / sizeof(t[0]) = número de elementos de t.

Operador lógico de negação

A negação lógica é usada para negar uma condição, passando de verdadeiro para falso e vice-versa. Em C, uma expressão é falsa se ele retorna o valor é 0, caso contrário retorna o valor 1.

Operadores lógicos (&&, ||)

Os termos em que aparece os operadores são avaliados da esquerda para a direita, e a avaliação é assim verdadeira ou falsa.

Exemplo: para saber de j é maior ou igual a 20 e menor ou igual a 56 devemos escrever:

j >= 20 && j <= 56

Para aprofundar com mais informações sobre este tópico por favor consultar este link: http://juliobattisti.com.br/tutoriais/katiaduarte/cbasico002.asp

Expressões

Uma expressão é um resultado sintaxicamente correto de operadores e operandos. Retorna, portanto, um valor (a instrução, ele determina uma ação para fazer). Qualquer expressão tem pelo menos dois atributos: tipo e valor.

Exemplo: se i é um valor inteiro 10, em seguida 2*i+3 o seu valor inteiro é 23.

Pode-se também se deparar com expressões com efeitos paralelos, que modifica os elementos. Ex: a++ (retorna a e substitui a por a+1), y = 2 (retorna 2 e dá à y lo valor 2).

Conclusão

Nesta atividade foi apresentada os diferentes tipos de operadores, desde os de atribuição, os de incremento e decremento, relacionais e lógicos. Utilizando estes conceitos resolve os seguintes exercícios de aplicação.

Avaliação

1. Quais dos seguintes operadores são aritméticos:
   a) +
   b) &
   c) <
   d) %
   e) i++
2. Os operadores relacionais são usados para:
   a) Combinar valores
   b) Comparar valores
   c) Distinguir diferentes tipos de variáveis
   d) Trocar variáveis por valores lógicos

3. Verdadeiro ou falso?

   a) 1>2

   b) ‘a’<‘b’

   c) 3==2

   d) 3>=2

4. A precedência dos operadores determina qual é o operador:

   a) Mais importante

   b) Usado primeiro

   c) Mais adequado

   d) Que opera em números maiores

Atividade 1.3 - Manipulação de operações de leitura e escrita

Introdução

Esta última atividade fornece as principais funções das entradas e saídas de biblioteca para ler a partir do teclado e escrever na tela. Isso permitirá a criação de primeiros programas em linguagem C.

Detalhes da atividade

Funções de entrada e saída

A biblioteca principal para entradas e saídas elementares (leitura e escrita) é a seguinte:

```
<stdio.h>
```

Para incluir esta biblioteca e acessar essas funções, use a seguinte sintaxe: #include <stdio.h>

Para escrever mensagem no ecrã usa-se a função: printf

Exemplos:

```
printf("i valor : %d", i);

printf("x = %d\t y = %f\n", x, y);

printf("Bom dia a todos!");
```

Para ler informações do teclado usa-se a função: scanf

Exemplo:

```
scanf("%d%d%f", &a, &b, &c);
```

Outras funções e macros comuns

getchar(): retorna o próximo caractere da entrada do ficheiro atual (teclado). Se o final do arquivo for encontrado, o valor EOF é retornado.

char c; c = getchar();

putchar(c): envia o caractere especificado como um parâmetro no ficheiro de saída corrente (écran).

gets(s): lê uma linha a partir da entrada padrão e armazena em uma área reservada de memória, completando por '\0'.

char s[20]; gets(s);

puts(s): grava na saída padrão da linha fornecido parâmetro. O caráter '\ n' é automaticamente adicionado após o último caractere da linha.

Conclusão

As entradas e saídas serão detalhadas nos capítulos seguintes ao serem utilizadas em vários outros exemplos mais completos. Aqui, apenas as funções scanf e printf são focadas, mas serão vistas outras funções para manipular as entradas saídas geralmente no estudo de ficheiros.

Avaliação

1. Escreva programas simples em C que contém a função main(), printf e scanf e que realize as seguintes operações:

    - Escrever a cadeia de caracteres “bomdia”;
    - Escrever o número inteiro 32567 nos formatos decimal, hexadecimal, octal?

2. O seguinte programa apresenta alguns erros. Quais são?

```
main()

{

int a=1; b=2; c=3;

printf(“os números são: %d %d %d\n, a, b, c, d );

}
```

3. Qual o erro de lógica presente neste programa?

```
main()

{

int a,b,c;

printf(“Digite 3 numeros: \n” );

scanf(“%d %d %d”, a, b, c);

printf(“ \n %d %d %d\n, a, b, c);

}
```

Resumo da Unidade

Em C existe um conjunto de operadores desde os aritméticos, aos de incremento e de decremento bem como os lógicos e relacionais. O uso correto dos mesmos dentro de um programa é fundamental para o correto funcionamento dos mesmos.

Esta unidade permitiu ainda usar as funções main(), prinf e scanf, que são as funções básicas e presentes em todos os programas desenvolvidos na linguagem C.

Avaliação da Unidade

Verifique a tua compreensão!

Trabalho de casa 1

Instruções

Responde às questões colocadas, a avaliação tem duração de 2h.

Critérios de Avaliação

A pontuação é indicada nas seguintes perguntas.

Avaliação

Questão 1 (2 pontos)

1. Tendo em conta as seguintes expressões indica a equivalência entre cada uma delas, por exemplo i+= 2 equivale a i=i+2;

   a) X*=y+1; equivale a ------------;

   b) t/=2.5; equivale a -----------;

   c) P%= 5; equivale a ------------;

   d) D- =13 equivale a ------------;

   e) Apresenta um exemplo teu nesta alínea.

Questão 2 (2 pontos)

2. Considerando o seguinte programa a seguir, qual a sua saída?

```
main () {
Int total=0;
Int cont=10;


printf (“Total= %d\n”, total);
total+=cont;
printf (“Total= %d\n”, total);
total+=cont;
printf (“Total= %d\n”, total);
}
```

Questão 3 (3 pontos)

1. Quais são os tipos de valores lidos nas variáveis de y e z (int) tendo em conta a seguinte declaração: scanf (“%d%d” , &y , &z).

2. Tendo em conta o seguinte programa:

```
main ()
{
Int a, b, c;
printf(“ %d\n”, scanf(“%d %d %d”, &a, &b, &c));
}
```

execute - o digitando os seguintes valores:

a) 1 2 3

b) 1 2 a

c) a 3 4

d) 3.2 1 2

Questão 4 (4 pontos)

Associe os seguintes tipos aos correspondentes formatos de leitura e escrita:

Int	%e
float	%ld
char	%f
long int	%d
short int	%hd
short int	%c

Questão 5 (4 pontos)

Considerando que todas as variáveis são do tipo int. Encontre o valor de cada uma delas:

a) x = (2+3) *7;

b) y = (5+1) /2*6;

c) i = j= (2+3) /4;

d) a = 3+6*(b=7/2);

e) c=5+10%4/2;

Questão 6 (5 pontos)

Qual é o valor lógico de a e b (verdadeiro ou falso?) tendo em conta as seguintes expressões:

a = 10 e b = 3.5 a expressão a > b devolve ______?

a = 10 e b = 3.5 a expressão a >= b devolve um ______?

a = 10 e b = 3.5 a expressão a < b devolve um ______?

a = 10 e b = 3.5 a expressão a <= b devolve um ______?

a = 10 e b = 3.5 a expressão a == b devolve um ______?

a = 10 e b = 3.5 a expressão a! = b devolve um ______?

Podes consultar este link para acesso a todas as tabelas sobre operadores:

http://web.ist.utl.pt/ist153068/ficheiros/teoricas/Programacao_I_Cap_4_Operadores_e_regras_de_precedenias.pdf

Unidade 2. Estruturas de controlo e funções

Introdução à Unidade

Nas sessões anteriores, um programa foi analisado como uma sequência de instruções a serem executadas. O controlo de fluxo de um programa pode ser realizado através de instruções que podem ser executadas mediante uma ou mais condições predefinidas.

Assim nesta unidade vai ser apresentada a estrutura de controlo de fluxo da linguagem de programação C incluindo tanto os comandos de seleção e os comandos de repetição. As funções e procedimentos também serão introduzidas nesta unidade bem como a sua utilização em alguns exemplos.

Objetivos da Unidade

Após a conclusão desta unidade, deverás ser capaz de:

1. Escrever programas utilizando o conceito de estrutura de controlo de fluxo
2. Desenvolver os conceitos de funções e procedimentos
3. Escrever programas utilizando funções

Termos-chave

Ciclos: instruções de repetição que permitem a execução, de forma repetitiva de um conjunto de instruções.

Função: permitem estruturar o código de forma de uma forma mais modular aproveitando as potencialidades da programação estruturada.

Procedimento: é uma função que não retorna nada. Ela é definida colocando-se o tipo void como valor retornado.

Estrutura de controlo: são comandos que especificam a ordem em que o processamento é feito.

Programação modular: técnica de programação utilizada para a implementação de aplicações informáticas de médio e grande complexidades.

Atividades de Aprendizagem

Atividade 2.1 - Estrutura de Controlo

Introdução

As atividades para esta unidade vão permitir-lhe conhecer as técnicas de utilização das estruturas de controlo através de criação de programas na linguagem C utilizando os comandos de seleção e de repetição. Estes comandos incluem o if, o switch e os ciclos while, for e do..while. Vamos começar as nossas atividades!

Detalhes da atividade

Os comandos de controlo de fluxo são a essência de qualquer linguagem de programação uma vez que controlam o fluxo da execução de um programa. Estes comandos são importantes em uma linguagem de programação uma vez que especificam a ordem pelo qual o processamento é feito. Nesta unidade vamos abordar os comandos de seleção e os comandos de repetição.

A título de revisão analisa as condições abaixo, o valor numérico e o significado lógico das condições. Considere as variáveis int i = 0, j = 3; temos:

condição	**valor numérico**	**significado lógico**
(i == 0)	1	verdadeiro
(i > j)	0	falso
(i)	0	falso
(j)	3	verdadeiro

Comandos de seleção - Instruções condicionais

1. if else:

A instrução if-else é uma das instruções de controlo de fluxo da linguagem C que permite indicar quais as circunstâncias em que uma determinada instrução ou conjunto de instruções deve ser executada.

Este comando permite fazer uma seleção, a partir de uma ou mais alternativas, a seleção é baseada no valor de uma expressão de controlo.

sintaxe:

```
if (condição)
 instrução 1;
else instrução 2;
```

A condição é avaliada, se o resultado da condição a avaliar for verdadeiro, será executada a instrução 1, se o resultado da condição a avaliar for falso, será executada a instrução 2 (caso existe o else).

Exemplo:

```
1 #include <stdio.h>
2 #include <stdlib.h>
3
4 int main(int argc, char *argv[])
5 {
6
7 int i;
8 printf("Digite um numero: ");
9 scanf("%d",&i);
10 if (i==0)
11         printf("O numero introduzido e zero\n!");
12 else
13 printf("O numero nao e zero\n");
14
15   system("PAUSE");
16   return 0;
17 }
```

Existem casos em que o teste de uma condição não é suficiente para a tomada de uma decisão, neste caso pode ser necessário testar mais do que eu uma condição. Então as instruções a serem utilizadas nestes casos será “if-else encadeados”.

Exemplo:

```
1 #include <stdio.h>
2 #include <stdlib.h>
3
4 int main(int argc, char *argv[])
5 {
6 int i;
7 printf("Digite sua idade: ");
8 scanf("%d",&i);
9 if (i > 70)
10         printf("Estas Velho!\n");
11 else
12         if (i > 21)
13                 printf("Es Adulto\n");
14         else
15                 printf("Es Jovem\n");
16     system("PAUSE");
17     return EXIT_SUCCESS;
18 }
```

2. Switch case:

Este comando adapta-se a situações em que para a tomada de decisões o número de possibilidades a testar é elevada. Portanto, para reduzir esta complexidade usa-se o if-else consecutivo e encadeado.

O comando default tem uso opcional. A sequência de comandos é executada até que o comando break seja encontrado.

sintaxe:

```
switch(expressão)
{
case constante1: instruções 1;
break;
case constante2: instruções 2;
break;
….
default: instruções;
        }
```

Uma expressão é testada sucessivamente para uma lista de expressões que podem ser do tipo int, char, long. A expressão é avaliada, e em seguida o switch compara o resultado da expressão, com o valor de cada constante em cada um dos case.

Como já foi dito o comando default é opcional. No caso do valor da expressão não ser igual a nenhum dos case, nada é executado, terminando o switch. O programa continua na instrução a seguir ao switch.

A instrução break, ao ser encontrada dentro do programa vai permitir parar a execução dentro de switch, continuando o programa na instrução imediatamente a seguir.

Exemplo:

```
1 #include <stdio.h>
2 int main ()
3 {
4   int valor;
5
6   printf ("Digite um valor de 1 a 7: ");
7   scanf("%d", &valor);
8
9   switch ( valor )
10  {
11     case 1 :
12       printf ("Domingo\n");
13     break;
14
15     case 2 :
16       printf ("Segunda\n");
17     break;
18
19     case 3 :
20        printf ("Terça\n");
21     break;
22
23     case 4 :
24        printf ("Quarta\n");
25     break;
26
27     case 5 :
28        printf ("Quinta\n");
29     break;
30
31     case 6 :
32        printf ("Sexta\n");
33     break;
34
35     case 7 :
36        printf ("Sabado\n");
37     break;
38
39     default :
40       printf ("Valor invalido!\n");
41  }
42
43  getch();
44  return 0;
45 }
```

Comandos de repetição- comandos iteração

Os comandos de repetição (ou iteração) especificam a execução de laços (ciclos), isto é, tem a função de fazer com que outros comandos sejam executados zero ou mais vezes. Na linguagem C um laço pode ser determinado através de três comandos como: for, while e do… while.

3. For

A instrução for (ou ciclo for) adapta-se particularmente a situações em que o número de iterações é conhecido à partida.

Sintaxe:

```
For (inicialização; condição; pós-instrução) instrução;
```

Como funciona?

Inicialização: aqui são inicializadas as variáveis presentes no ciclo, e é executada apenas uma vez.

Condição: a condição é avaliada. Se o resultado devolver o valor falso (zero), então o ciclo termina e o programa continua na instrução imediatamente s seguir. E se o resultado da condição devolver o valor verdadeiro, então é executada a instrução.

Instrução: depois de executada a instrução presente no ciclo, é executada a pós instrução, operações de incremento e decremento.

Exemplo 1:

```
#include <stdio.h>
void main(void)
{
float MA, /*média anual de um dado aluno */
ACM, /* acumulador */
MAT; /* media anual da turma */
int CON; /* contador */
ACM = 0;
for(CON = 1; CON <= 50; CON = CON + 1)
{
scanf (“%f”, &MA);
ACM = ACM + MA;
}
MAT = ACM / 50;
printf (“media anual da turma de %d alunos = %2.1f “, 50, MAT);}
```

Exemplo 2:

```
1   #include <stdio.h>
2   #include <stdlib.h>
3
4   main()
5   {
6      int i,x;
7
8      for(i=1; i<=10; i++)
9      {
10        x=i;
11        printf("%d\t", x);
12     }
13     system("PAUSE");
14     return 0;
15  }
16
```

4. While

A instrução While (também chamada ciclo while), executa uma instrução ou bloco de instruções enquanto uma determinada condição for verdadeira.

sintaxe:

```
While(condição)

    Instrução;
```

O seu funcionamento pode ser resumido nos seguintes pontos: a condição é avaliada e se o resultado da avaliação for falso (0 - zero), o ciclo termina e o programa continua na instrução imediatamente a seguir ao while. Se o resultado da avaliação for verdade (diferente de zero), é executada a instrução associada ao while.

```
1  #include <stdio.h>
2  #include <stdlib.h>
3
4
5  main()
6  {
7    int contador=1;
8    while (contador<=100)
9    {
10         printf("%d\n",contador);
11         contador ++;
12         }
13
14        system("PAUSE");
15   return 0;
16 }
```

5. Do... while

A instrução do... while difere dos ciclos anteriores porque o teste da condição é realizada no fim do corpo do ciclo e não antes, como acontece com os ciclos while e For. Desta forma o corpo do ciclo do... while é executado pelo menos uma vez, enquanto nos ciclos while e For pode nunca ser executado caso a condição seja falsa à partida.

Sintaxe:

```
Do
Instruçao;
While    ( condicao)
```

E*xemplo*:

```
1  #include <stdio.h>
2  #include <stdlib.h>
3
4  main()
5  {
6     int num;
7     do{
8     puts("Digite um numero positivo:");
9     scanf("%f",&num);
10 }
11 while(num <= 0.0);
12
13    system("PAUSE");
14    return 0;
15 }
```

Resumo dos ciclos

for(exp1; exp2; exp3)
instrucao
	é equivalente a	exp1;
		`while(exp2)`
		`{`
		`instrução`
		`exp3;`
		`}`

Exemplo:
for (c = 'a'; c <= 'z'; c++)
putchar(c);
	é equivalente a	`c = 'a';`
		`while(c <= 'z')`
		`{`
		`putchar(c);`
		`c = c + 1;`
		`}`

Instrução "Break"

A instrução break, quando aplicada dentro de um ciclo, termina o correspondente ciclo, continuando o programa na instrução imediatamente a seguir a esse ciclo. Esta instrução pode ser utilizada para terminar uma sequência de instruções dentro de um switch ou para terminar também um ciclo.

Instrução "Continue"

A instrução continue dentro de um ciclo permite que a execução de instrução ou bloco de instruções corrente seja terminado, passando à próxima iteração do ciclo.

Nota 1: A instrução continue quando presente dentro de um ciclo, passa o ciclo para a próxima iteração.

Nota 2: A instrução continue, só pode ser utilizada dentro de ciclos enquanto o break pode ser utilizado em ciclos ou na instrução.

Ciclos Encadeados

Ciclos encadeados, são ciclos (while, for ou do... while) que estejam presentes dentro de outros ciclos.

Não existe qualquer limitação ao número de ciclos que pode ocorrer dentro de outros ciclos.

Ciclos Infinitos

Denominam-se por ciclos infinitos, os ciclos que nunca terminam, isto é, apresentam condições que são sempre verdadeiras.

Exemplo:

```
1  #include <stdio.h>
2  #include <conio.h>
3  int main (void)
4  {
5       int n;
6       for (;;)
7       {
8           printf("Digite um numero inteiro: ");
9           scanf("%d", &n);
10          if (n == 7)
11          {
12              printf("Saindo do loop...\n");
13              break; //força a saída do loop
14          }
15          printf("Numero: %d\n",n);
16      }
17      printf("Fim de programa");
18      getch();
19      return(0);
20 }
21
22
```

Nota 1: quando no ciclo for não for colocada qualquer condição, esta é substituída pela condição verdadeira.

Este tipo de ciclo é utilizado normalmente quando não se sabe à partida qual o número de vezes que se vai iterar o ciclo. Para terminar um ciclo infinito usa-se a instrução break ou return.

Operadores de Incremento e decremento

São dois operadores bastante úteis para simplificar expressões:

++ (incremento de 1)

-- (decremento de 1)

Podem ser colocados antes ou depois da variável a modificar. Se inseridos antes, modificam o valor antes da expressão a ser usada e, se inseridos depois, modificam depois do uso.

Alguns exemplos:

```
x = 2;

var = ++x;
```

No caso acima, o valor de var será 3 e o de x será 3.

```
x = 2;

var = x++;
```

No caso acima, o valor de var será 2 e o de x será 3.

Conclusão

Nesta unidade desenvolvemos os comandos de seleção (if else e switch) e os comandos de repetição (for, while e do-while). Também apresentamos as instruções break e continue e os seus respetivos exemplos de utilização bem como um breve apanhado sobre os ciclos encadeados e ciclos infinitos. Vamos a seguir realizar alguns exercícios de aplicação destes conceitos.

Avaliação

1. Identifique os erros no programa seguinte:

```
main()
{
int x[10];
int t;
for(t=0;t<10)
{
x[t]=t*2;
printf(“%d\n”, x[t]) ; }
}
```

2. Tendo em conta o seguinte programa escreva a sua saída correta?

```
1 #include <stdio.h>
2 #include <conio.h>
3
4 void sistema_cor(int cor)
5 {
6 switch (cor)
7 {
8 case 'R': case 'r': printf("\ncor vermelha");
9 break;
10 case 'G': case 'g': printf("\ncor verde");
11 break;
12 case 'B': case 'b': printf("\ncor azul");
13 break;
14 default: printf("\ncor inválida");
15 break;}
16 }
```

a. Explique a função do comando break neste exercício em concreto.

b. Explique a função do comando default neste exercício em concreto.

3. Os códigos das alíneas a) e b) são equivalentes? Justifique a tua resposta?

a)	a)

`#include <stdio.h>`
`main()`
`{`
`int soma=0;`
`int n;`
`for (i=1; i<=10; i++)`
`{`
`scanf ("%d", &n);`
`soma +=n;`
`}`
	`#include <stdio.h>`
	`main()`
	`{`
	`int soma=0;`
	`int n;`
	`while(i<=10)`
	`{`
	`scanf ("%d", &n);`
	`soma +=n;`
	`}`

4. Reescreva o programa abaixo utilizando o ciclo "for" Justifique todos os passos criados com comentários nos códigos?

```
1 #include <stdio.h>
2 main()
3 {
4 char caracter;
5 long int caracteres_lidos=0;
6 scanf("%c", &caracter);
7
8 while( caracter!=EOF) { /* Enquanto carácter lido diferente de
9 caracteres_lidos++; EOF executa ciclo */
10
11 if(caracter>='A' && caracter<='Z')
12 caracter=caracter - 'A' + 'a';
13 printf("%c",caracter);
14 scanf("%c", &caracter);
15 }
16 printf("\nForam lidos %ld caracteres!\n", caracteres_lidos);
17 }
```

Atividade 2.2 –Funções e procedimentos

Introdução

Nesta unidade vais aprender os conceitos sobre as funções em C e identificar as suas aplicabilidades num programa. Da mesma forma é apresentada o conceito de variáveis locais e globais e a passagem de funções por parâmetros e valores.

Detalhes da atividade

Nesta unidade vais aprender a trabalhar com funções em C.

Uma função é um conjunto de comandos agrupados em um bloco que recebe um nome e através deste pode ser ativado. Todo programa em C tem pelo menos uma função chamada main que é obrigatória. A execução do programa inicia pelo primeiro comando da função main.

Sintaxe de uma função:

```
Tipo  Nome (Parametros)
  {
     corpo da funçao  }
```

TIPO é o tipo de valor retornado pela função. Se nada for especificado o compilador considera que será retornado um valor inteiro.

NOME: é o nome da função, tal como o nome de uma variável.

PARAMETROS é a lista das variáveis que recebem os argumentos quando a função é chamada. Deve incluir o tipo e nome de cada variável. Sua sintaxe é:(tipo variável1, tipo variável2, ..., tipo variáveln).

CORPO é onde estão as instruções da função.

Um valor deve ser retornado de uma função que não é declarada como void, da mesma maneira que um valor não pode ser retornado de uma função void. O valor de retorno é especificado pelo comando return, Veja exemplos a seguir.

int f1() {}	// erro nenhum valor retornado
void f2() {}	// correto
int f3() {return 1;}	// correto
void f4() {return 1;}	// erro valor retornado em uma função void
int f5() {return ;}	// erro falta valor de retorno
void f6() {return ;}	// correto

Exemplo de uma Função:

```
1  #include <stdio.h>
2  #include <stdlib.h>
3
4  float media2(float a, float b){
5        float med;
6        med = (a + b) / 2.0;
7        return(med);
8      }
9   main(){
10       float num_1, num_2, med;
11       printf("Entre o primeiro numero:");
12       scanf("%f", &num_1);
13       printf("Entre o segundo numero:");
14       scanf("%f",&num_2);
15       med = media2(num_1, num_2);    // chamada a função
16       printf("A media destes numeros e %.0f\n", med);
17    system("PAUSE");
18    return 0;
19 }
```

Características de uma Função

- Cada função tem que ter um nome único, o qual serve para sua invocação algures no programa que pertence.
- Pode ser invocada a partir de outras funções.
- Uma função (como seu nome indica) deve realizar uma única tarefa bem definida.
- Uma função deve comportar-se como uma caixa negra. Não interessa como funciona, o que interessa é que o resultado final seja o esperado, sem efeitos colaterais.
- O código de uma função deve ser o mais independente possível do resto do programa deve ser tão genérico quanto possível, para poder ser reutilizado noutros projetos.
- Uma função pode receber parâmetros que alteram o seu comportamento de forma a adaptar-se facilmente a situações distintas.

Variáveis locais e global em uma função

As variáveis criadas em uma função são locais, ou seja, podem ser declaradas dentro do corpo de uma função e só são conhecidas centro desta função. Assim serão destruídas após o término da função.

Exemplo:

/* Mantendo o valor de uma variável entre as * chamadas de uma função */

```
1 #include <stdio.h>
2 int soma_1(int a);
3
4 int main()
5   {
6     int nr = 1;
7  printf("Chamando a função a primeira vez: valor + 1 = %d\n",soma_1(nr));
8 printf("Chamando a função pela segunda vez: : valor + 1 = %d\n",soma_1(nr));
9 printf("Chamando a função pela terceira vez: : valor + 1 = %d\n",soma_1(nr));
10    return(0);
11  }
12 int soma_1(int a)
13  {
14    static int valor = 1;
15
16    printf("valor = %d\n",valor);
17    valor = valor + a;
18    return(valor);
19  }
```

Caso uma variável local a função tenha o mesmo nome de uma variável global, a variável local será usada e não a global.

Notas: Depois de terminada a execução de uma determinada função, todas as suas variáveis são destruídas e sempre que possível deve-se recorrer a variáveis locais, evitando-se assim os efeitos paralelos que ocorrem quando se utilizam variáveis globais.

Exemplo:

```
1  #include <stdio.h>
2  int a = 1; /* variável global */
3  void exibe(void)
4    {
5      int a = 10; /* variável local a função exibe() */
6      printf("a dentro da função = %d\n",a);
7    }
8
9  int main()
10   {
11     printf("\n");
12     printf("a dentro de main = %d\n",a);
13     exibe();
14     printf("\n");
15     return(0);
16   }
```

Parâmetros e Argumentos

A comunicação com uma função faz-se através dos argumentos que lhe são enviados e dos parâmetros presentes na função que os recebem.

O número de parâmetros de uma função depende da necessidade do programador. Os parâmetros de uma função devem ser separados por vírgula, cada função necessita assim saber qual o tipo de cada um dos parâmetros a serem utilizados.

Portanto, um parâmetro é uma variável local de uma função.

Exemplo correto:

função (int x, int y, int k, int z)

Exemplo incorreto:

função (int x, y, k, z)

Argumentos

Argumentos são os valores usados para chamar a função e parâmetros são as variáveis, declaradas na definição da função, que recebem estes argumentos.

Exemplo:

```
1 #include <stdio.h>
2 #include <stdlib.h>
3
4 int soma(int a, int b) /* "a" e "b" são os parâmetros da função "soma" */
5   {
6     int resultado;
7     resultado = a + b;
8     return(resultado);
9   }
10
11 int main()
12   {
13     printf("A soma entre 5 e 2 é %d\n",soma(5,2));
14    /* No comando printf acima a função "soma" é chamada
15     * com os argumentos 5 e 2 */
16     return(0);
17   }
```

A correspondência com uma função faz-se através de argumentos que lhe são enviados e dos parâmetros presentes na função que os recebem.
Um parâmetro é uma variável local à função a que pertence, é inicializado com o valor enviado pelo programa que o invoca.

Chamada por valor e chamada por referência

Utilizando a chamada por valor, os valores dos argumentos passados não são modificados. Na realidade é passada uma cópia dos valores para a função.

Na chamada por referência são passados os endereços de memória onde estão os argumentos. Neste tipo de chamada os valores podem ser modificados.

Exemplo chamada por valor:

```
1 #include <stdio.h>
2 #include <stdlib.h>
3 #include <math.h>
4
5 int sqr (int x)
6   {
7     x=x;
8     return(x);
9   }
10 main ()
11 {
12     int t=10;
13     printf("%d  %d", sqr(t), t); // saída do programa: 100   10
14  }
```

Exemplo chamada por referência:

```
1 #include <stdio.h>
2 #include <stdlib.h>
3 #include <math.h>
4
5 void swap (int *x, int *y)
6   {
7      int temp;
8 temp=*x;
9            *x=*y;
10      *y=temp;
11   }
12 main ()
13 {
14      int i,j;
15            i=10;
16            j=20;
17      swap(&i,&j); // passa os endereços de i e j)
18 }
```

Programação modular

Breve introdução

No processo de programação existe algumas técnicas de implementação de programas desde os programas de pequena, média e grande complexidade.

No início da década de 1970, Niklaus Wirth, desenvolveu a técnica de programação procedimental em que esta técnica temos uma estrutura de dados centralizada e o algoritmo é implementado através da definição de novas operações no âmbito da linguagem de programação, através da criação de subprogramas e pelo estabelecimento rigoroso das consequentes dependências de informação.

No início da década de 1980, surgiu uma nova técnica de programação denominada de programação modular, que consistia na decomposição funcional de uma aplicação num conjunto de módulos bem estruturados que cooperam para implementar a funcionalidade pretendida, ou seja, propunham a modularidade como forma de lidar com a complexidade. Assim com esta técnica a solução do problema deixou de ser analisada na perspetiva funcional das operações, para o ser na perspetiva composicional das estruturas de dados.

Tipos de dados abstratos

Um módulo que providencia uma estrutura de dados bem encapsulada, um conjunto coeso de operações que atuam sobre essa estrutura de dados e que é completamente independente de outros módulos, pode ser encarado como um novo tipo de dados da linguagem e designa-se por tipo de dados abstrato – TDA (abstrat data type).

Conclusão

Nesta atividade desenvolvemos o conceito de função como um dos aspetos importantes a ter em consideração no processo de escrita de um programa em linguagem C. Também foi notado que com a sua utilização o programa fica decomposto em partes o que facilita a compreensão e reutilização do código introduzindo assim o conceito de programação modular que facilita ainda o processo de compilação do programa. A passagem por valor e por referencia também foi apresentada e as suas principais utilidades e diferenças dentro de uma função e foi ainda apresentada uma breve introdução à programação modular.

Para mais informações sobre os paradigmas de programação e a evolução no desenvolvimento do programa recomenda-se a leitura do capítulo I do livro “The C++ Programming Language”, 2ª edição, de Bjarne Stroustrup, Editora Addilson- Wesley, 1995.

Avaliação

1. Qual é a saída do seguinte programa?

```
#include<stdio.h>
 struct data
{
  int dia;
  char mes [10];
  int ano;
};
struct Aluno
{
  char nome [30];
  char sexo;
};
```

```
void mostrar ()
{
   int i, j,Aluno;
   char m [2][3] ={{'m','a','r','i','a'},{'M','o','n','t','
e','i','r','o'}};

   for (i=0; i<10; i++)
   for (t=0; t<30; t++)
   if(m[i][j]! =0);
      printf("m[i][j]: %s\n", m);
}
Adicionar_Aluno(m);
```

2. Verifique se um número passado como parâmetro é primo. A função deve retornar 1 se o número for primo e 0 caso contrário. Usando essa função, escreva programa que imprime todos os números primos de 2 a n (n lido).

3. Escreva uma função que recebe como parâmetro um inteiro n e escreva a soma dos seus divisores próprios. Um divisor de um número e próprio se for diferente desse numero e da unidade. Por exemplo, a soma dos divisores próprios de 12 e 2 + 3 + 4 + 6 = 15.

4. Escreva uma função inteira para determinar o ano de ordem duma data dentro do ano respetivo. Queremos saber se essa data representa o 1° dia do ano, o 2° dia do ano, etc. A função tem três argumentos: dia, mês, mais a indicação se o ano é bissexto ou não. Exemplos:

```
ordem(1, 1, false) == 1    // 1 de janeiro dum ano comum
ordem(1, 3, false) == 60   // 1 de março dum ano comum
ordem(1, 3, true) == 61    // 1 de março dum ano bissexto
ordem(31, 12, true) == 366// 31 de dezembro dum ano bissexto
```

Resumo da Unidade

Com a conclusão desta unidade já tens todos os elementos essenciais para criar programas mais completos na linguagem C. As instruções de seleção e de repetição como if-else e switch e os ciclos for, while e do-while vão dar lhe possibilidades de elaborar programas mais completos. A utilização de funções e os conceitos de programação modular vão completar todos os aspetos fundamentais necessários para a programação estruturada. Vamos então praticar!

Avaliação da Unidade

Verifique a sua compreensão!

Trabalho de casa 4

Instruções

Responda com clareza todas as questões a seguir colocadas, a avaliação proposta é para 3 horas aproximadamente.

Critérios de Avaliação

A pontuação total equivale a 20 pontos que será distribuída nas referidas questões de forma equilibrada.

Avaliação

Questão 1 (5 pontos)

1. Passe o seguinte trecho de código para o compilador e comenta todas as linhas de código assinaladas. Verifique qual é o resultado deste programa.

```
1 #include <stdio.h>
2 float max(float a, float b){      //
3 if(a > b){ //
4 return(a);    //
5 }else{ //
6 return(b);     //
7       }}
8     void main(){ //
9 float num1,num2,m; //
10 clrscr(); //
11 puts("*** Valor maximo de dois reais ***"); //
12 puts("Digite dois numeros:");//
13 scanf("%f %f",&num1,&num2);//
14 m = max(num1,num2); //
15 printf("O maior valor e': %f",m);//
16 getch();//
```

Questão 2 (10 pontos)

2. Escreva funções para:

a. $(a+1)^n$ Receber um real a e um natural n e retorna a n - ésima potência de a+1, isto é .

$\frac{1}{n!}\sum_{i=1}^{n} i!$

b. Calcular e retornar o valor de

Questão 3 (2 pontos)

3. Diga qual é a saída do seguinte programa. Explique todas as instruções presentes.

```
float calcula (int v[],
int a)

{

float s = 0;

int i;

for (i=0; i< a; i++)

s = s + v[i];

return s/a;

}
```

Questão 4 (3 pontos)

4. Escreva um programa que utilize uma função para a troca de valores de duas variáveis. Os valores deverão ser introduzidos pelo utilizador no programa principal. Estas variáveis deverão ser do tipo string.

Unidade 3. Vetores, Ponteiros, Strings e carateres, estruturas, união e enumeração

Introdução à Unidade

Depois de termos estudado nas unidades anteriores os tipos de básicos de dados em C (char, int, float e double), bem como as estruturas de controlo (if, switch, while, for e do…while) e ainda a utilização das funções C, nesta unidade, vamos estudar a forma de como podemos processar um conjunto de dados do mesmo tipo por forma a desenvolver programas ainda mais aprimorados. Estes tipos de dados são importantes uma vez que permitem realizar tarefas de resolução de problemas mais complexas, tarefas que os com os outros tipos de dados básicos não são possíveis de solucionar. Vamos assim desenvolver e aplicar os conceitos destes tipos de dados a conhecer: vetores, ponteiros, strings e carateres, estruturas, união e enumeração.

Objetivos da Unidade

Após a conclusão desta unidade, deverá ser capaz de:

1. Escrever um programa utilizando tipos de dados complexos
2. Manipular vetores e strings
3. Desenvolver programas utilizando ponteiros
4. Efetuar uma alocação de memória
5. Utilizar estruturas de dados dinâmicos

Termos-chave

Dados complexos: novos tipos de dados que são definidos pela composição a partir dos dados básicos como (inteiro, real, caracter).

Estrutura: uma estrutura é uma coleção arbitrária de variáveis logicamente relacionadas.

Vetor: conjunto de elementos do mesmo tipo cujos elementos são distribuídos por linhas e colunas e que podem ser encontrados facilmente através dos seus índices.

Ponteiro: um ponteiro é uma referência a uma posição de memória. Ponteiros podem ser constantes ou variáveis e através deles podemos acessar dados ou código.

Alocação: é o processo através do qual o programador reserva o espaço na memória e recupera o endereço do seu armazenamento.

Atividades de Aprendizagem

Atividade 3.1 - Vetores e ponteiros

Introdução

Esta atividade vai introduzi-lo aos tipos de dados estruturados como vetores e ponteiros. Nesta atividade vamos apresentar-lhe os principais conceitos sobre e uso dos mesmos em programas simples utilizando a linguagem C e algumas notas importantes a considerar a quando da utilização dos mesmos.

Detalhes da atividade

Definição de vetores e ponteiros

Um vetor vulgarmente conhecido como array é um conjunto finito de elementos do mesmo tipo, armazenados na memória em endereços contíguos. Os elementos do mesmo tipo podem ser acedidos individualmente a partir de um único nome.

Exemplo da sua utilização

Vetor de comissão mensais associadas a um determinado empregado ao longo de um ano:

12000	5000	2300	1230	7400	…

Declaração

A sua declaração é feita respeitando as mesmas regras de declaração de uma variável, a sua seguinte sintaxe:

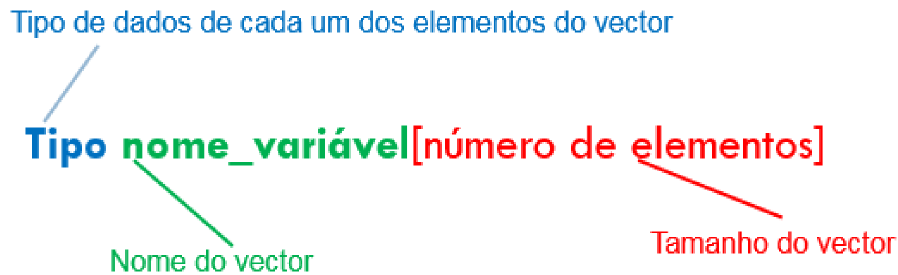


Figura 1: sintaxe de declaração de uma matriz unidimensional1

Exemplo: A declaração int v[10]; indica que v é um vetor de 10 elementos, ambos do tipo inteiro (int). Aloca em memória um espaço de 10 × 4 bytes consecutivos e o espaço é designado de v.

Outros exemplos

Inicialização de um vetor com todas as vogais do alfabeto:

char vogal v [5] = {'a','e','i','o','u' };

- Declaração de um vetor rendas como 100 reais:

float renda [100];

Inicialização do vetor com 3 elementos inicializados com os valores 5,10,15

int v [3] = {5,10,15};

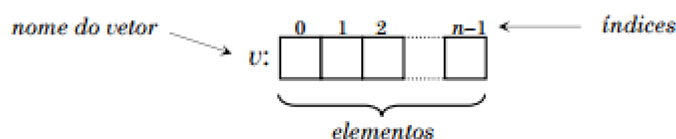
Inicialização do vetor com v elementos inicializados com os valores 2,3,5 3 os restantes ficam a 0.

int v [6] = {2,3,5}

uma declaração e inicialização incorreta

int v []

Para acessar um elemento de um vetor aplicamos o operador de [ ]. Os elementos são numerados de v [0], v [1], v [2], ..., a v[n−1] elementos. A seguir temos um exemplo de um vetor v, com os seus índices variando de 0 a n-1.



Em C, vetores unidimensionais globais e estáticos são automaticamente zerados pelo compilador. Mas, se for necessário, podemos inicializá-los explicitamente no momento em que os declaramos. Nesse caso, os valores iniciais devem ser fornecidos entre chavetas e separados por vírgulas conforme a sintaxe a seguir:

tipo nome-do-vetor[N] = {constante-1, constante-2..., constante-N};

Exemplo: int t [10] = {4,8,7,6,5,6,3,2,4,9}

Os valores são armazenados, a partir da posição 0 do vetor, na ordem em que são fornecidos; por exemplo, o valor 7 é armazenado em t [2].

Inicialização de um vetor com tamanho implícito

```
1   #include <stdio.h>
2   #include <stdlib.h>
3   #define N 10
4   main ()
5
6   {
7   int t[N]; int i;
8   for (i = 0; i < N; i++)
9   {
10      printf(" bom dia t[%d] :",i);
11      scanf("%d",&t[i]);}
12  for (i = 0; i < N; i++)
13      printf("n[%d] = %d\n",i,t[i]);
14        return 0;
15  }
16
```

Algumas notas sobre vetores

- Os elementos de um vetor são sempres armazenados em posições contíguas de memória.
- Os elementos de um vetor declarado sem qualquer inicialização contêm valores aleatórios.
- O índice do primeiro elemento de um vetor é sempre (ZERO).
- Os índices de um vetor com n elementos variam sempre de 0 a n-1.
- O valor existente numa posição dum vetor v, pode ser obtido através do índice em que essa posição está armazenada v[índice].
  Se o nº de inicialização for menor que o número de elementos do vetor, os elementos em falta são inicializados com o valor zero.
- Não se pode declarar vetor sem dimensão ou tamanho.

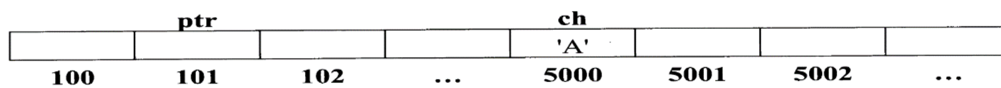
Para mais informações sobre estas notas consultar o capítulo 6 do livro (Damas, 2014, página 195).

Ponteiros em C

Como sabemos, tudo o que acontece no computador (ou quase tudo), se passa ou deve ser armazenado na memória RAM do computador (Randon Acess Memory, memória de acesso aleatório). É também na RAM que as variáveis que nós declaramos são armazenadas, podemos então dizer que a RAM é vista como um enorme vetor de bytes consecutivos.

Portanto, um ponteiro é uma referência a uma posição de memória. Ponteiros podem ser constantes ou variáveis e através deles podemos acessar dados ou código. São mecanismos flexíveis para a manipulação de dados, pois permitem manipular diretamente dados contidos em endereços específicos de memória.

Exemplo:



O prt é um ponteiro, por isso deverá conter o endereço de memória de outra variável, neste caso vai armazenar o endereço da variável ch.

Declaração e inicialização de ponteiros

Para declarar um ponteiro seguimos a mesma lógica de declaração de variáveis e vetores apresentados acima. A sua sintaxe é:

tipo_do_ponteiro *nome_da_variável;

Exemplos: tipo *variavel;

- int *nome;
- float *salario;
- char *sexo;
- struct aluno *disciplina;

Outros Exemplos: char *pc; int *pi;   float *pf; onde pc, pi e pf contém os endereços de variáveis do tipo char, int e float, respetivamente.

O operador unário & permite obter o endereço de uma variável:

Exemplo:

```
int  i, *pi; float f, *pf;

pi = &i; pf = &f;
```

Qualquer declaração de variável ocupa espaço na memória do computador. Portanto, o operador de referência & permite saber qual é o endereço na memória.

É o asterisco (*) que faz o compilador saber que aquela variável não vai guardar um valor, mas sim um endereço para aquele tipo especificado. Vamos ver exemplos de declarações:

O operador unário * fornece então o acesso direto ao valor do objeto apontado. Assim, se o pi é um apontador para um número inteiro i, *p representa o valor do i.

Exemplos:

int *pt; // ponteiro (pt) do tipo inteiro

char *temp, *pt2; ponteiros (temp, pt2) do tipo caracter

Exemplo:

```
1  #include <stdio.h>
2  #include <stdlib.h>
3
4  int main()
5  {
6      int i = 3;
7      int *p;
8      p = &i;
9      printf("*p = %d \n",*p);    //  *p = 3
10     return 0;
11
12 }
13
14
15
```

Os exemplos a seguir vão ajudar-te a entender melhor os conceitos de endereço e inicialização de ponteiros.

Suponha que a, b e c são variáveis inteiras então:

```
int *p;       /* p é um ponteiro para um inteiro */

int *q;

p = &a;       /* o valor de p é o endereço de a */

q = &b;      /* q aponta para b */

c = *p + *q;
```

Outro exemplo:

```
int *p;

int **r;          /* r é um ponteiro para um ponteiro para um
inteiro */

p = &a;         /* p aponta para a */

r = &p;         /* r aponta para p e *r aponta para a */

c = **r + b;
```

Outro exemplo:

```
int A [100] [300], i, j;

int *p;                  /* ponteiro para inteiro */

i = 3; j = 4;

p = A [0];               /* p aponta para o 1º elemento da 1ª
linha de A */

/* equivale a fazer p = &A [0] [0] */

P [i*300+j] = 4; /* equivale a fazer M[ i ] [j ] = 4   */
equivale a fazer M[ i ] [ j ] = 4 */
```

Operações sobre ponteiros

O valor de um ponteiro é um número inteiro, podemos aplicar as seguintes operações aritméticas:

adição: adiciona um número inteiro de um ponteiro. O resultado é um ponteiro do mesmo tipo que o ponteiro inicial;

subtração: subtrai um número inteiro de um ponteiro. O resultado é um ponteiro do mesmo tipo que o ponteiro inicial;

Ao contrário dos dois ponteiros apontam ambos para o mesmo tipo de objetos. O resultado é um número inteiro. Note-se que a soma dos dois apontadores não é permitida.

Exemplo:

```
1   #include <stdio.h>
2   #include <stdlib.h>
3
4   main()
5   { int i = 3; int *p1, *p2;
6       p1 = &i; p2 = p1 + 1;
7       printf("p1 = %ld \t p2 = %ld\n",p1,p2);
8   }
9
10
```

Resultado:

p1 = 4831835984   p2 = 4831835988.   // Diferença de 4 unidades (bytes)

```
1   #include <stdio.h>
2   #include <stdlib.h>
3
4
5   main()
6   { double i = 3; double *p1, *p2;
7       p1 = &i; p2 = p1 + 1;
8       printf("p1 = %ld \t p2 = %ld\n",p1,p2);
9   }
10
11
12
```

Resultado:

p1 = 4831835984   p2 = 4831835992. // Diferença de 8 unidades (bytes)

Operações Incremento

Um apontador pode ser incrementado assim como a variável. No entanto, o incremento de 1 unidade não significa que o endereça anteriormente armazenado no apontador seja incrementado a um byte.

Exemplo

```
ptr ++;

ptr=ptr+2;

prt +=4;   /* se ptr apontar para um float avança 4*4=16 bytes
*/
```

Operações decremento

O decremento funciona da mesma forma que o incremento apresentado acima. Por exemplo, se p e q são dois apontadores de tipo inteiro, a expressão p - q é um número inteiro cujo valor é igual a (p - q) / sizeof (tipo).

Exemplo de incremento e decremento ponteiros / matrizes

```
1   #include <stdio.h>
2   #include <stdlib.h>
3
4   #define N 5
5   int tab[5] = {1, 2, 6, 0, 7};
6   main() { int *p;
7       printf("\n ordem crescente:\n");
8       for (p = &tab[0]; p <= &tab[N-1]; p++)
9           printf(" %d \n",*p);
10          printf("\n ordem decrescente:\n");
11      for (p = &tab[N-1]; p >= &tab[0]; p--)
12          printf(" %d \n",*p);
13      }
14
```

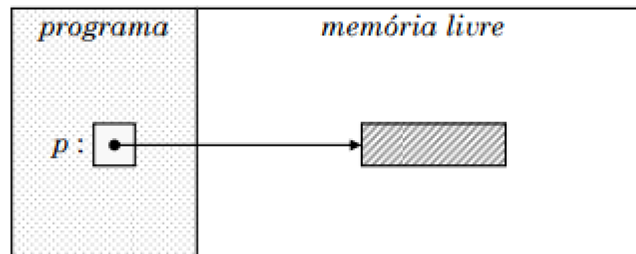
Resumo das operações sobre ponteiros

Operação	**Exemplo**	**Observações**
Atribuição	Ptr=&x	Podemos atribuir um valor (endereço) a um apontador. Se quisermos que aponte para nada podemos atribuir-lhe o valor da constante NULL
Incremento	Ptr=ptr+2	Incremento de 2*sizeof(tipo) de ptr
Decremento	Ptr=ptr-10	Decremento de 10*sizeof(tipo) de ptr
Apontado por	*ptr	O operador asterístico permite obter o valor existente na posição cujo endereço está armazenado em ptr.
Endereço	&Ptr	Tal como qualquer outra variável, um apontador ocupa espaço em memória. Desta forma podemos saber qual o endereço que um apontador ocupa em memória.
Diferença	Ptr1-ptr2	Permite-nos saber qual o número de elementos entre ptr1 e ptr2
Comparação	Prt1-ptr2	Permite-nos verificar, por exemplo, qual a ordem de dois elementos num vetor, através do valor dos seus endereços.

Figura 2: Operações sobre ponteiros: Fonte (Damas, 2014, pág. 283)

Alocação dinâmica de memória

Uma das aplicações mais interessantes de ponteiros é a alocação dinâmica de memória, que permite a um programa requisitar memória adicional para o armazenamento de dados durante a sua execução. Em geral, após um programa ter sido carregado para a memória, parte dela permanece livre. Então, usando a função malloc(), o programa pode alocar uma parte dessa área livre e acessá-lo através de um ponteiro como é ilustrado a seguir.



A função malloc

A função malloc pertence à biblioteca padrão stdlib.h e a sua sintaxe é a seguinte:

malloc(numero-bytes)

A função malloc() exige como argumento o tamanho, em bytes, da área de memória que será alocada. Então, se o espaço livre é suficiente para atender à requisição, a área é alocada e seu endereço é devolvido; senão, a função devolve um ponteiro especial, representado pela constante NULL. Além disso, como malloc() não tem conhecimento sobre o tipo dos dados que serão armazenados na área alocada, ela devolve um ponteiro do tipo void ⍰.

O argumento número-bytes é dada frequentemente usando a função sizeof () que retorna o número de bytes usados para armazenar um objeto.

Exemplo:

```
#include <stdlib.h>

int *p;

p = (int*)malloc(sizeof(int));

// ou (int*)malloc(4);
```

Inicialização de ponteiro com malloc

```
#include <stdio.h>

#include <stdlib.h>

main() {

 int i = 3;

 int *p;

 printf("valor de p antes de inicialização = %ld\n", p);
// mostra 0

 p = (int*)malloc(sizeof(int));

 printf("valor de p após inicialização= %ld\n",p);  // ex:
5368711424

*p = i;

printf("valor de *p = %d\n",*p);                              //
mostra 3

}
```

Inicialização de ponteiro sem malloc (atribuição direta com o endereço de um objeto existente)

```
#include <stdio.h>

#include <stdlib.h>

main() {

 int i = 3;

 int *p;

 printf("valor de p antes de inicialização= %ld\n", p);
// mostra 0

 p = &i;

printf("valor de p após inicialização= %ld\n",p);    // ex.
4831836000

printf("valor de *p = %d\n",*p);                            //
mostra 3

}
```

Nota:

A função malloc permite também alocar espaço para vários objetos contíguos na memória. Por exemplo:

```
#include <stdio.h>

#include <stdlib.h>

main() {

 int i = 3;

 int j = 6;

 int *p;

 p = (int*)malloc(2 * sizeof(int));

 *p = i;    *(p + 1) = j;

 printf("p = %ld \t *p = %d \t p+1 = %ld \t *(p+1) = %d
\n",p,*p,p+1,*(p+1));

 }
```

Foi, portanto, reservado para o endereço indicado pelo valor de p, 8 bytes em memória, que permite armazenar 2 objetos de tipo int.

O programa mostra então: p = 5368711424   *p = 3   p+1 = 5368711428   *(p+1) = 6.

Libertar espaço de armazenamento alocado

Quando não é mais necessário usar o espaço de memória alocada dinamicamente deve-se liberar este espaço através da função free.

free(nome-do-ponteiro);

Portanto, em qualquer malloc () deve ser associado a instrução/função

free() para libertar a memória.

Conclusão

Nesta unidade foram destacados os principais conceitos sobre vetores, ponteiros e o processo de alocação de memória com a função malloc (). Os exemplos apresentados permitem praticar e entender melhor estes conceitos. Utilizando estes conceitos vistos nesta seção resolve os seguintes exercícios.

Avaliação

1. Escreva um programa que declare um vetor com n=10 números reais e coloque na i-ésima posição o resultado de i*(n-1).

2. Escreva um programa que devolve o valor máximo de um vetor de n números inteiros, soma os elementos positivos e mostra a quantidade de elementos negativos.

3. Diga qual é o resultado inicial dos seguintes códigos. Verifique a sua resposta, e seguidamente executa cada um dos programas no compilador e acrescenta comentários em cada uma das linhas de instrução.

void fct1(int a, int b);
 	
void fct2(int* ptr1, int* ptr2);
 	
int main(void)
 	
{
 	
int a = 3;
 	
int b = 4;
 	
fct1(a, b);
 	
printf("%d %d \n", a, b);
 	
fct2(&a, &b);
 	
printf("%d %d \n", a, b);
 	
return 0;
 	
}
	main()
	
	{
	int vet[] = {4,9,12};
	int i,*ptr;
	ptr = vet;
	for(i = 0 ; i < 3 ; i++) {
	
	printf("%d ",*ptr++);
	}
	}
		define MAX 100
		void maior (int A[MAX] [MAX], int n, int k, int Lin, int Col)
		
		{
		int i, j;
		
		for (i=0; i<n; i++)
		{
		for (j=0; j<n; j++)
		{
		i f (A [i] [j] > k)
		
		{
		k = A [i] [j];
		Lin = i;
		Col = j;}
		} }
		}

4. Qual é o resultado do seguinte programa:

```
#include <stdio.h>

#include <conio.h>

#include <alloc.h>

#include <stdlib.h>

void main()

{

char frase [ 255], * aux;

printf (“Entre com uma frase qualquer: “);

gets ( frase);

aux = frase;

while ( * aux) printf ( “%c”, * aux ++);

printf ( “\n”);

getch ( );
```

5. Faça de duas maneiras diferentes, um programa que lê 10 inteiros em uma matriz e que procura o maior e o menor elemento desta matriz:

   a. Usando o conceito de matriz

   b. Usando o conceito de ponteiro

6. Qual o resultado do seguinte programa:

```
# include <stdio.h>

main()

{

int i, j;

for (i = 1; i <= 5; i = i + 1)

{

for (j = 1; j <= 10; j = j + 1)

printf(“%2d * %2d = %2d\n”, i,j,i*j);

getchar();

}

}
```

Atividade 3.2 - Correspondências entre vetor e ponteiros, strings e caracteres

Introdução

Esta atividade apresentará a relação existente entre os apontadores com e vetores e strings, uma vez que os apontadores são normalmente utilizados na manipulação de vetores e strings. Compreender esta relação é muito importante para o desenvolvimento de uma programação estruturada e de problemas com um maior grau de complexidade.

Vamos então aprender os conceitos importantes e aplicar igualmente um conjunto de exemplos e exercícios.

Detalhes da atividade

Relação entre vetor e apontador

Introdução

Todo o vetor em C é realmente um ponteiro constante. Na declaração, int v [10]; v é um ponteiro constante (não editável) cujo valor é o endereço do primeiro elemento do vetor. Podemos usar um ponteiro inicializado para percorrer alguns os elementos do vetor.

O nome de um vetor corresponde ao endereço do seu primeiro elemento, isto, se v for um vetor, v==&v [0], ou seja, o nome do vetor não é mais do que o endereço do primeiro elemento deste vetor.

Embora o nome de um vetor seja um apontador para o primeiro elemento do vetor, este apontador não pode ser alterado durante a execução do programa a que pertence.

Exemplo:

```
Int v [3] = {10,20,30};      // vetor com 3 elementos

Int *ptr;                      // apontador para inteiro
```

Apontar para o primeiro elemento do vetor

Existem duas formas de colocar o apontador ptr a apontar para o 1° elemento do vetor v:

```
prt=&v [0];   // prt fica com o endereço do primeiro elemento

prt= v;       // pois v==&V [0];
```

Exemplo:

```
Int v [3] ={10,20,30};                    // vetor com 3
elementos

Int *ptr;                                      // apontador
para inteiro

ptr=v;                                        // passa a
apontar para 1° elemento do vetor

printf(“%d %d\n”, v [0], *ptr);  //             10   10
```

Exemplo:

```
1 #include <stdio.h>
2 #include <stdlib.h>
3 #define N 4
4
5 int v[4] = {1, 2, 6,  7};
6 main() {
7  int i;
8  int *p;
9  p = v;
10  for (i = 0; i < N; i++) {
11      printf(" %d \n",*p);
12      p++; }
13 }
```

Nota:

Um ponteiro deve ser sempre inicializado por uma alocação dinâmica, ou atribuindo uma expressão endereço, por exemplo p = &i;

Operador de indexação [ ] dos vetores podem também ser aplicado a qualquer objeto p de tipo ponteiro. Ele está ligado ao operador * pela fórmula: p [i] = *(p + i);

Ponteiros e vetores são manipulados exatamente da mesma maneira. Por exemplo, o programa da página anterior poderá ser escrito desta forma:

```
1 #include <stdio.h>
2 #include <stdlib.h>
3 #define N 4
4
5 int v[4] = {1, 2, 6, 7};
6 main() {
7  int i;
8  int *p;
9  p = v;
10  for (i = 0; i < N; i++)
11  printf(" %d \n", p[i]);
12  }
```

Vetores multidimensional

Agora vamos aprender algumas técnicas de como trabalhar também com vetores com mais do que uma dimensão (denominados de vetores multidimensionais), recordando que não existe qualquer limite para o número de dimensões que um vetor pode conter.

Um vetor multidimensional (também denominada de matriz) é uma coleção homogénea bidimensional, cujos elementos são distribuídos em linhas e colunas. Se A é uma matriz m×n, então suas linhas são indexadas de 0 a m−1 e suas colunas de 0 a n−1. Para acessar um particular elemento de A, escrevemos então A[i][j], sendo i o número da linha e j o número da coluna que o elemento ocupa.

Para declarar um ponteiro que aponta para um objeto do tipo * (matriz de duas dimensões) deve seguir a sintaxe seguinte:

tipo ** nome-do-ponteiro;

Exemplo com um programa de alocação de uma matriz

```
main() {

int m, n; int **v;

 ....

v = (int**)malloc(m * sizeof(int*)); // m ponteiro que
corresponde a linhas

for (i = 0; i < m; i++) v[i] = (int*)malloc(n * sizeof(int));
// cada linha com n int

….

for (i = 0; i < m; i++) free(v[i]);  //espaço alocado
liberado

free(v);

}
```

Definição de constantes

A escrita de programas deve ser realizada de forma a que uma pequena alteração não provoque grandes transformações no código dos mesmos.

Uma constante não é mais que um nome a que corresponde um valor fixo (não se pode alterar ao longo de uma execução).

Nota: As constantes devem ser declaradas fora das funções, de forma a serem "visíveis" ao longo de todo o programa. Normalmente a sua definição é realizada diretamente a seguir às linhas das bibliotecas #include-

Matriz de várias dimensões

Podemos também declarar uma matriz multidimensional em C (ex: 2 dimensões) conforme a sintaxe abaixo:

tipo nome-da-matriz[numero-linhas][numero-colunas]

Na verdade, uma matriz bidimensional é uma matriz unidimensional, no qual cada elemento é por si só uma matriz. É possível acessar um elemento desta matriz com a expressão "matriz [i][j]". Para inicializar uma matriz multidimensional em tempo de compilação, usa-se uma lista na qual cada elemento é uma lista de constantes:

Exemplo:

```
1   #include <stdio.h>
2   #include <stdlib.h>
3   #define M 2
4   #define N 4
5
6   int tab[M][N] = {{1, 2, 3,5}, {4, 5, 6,9}};
7
8   int main()
9   {
10      int i, j;
11       for (i = 0 ; i < M; i++) {
12          for (j = 0; j < N; j++)
13              printf("tab[%d][%d]=%d\n", i, j, tab[i][j]);
14       }
15  }
16
17
```

Strings e carateres

A linguagem C tem a capacidade de manipular facilmente os seus tipos de dados básicos (char, int, float e double), no entanto, apresenta algumas limitações no que respeita ao tratamento de vetores e strings, não fazendo o seu processamento diretamente.

Uma string em C é um conjunto de carateres armazenados num vetor. Daí a importância de desenvolver estes dois conceitos na programação em C e puder conhecer assim a sua utilidade.

Em C, uma string é uma série de caracteres terminada com um caracter nulo, representado por '\0'. Tal como o vetor é um tipo de dado capaz de armazenar uma série de elementos do mesmo tipo, a string é uma série de caracteres, é bastante natural que ela possa ser representada por uma matriz/cadeia de caracteres.

Exemplos de strings:

"maria"

"Paulo Barbosa"

"Jose"

"A"

"Bolo de chocolate com peso 2,5 kg"

Exemplos de carateres:

'L'

'>'

'A'

Nota: 'A' é o caracter A (1 byte). A string "A" é um vetor de carateres que internamente ocupa, não 1 mas 2 bytes, e cujo primeiro elemento é o caracter 'A'.

Exemplo:

```
char *ch;

ch = "Esta é uma string";

#include <stdio.h>

main() {

 int i;

 char *ch;

 cadeia = "cadeia de caracteres";

 for (i = 0; *ch != '\0'; i++) ch++;

 printf("numero de caracteres = %d\n",i);

}
```

Outro Exemplo:

```
char *s;
s = malloc (10 * sizeof (char));
s [0] = 'A';
s [1] = 'B';
s [2] = 'C';
s [3] = '\0';
s [4] = 'D';
```

Depois da execução desse fragmento de código, o vetor s [0..3] contém a string ABC. O caractere nulo marca o fim dessa string. A parte s[4..9] do vetor é ignorado.

Inicialização automática de strings

O processo de inicialização de strings segue os mesmos procedimentos de inicialização e vetores já apresentado acima.

Exemplos:

```
Char nome [20] =" Andre";
Char nome [20] = {'A', 'n', 'd', 'r', 'e' };
```

Leitura e escrita de strings

Função printf e puts.

A função printf recebe como formato uma string, que pode ser escrita diretamente.

Exemplo:

```
Printf ("hello world");
```

A função puts coloca no ecrã a string passada à função e em seguida faz automaticamente a mudança de linha.

Exemplo:

```
Puts ("hello world") é equivalente a Printf ("hello world\n")
```

Exemplo:

```
1 #include <stdio.h>
2 #include <string.h>
3
4 // as mais utilizadas : printf, scanf, gets, puts
5 main ()
6 {
7       char lugar[25]; char * s;
8       int dia, mes, ano;
9       printf("Entrar o nome: \n");
10      gets (s);
11      printf("Entrar a data de nascimento: \n");
12      scanf("%s %d %d %d ", dia, mes, ano);
13 }
```

Manipulação de strings em C, funções padrão

As strings não podem ser manipuladas diretamente (comparadas, concatenadas, etc.), pois não são um tipo básico da linguagem C.

A seguir serão apresentadas um conjunto destas funções que são standards e as mais utilizadas.

Nome funções	O que que faz
strlen	Devolve o comprimento de um string
strcpy	Copia uma string para outra
strcat	Concatenação de string
strcmp	Comparação alfabética de string
stricmp	Comparação de string com ignore case
strchr	Procura um carácter numa string
strstr	Procura uma string dentro da outra
strlwr	Converte todos os caracteres de uma string para minúscula
strupr	Converte todos os caracteres de uma string para maiúscula

Figura 3: Funções standard de manipulação de string em C: Fonte (Damas, 2014, pág. 252)

Exemplo:

```
1   #include <stdio.h>
2   #include <stdlib.h>
3   #define N 8
4
5   char tab[N] = "exemplo";
6   int main()
7   {
8   int i;
9   for (i = 0; i < N; i++)
10      printf("tab[%d] = %c\n",i, tab[i]);
11          return 0;
12  }
13
```

Conclusão

Acabamos de terminar as atividades anteriores de relação entre as vetores e ponteiros, bem como a manipulação de strings. Conhecendo estes importantes tipos de dados, vamos aplicar um conjunto de problemas mais completos envolvendo estes conceitos. Vamos exercitar!

Avaliação

1. Faça comentários em todas as linhas de instrução presentes no seguinte trecho de código:

```
for (i=0; i<n; i++) {

for (j=0 j<m; j++) {

printf ("%d ", M[i] [j]);

}

printf ("\n");

}
```

2. O que fazem os seguintes programas abaixo:

```
#include <conio.h>
#include <stdio.h>
void main() {
     int vet[] = {4,9,12};
     int i,*ptr;
     ptr = vet;
     for(i = 0 ; i < 3 ; i++) {
          printf("%d ",*ptr++);
     }
}
```

```
#include <conio.h>
#include <stdio.h>
void main(){
     int vet[] = {4,9,12};
     int i,*ptr;
     ptr = vet;
     for(i = 0 ; i < 3 ; i++) {
          printf("%d ",(*ptr)++);
     }
}
```

3. Faça um programa que leia um inteiro n < 100 e os elementos de uma matriz real quadrada A$n \times n$ e verifica se a matriz A tem uma linha, coluna ou diagonal composta apenas por zeros.

4. Considere a seguinte matriz de inteiros, contendo sete elementos:

```
int valores [] = {4, 5, 1, 8, 2, 2, 10};
```

a. Escreva um programa C que usa um laço for para percorrer todos os elementos desta matriz e mostrar a soma de seus valores. Seu programa deverá mostrar uma saída com a mensagem:

A soma dos valores da matriz é: 32

5. Reescreva o seguinte código no compilador e:

Adicione comentários nas linhas de código de todos os ciclos for.

Execute o programa e explique cuidadosamente, mostrando os resultados.

```
1 #include <stdio.h>
2 #include <stdlib.h>
3
4 main() {
5 int MAT[4][3];
6 int L, C;
7
8  for (L=0; L<=3; L++) {
9      for (C=0; C <= 2; C++) {
10         printf("Entre com o valor para a posicao MAT[%d][%d]:\n",L,C);
11         scanf("%d",&MAT[L][C]);
12     }
13  }
14 for (L=0; L<=3; L++) {
15     for (C=0; C <= 2; C++) {
16         printf("Valor MAT[%d][%d]=%d:\n",L,C,MAT[L][C]);
17     } }
18  printf("FIM\n");
19   system("PAUSE");
20 }
```

6. Escreva um programa que pede ao utilizador para introduzir um número inteiro entre 1 a 7 e que mostra o nome do dia correspondente da semana (por exemplo segunda por 1, terça-feira por 2, ...). Utilize a tabela de strings para incluir os dias de semana.

Atividade 3.3 - Outros tipos de dados: Estrutura, União, enumeração

Introdução

Após a utilização de vetores e ponteiros, vamos prosseguir com o estudo de outros tipos de dados que podem também ser definidos pelos utilizadores e que são importantes também na programação uma vez que os mesmos permitem resolver problemas mais complexos e diversificados. As estruturas formam a base a partir da qual é possível implementar diversos outros tipos de dados mais complexos. Esta unidade introduz o uso de estruturas, bem como uniões e enumerações.

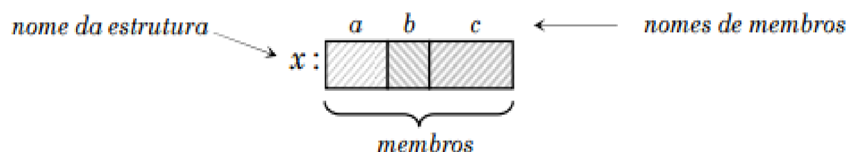
Detalhes da atividade

O que é uma estrutura

Estrutura (agrupamento de variáveis sob um nome) é uma coleção arbitrária de variáveis logicamente relacionadas. Assim como na matriz, essas variáveis compartilham o mesmo nome e ocupam posições consecutivas de memória. As variáveis que fazem parte de uma estrutura são denominadas de membros ou campos e são identificadas por nomes.

A declaração de estruturas segue a seguinte sintaxe:

```
struct    [nome_da_estructura] {
                    tipo1 campo1, campo2;
                    ....
                    typeN campo; };
```



Conforme apresentado acima temos uma estrutura x e os membros são x.a, x.b e x.c.

Exemplo: estrutura para armazenar uma data

```
struct {
 int dia;
 int mes;
 int ano;
} hoje;
```

Esse trecho de código declara a variável hoje como uma struct cujos membros são dia, mes e ano, todos do tipo int. Para atribuir valores a eles, podemos escrever:

hoje.dia = 15;

hoje.mes = 1;

hoje.ano = 2007;

Definição de tipo de estrutura typedef

Uma das desvantagens existente na utilização de estrutura está na declaração das variáveis que tem sempre se ser precedidas da palavra reservado struct seguida do nome da estrutura.

Neste caso uma estrutura pode ser representada unicamente através de uma palavra (sinónimo), tal como é possível fazer com os tipos básicos da linguagem C.

Para resolver esse problema, é possível usar a palavra reservada typedef cuja sintaxe é a seguinte:

Typedef tipo_existente sinónimo

Exemplo:

```
typedef struct data {
int dia;
int mes;
int ano;
};
```

Esse fragmento de código cria um tipo de estrutura, cujo rótulo é data, através do qual podemos declarar variáveis da seguinte maneira:

struct data hoje;

struct data ontem, amanha;

O rótulo de uma estrutura, usado isoladamente, não é reconhecido pelo compilador com sendo um tipo de dados. Assim, o uso da palavra struct é obrigatório.

Estruturas dentro de estruturas

É importante ressaltar que uma estrutura pode conter, na sua definição, variáveis simples, vetores, apontadores ou mesmo estruturas.

Todos o novo tipo de estrutura tem que ser previamente declarado/definido.

Exemplo:

```
typedef struct
{
int dia;
char mes [3+1];
int ano;
} DATA;

typedef struct s_pessoa
{
Char nome [100];
Int idade;
Float salario;
DATA dt_nasc;
} PESSOA;
```

Foi, portanto, declarado um novo tipo de estrutura denominado PESSOA que contém o nome, a idade, o salário e a data de nascimento.

A linguagem C permite a utilização de ponteiros para estrutura exatamente como permite ponteiros para os outros tipos de dados. Como outros ponteiros, ponteiros para estrutura também são declarados colocando o símbolo * na frente do nome da estrutura.

Exemplo:

```
struct date {
        int dia;
        int mes;
        int ano;
          }
struct date *dat;          // ponteiro para uma estrutura
struct date tdate[5]; // matriz da estrutura
tdate[0].dia = 12;  tdate[2].mes = 4;  dat = &tdate[2]; dat
-> dia = 12;
dat++; dat-> mes = 2;
```

Passagem por endereço

```
struct date {
        int dia;
        int mes;
        int ano;
   }
void initdia (struct date *s, int j) {
    s ->dia=j;
   }

main() {
      struct date dt;
      initdia(&dt,12);
      printf("%d ", dt.dia);
    }
```

Definição de novos nomes aos tipos de dados

A linguagem C permite também que sejam definidos novos nomes aos tipos de dados, utilizando a palavra-chave typedef.

A forma geral de um comando typedef é:

typedef "tipo" "novo_nome", onde tipo é qualquer tipo de dados permitido e novo_nome é o novo nome para esse tipo.

Para declarar uma variável do tipo "mystruct", é usada uma declaração como se segue:

struct mystruct s;

Exemplos de utilização de typedef em estruturas:

```
typedef struct {

unsigned x;

float f;

}mystruct;
```

O typdedef torna o código mais fácil de ler, então pode-se declarar variáveis deste tipo de estrutura usando neste exemplo a declaração: mystruct s.

Tipo de dado - União

Em linguagem C, uma union é uma posição de memória que é compartilhada por duas ou mais variáveis diferentes, geralmente de tipos diferentes, em momentos diferentes. A sua definição é semelhante à definição de uma estrutura, sua sintaxe:

```
union identificador{

tipo nome_variavel;

tipo nome_variavel;

tipo nome_variavel;

…..

}variáveis_uniao;
```

Exemplo:

```
union u_type{
int i;
char ch;
};
```

Tipo de dado - enumeração

Uma enumeração é uma extensão da linguagem C acrescentada pelo padrão ANSI. É um conjunto de constantes inteiras que especifica todos os valores que uma variável desse tipo pode ter. A sua sintaxe:

emum < identificador> {lista_de_enumeração}<lista de variáveis">

Exemplo:

```
enum dia_semana {
domingo=1,        // tratará como 1
segunda,          // tratará como 2
terca,              // tratará como 3
quarta,           // tratará como 4
quinta,          // tratará como 5
sexta,         // tratará como 6
sabado,    // tratará como 7
         } ;
```

Conclusão

Nas sessões anteriores vimos que todos os dados eram armazenados em variáveis, e também que através do uso de vetores foi possível armazenar um conjunto de dados do mesmo tipo em apenas uma variável.

Concluímos então que com o uso de estruturas, é possível agrupar sobre um único nome um conjunto de uma ou mais variáveis por forma a facilitar a sua utilização.

A utilização de uniões e enumerações permitem também modelar objetos mais complexos do que com o uso de vetores/matrizes vistos anteriormente. Estes tipos de dados podem ser utilizados em combinação com ponteiros por exemplo para criar listas, pilhas, filas, árvores (que são estruturas de dados dinâmicos) a ser apresentados em outros módulos.

Avaliação

1. Defina em C um novo tipo de estrutura denominado e pessoa, que contenha as seguintes caraterísticas: nome, Idade, salário e um indicador que mostra se o registo está apagado ou não.

2. Para cada uma das situações a seguir, declare o que se pede:

   a. para um concurso de beleza precisa-se armazenar os dados das 30 candidatas, que são: número da inscrição, nome, altura, peso, naturalidade e estado.

   b. um apicultor necessita armazenar os dados de 100 colmeias, que são: código, região, estado, qualificação (forte, médio, fraco), número de abelhas;

   c. um comerciante deseja armazenar os dados de 50 produtos, que são: código, descrição, quantidade em estoque, quantidade mínima, preço;

   d. para uma locadora de vídeo deve-se armazenar os dados de 500 filmes, que são: código, título, gênero, categoria e valor da locação.

3. Seja a seguinte estrutura:

```
struct time {
inthora;
int min;
int seg; }
```

e seja a seguinte declaração:

struct time tempo; // variável global

Faça um programa contendo funções para:

a) receber uma quantidade de tempo em segundos e transformar a mesma em horas, minutos e segundo, preenchendo e imprimindo a variável tempo.

b) ler a variável tempo (campos hora, min e seg) e devolver a quantidade de horas em segundos.

Resumo da Unidade

Os diferentes tipos de dados vistos nesta unidade permitem-nos aprofundar os nossos conhecimentos no desenvolvimento de programas.

Os tipos de dados básicos permitem-nos desenvolver algoritmos simples enquanto que os algoritmos mais complexos são desenvolvidos através das técnicas mais sofisticadas de estruturas de dados avançados.

Nesta unidade trabalhamos com os tipos de dados desde de vetores, ponteiros, matrizes, strings e carateres a estruturas, união e enumeração.

Com estes conhecimentos pensamos que já estás apto para programar em C, resolvendo assim problemas mais complexos que eventualmente aparecem no nosso dia a dia.

Vamos praticar!

Avaliação da Unidade

Verifique a tua compreensão!

Trabalho de casa

Instruções

Responde a todas as questões a seguir colocadas, a avaliação tem a duração de 2h.

Critérios de Avaliação

As pontuações serão colocadas em cada uma das questões que no total corresponde a 20 pontos.

Avaliação

Questão 1 (10 pontos)

1. Qual a diferença entre "A" e 'A'?
2. Escreva um programa que leia nomes e sobrenomes de pessoas e os mostre na tela no formato Sobrenome, Nome. O programa deve terminar quando um nome nulo for introduzido.
3. Escreva um programa que solicite ao utilizador nomes de pessoas e os apresente na tela até que seja introduzido o nome "SAIR".
4. Escreva um programa através de um menu utilizando as funções de manipulação de string como:

a. ler a string do teclado

b. copiar uma string

c. colocar os caracteres de uma string a maiúscula

d. mostrar a inversa da string

e. concatenar strings

Questão 2 (5 pontos)

1. Escreva um programa que o utilizador ao introduzir o país é lhe mostrado a respetiva moeda, conforme a numeração a seguir:

   Euro 1 - Portugal

   Dólar 2 - Estados Unidos

   Escudos 3 - Cabo Verde

   Real 4 - Brasil

   Libra 5 - Reino Unido

   Peso 6 - Argentina

   Franco 7 - Benim

   Xelim 8 - Quénia

Questão 3 (5 pontos)

1. Seja a seguinte estrutura, contendo os seguintes campos:

```
struct time
{
        int hora;
        int min;
        int seg;
}
```

Seja a seguinte declaração: struct time tempo; // variável global

a. Escreva um programa contendo funções para:

b. Receber uma quantidade de tempo em segundos e transformar a mesma em horas, minutos e segundo, preenchendo e imprimindo a variável tempo.

c. Ler a variável tempo (campos hora, min e seg) e devolver a quantidade de horas em segundos.

d. Função principal: chama as duas funções acima.

Trabalho Laboratorial da unidade 3

Laboratório 1

Objetivos: criar projetos no DEV C++, aplicar o conceito de matriz, aplicar operações sobre matrizes, comentar códigos.

Atividades: Todos os exercícios a seguir devem ser realizados no compilador Dev C++. Guarde todos os seus programas que não tem indicação de nome na tua pasta de trabalho com nomes sugestivos da seguinte forma exer1, exer2...

Exercício 1: ligue o computador e abra o compilador DEV-C++, crie um novo projeto com o nome de mat_pont_string. Guarde este projeto na sua pasta de trabalho.

Exercício 2: escreva um programa em C que carrega uma matriz inteira com os números de 0 a 99. Guarde este programa na sua pasta de trabalho.

Exercício 3: Qual a saída do seguinte programa?

```
#include<stdio.h>
#include <stdlib.h>
int main()
{
int * tab1 = malloc(4 * sizeof(int));
int i;
for(i = 0 ; i < 4 ; i++)
tab1[i] = i;
 int * tab2 = malloc(3 * sizeof(int));
for(i = 0 ; i < 3 ; i++)
tab2[i] = 100 + i;
int * tmp = tab1;
tab1 = tab2;
tab2 = tmp;
printf ("Elément de tab1 : %d\n", tab1[0]);
printf ("Elément de tab2 : %d\n", tab2[0]);
free (tab1);
free (tab2);

  system("PAUSE");
  return 0;

}
```

Exercício 4: escreva uma função que recebe uma matriz de números reais B de tamanho 300×300 das quais somente n linhas e colunas estão sendo consideradas e devolve a soma dos elementos de sua diagonal.

Recursos: Compilador Dev C++, livros, acesso à internet.

Duração: 1h

Respostas:

Exercício 1: veja todos os passos neste link: http://educaonline.eng.br/UNISANTA/HTML/DOWNLOAD/APOSTILAS/apostila%20de%20dev.pdf

caso ainda não tiver o DEV C++ instalado podes fazer o download e instalação no seu computador seguindo todos os passos neste link.

Exercício 2:

```
void main(void)
{
  int x[100];  /* isto reserva 100 elementos inteiros */
  int t;

  for(t=0; t<100; ++t) x[t] = t;
}
```

Exercício 3:

Elemento da Tab 1: 100

Elemento da Tab 2: 0

Exercício 4:

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

float soma_diagonal (float B[300][300], int n) {
  int i;
  float r = 0;
  for (i=0; i<n; i++) {
    r = r + B[i][i];
  }
  return r;
}

int main () {
  float C[300][300], soma;
  int m;

  m = 3;
  C[0][0] = 2; C[0][1] = -2, C[0][2] = 4;
  C[1][0] = 3; C[1][1] = -1, C[1][2] = 7;
  C[2][0] = 5; C[2][1] = -3, C[2][2] = 3;

  soma = soma_diagonal (C, m);

  printf ("Soma = %f\n", soma);

  return 0;
}
```

Critério de avaliação: este exercício tem no total 6 pontos distribuídos da seguinte forma: exercício 1: 1 ponto; exercício 2: 2 pontos; exercício 3: 2 pontos; exercício 4: 1 ponto

Referências e links:

Helbert, S. (1995). C Completo e Total. McGraw-hill, inc 3ª edição.

Apostila Dev C++ Online, disponível em: http://educaonline.eng.br/UNISANTA/HTML/DOWNLOAD/APOSTILAS/apostila%20de%20dev.pdf

C programming. Disponível em : http://www.indiabix.com/

Matrizes, Ponteiros e funções, disponível em: http://www.ime.usp.br/~hitoshi/introducao/23-matrizes_ponteiros.pdf

Laboratório 2

Objetivos: utilizar conceitos de ponteiro, realizar operações sobre ponteiros, aplicar a relação entre matriz e ponteiro.

Atividades:

Exercício 1: explique a diferença entre:

p++;   b) (*p) ++;   c) *(p++);

Exercício 2: realize o seguinte teste online sobre conceitos gerais de ponteiros neste link: http://www.indiabix.com/c-programming/pointers/

Exercício 3: verifique no compilador se o seguinte programa apresenta erros. Se sim identifique e corriga e se não explica a sua saída:

```
#include <stdio.h>

#include <stdlib.h>

int main()

{

    float i=10, *j;

    void *k;

    k=&i;

    j=k;

    printf("%f\n", *j);

    return 0;

}
```

Exercício 4: Qual a saída do seguinte programa:

```
#include<stdio.h>
int main()
{
    char str [20] = "Hello";
    char *const p=str;
    *p='M';
    printf ("%s\n", str);
    return 0;     }


Mello  b)Hello   c)HMello  d)MHello
```

Recursos: Compilador Dev C++, livros, acesso à internet.

Duração: 1:30h

Respostas:

Exercício 1:

Incremente o ponteiro, ou seja, o endereço. Após esta instrução, o ponteiro p passará a apontar para a posição de memória imediatamente superior. Se em um vetor, o ponteiro passará a apontar a próxima posição do vetor.

Incremente o conteúdo apontado por p, ou seja, o valor armazenado na variável para qual p está apontando.

Incremente p (como em p++) e acessa o valor encontrado na nova posição. Se em um vetor, esta expressão acessa o valor da posição imediatamente superior a armazenada em p antes do incremento.

Exercício 2: neste teste online existe o botão resposta basta clicar para veres as respostas corretas das questões.

Exercício 3: Este programa não contém erros, a sua saída é: 10.000000.

Exercício 4: Opção correta é alínea a).

Critério de avaliação: este exercício tem no total 6 pontos distribuídos da seguinte forma: exercício 1: 1 ponto; exercício 2: 2 pontos; exercício 3: 2 pontos; exercício 4: 1 ponto.

Referências e links:

Helbert, S. (1995). C Completo e Total. McGraw-hill, inc 3ª edição.

Apostila Dev C++ Online, disponível em: http://educaonline.eng.br/UNISANTA/HTML/DOWNLOAD/APOSTILAS/apostila%20de%20dev.pdf

C programming. Disponível em : http://www.indiabix.com/

Matrizes, Ponteiros e funções, disponível em: http://www.ime.usp.br/~hitoshi/introducao/23-matrizes_ponteiros.pdf

Laboratório 3

Objetivos: utilizar conceitos de string, realizar operações com as funções de manipulação de cadeias de strings.

Atividades:

Exercício 1: Qual das seguintes funções é usada para encontrar a primeira ocorrência de uma determinada cadeia de string em outra cadeia?

strchr()   b. strrchr()   c. strstr()   d) strnset()

Exercício 2: Qual é a saída do seguinte programa:

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

int main()
{
   printf("%d\n", strlen("123456"));
   return 0;
}
```

Exercício 3: Qual é a saída do seguinte programa se os carateres a, b e c foram introduzidos.

```
#include<stdio.h>

int main()
{
   void fun();
   fun();
   printf("\n");
   return 0;
}
void fun()
{
   char c;
   if((c = getchar())!= '\n')
      fun();
   printf("%c", c);
}
```

Exercício 4: escreva um programa em C que conta o número de vogais presentes em uma cadeia de caracteres a partir do teclado, por exemplo:

Entrar com a seguinte cadeia de caracteres: cadeia de carateres

A saída é a seguinte:

Número de a: 4

Número de e: 3

Número de i: 1

Nombre de o: 0

Nombre de u: 0

Exercício 5: Este programa compila sem erros? Justifique a tua resposta.

```
#include<stdio.h>
int main()
{
    char a[] = "India";
    char *p = "BIX";
    a = "BIX";
    p = "India";
    printf("%s %s\n", a, p);
    return 0;
}
```

Recursos: Compilador Dev C++, livros, acesso à internet.

Duração: 1:30h

Respostas:

Exercício 1: alínea c. strstr()

Exercício 2: saída 6

Exercício 3: cba

Exercício 4:

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

```
#include<stdlib.h>
int main (int argc, char *argv [])
 {
   char phrase[100];
   char * adr;
   int na=0, nu=0, ny=0, ne=0, ni=0, no=0;
   printf("Entra um texto \n");
   gets(phrase);
   adr = phrase; //endereço do primeiro carater
   printf ("\nDigitou:");
   printf("%s \n",frase);
while (*adr!='\0')
{
   switch (*adr){
         case 'a' :
               na++;
               break;
          case 'e' :
               ne++;
               break;
          case 'i':
               ni++;
               break;
          case 'o' :
               no++;
               break;
         case 'u' :
               nu++;
               break;
   }
```

```
    adr++;
  }
  printf ("\n O registo e o seguinte: \n");
  printf ("Numero de a: %d \n", na);
  printf ("Numero de e: %d \n", ne);
  printf ("Numero de i: %d \n", ni);
  printf ("Numero de o: %d \n", no);
  printf ("Numero de u: %d \n", nu);
  system ("pause");

  }
```

Exercício 5: Não, porque nós podemos atribuir uma nova sequência de caracteres para um ponteiro, mas não para uma matriz neste caso matriz a [].

Critério de avaliação: este exercício tem no total 8 pontos distribuídos da seguinte forma: exercício 1: 1 ponto; exercício 2: 1 ponto; exercício 3: 1 ponto; exercício 4: 1 ponto1, exercício 5: 2 pontos.

Referências e links:

Apostila Dev C++ Online, disponível em: http://educaonline.eng.br/UNISANTA/HTML/DOWNLOAD/APOSTILAS/apostila%20de%20dev.pdf

C programming. Disponível em: http://www.indiabix.com/

Unidade 4. Ficheiros, Testes e depuração de Programa

Introdução à Unidade

Esta unidade vai apresentar-lhe os conceitos sobre os comandos de entrada e saída fazendo referência ao sistema de ficheiros em C, uma ferramenta poderosa e flexível e que permite-lhe satisfazer qualquer necessidade de programação da linguagem C. A unidade vai ser concluída com a realização de testes e depuração de programas.

Objetivos da Unidade

Após a conclusão desta unidade, deverá ser capaz de:

1. Definir um ficheiro de dados
2. Utilizar as operações de manipulação de um ficheiro
3. Testar um programa
4. Depurar um programa

Termos-chave

Ficheiro: é uma sequência de bytes que reside em memória lenta, tipicamente um disco magnético.

Entrada/saída: troca de informação entre o computador e o utilizador.

Depuração: processo de examinar o código de (um programa), a fim de identificar, localizar e suprimir falhas ou erros.

IDE: é um programa de computador que reúne características e ferramentas de apoio ao desenvolvimento de software.

Atividades de Aprendizagem

Atividade 4.1 - Ficheiros em C

Introdução

Até este capítulo, todos os programas que desenvolvemos solicitaram aos utilizados dados de entrada que o programa manipulava e os mesmos eram perdidos após terminarmos o programa uma vez que estes dados não eram armazenados num repositório permanente.

Nesta atividade vais aprender a manipular ficheiros, usando a linguagem C. Para além do conceito de ficheiro vais também identificar algumas operações a serem executadas sobre o mesmo nomeadamente os modos de leitura e escrita entre outras operações.

Detalhes da atividade

Conceito de ficheiro

A utilização de ficheiros pode ser vista em dois sentidos:

- O ficheiro é uma fonte de dados para o programa: neste caso trata-se de um ficheiro de entrada de dados (input).
- O ficheiro é o destino dos dados gerados pelo programa: neste caso trata-se de um ficheiro de saída de dados (output).

Manipulação/operações básicas sobre ficheiro em C

A figura 4 apresenta as principais funções de manipulação de ficheiros utilizando a linguagem C:

Função	Ação
fopen()	Abre um arquivo
Fclose ()	Fecha um arquivo
putc() e fputc()	Escreve um caractere em um arquivo
getc() e fgetc()	Lê um caractere de um arquivo
fseek()	Posiciona em um registro de um arquivo
fprintf()	Efetua impressão formatada em um arquivo
fscanf()	Efetua leitura formatada em um arquivo
feof()	Verifica o final de um arquivo
fwrite()	Escreve tipos maiores que 1 byte em um arquivo
fread()	Lê tipos maiores que 1 byte de um arquivo

Figura 4: Funções de manipulação de ficheiros em C:
Fonte (souza, (s/d)

Abertura de ficheiro em C

Esta operação consiste em associar uma variável do nosso programa ao ficheiro que pretendemos processar, ou, por outras palavras, representar internamente o nome físico do ficheiro através de um nome lógico que corresponde à variável do nosso programa que o irá suportar.

A função fopen () abre uma stream em C e associa um ficheiro a ela. Ela retorna o ponteiro de ficheiro associado e mais frequentemente o ficheiro é um ficheiro em disco. Esta função devolve o valor NULL (nulo) ou um ponteiro associado ao ficheiro, devendo ser passado para função o nome físico do ficheiro e o modo como este ficheiro deve ser aberto.

Protótipo da função *fopen* ():

fopen (“nome_ficheiro”,”modo”);

Depois de abrirmos o ficheiro podemos realizar algumas operações sobre o mesmo, nomeadamente as operações básicas de leitura e escrita de dados.

Exemplo:

```
if ((ficheiro = fopen(“texto.txt”,”w”)) == NULL) {

printf (“\n ficheiro TEXTO.TXT não pode ser aberto: prima uma tecla”);

getch();

}
```

De acordo com este exemplo o ficheiro “texto.txt”, está sendo aberto apenas para escrita (w-write).

A tabela a seguir apresenta os modos de manipulação de ficheiros e os seus respetivos significados.

Modo	Significado
"r"	Abre um arquivo texto para leitura. O arquivo deve existir antes de ser aberto.
"w"	Abrir um arquivo texto para gravação. Se o arquivo não existir, ele será criado. Se já existir, o conteúdo anterior será destruído.
"a"	Abrir um arquivo texto para gravação. Os dados serão adicionados no fim do arquivo ("append"), se ele já existir, ou um novo arquivo será criado, no caso de arquivo não existente anteriormente.
"rb"	Abre um arquivo binário para leitura. Igual ao modo "r" anterior, só que o arquivo é binário.
"wb"	Cria um arquivo binário para escrita, como no modo "w" anterior, só que o arquivo é binário.
"ab"	Acrescenta dados binários no fim do arquivo, como no modo "a" anterior, só que o arquivo é binário.
"r+"	Abre um arquivo texto para leitura e gravação. O arquivo deve existir e pode ser modificado.
"w+"	Cria um arquivo texto para leitura e gravação. Se o arquivo existir, o conteúdo anterior será destruído. Se não existir, será criado.
"a+"	Abre um arquivo texto para gravação e leitura. Os dados serão adicionados no fim do arquivo se ele já existir, ou um novo arquivo será criado, no caso de arquivo não existente anteriormente.
"r+b"	Abre um arquivo binário para leitura e escrita. O mesmo que "r+" acima, só que o arquivo é binário.
"w+b"	Cria um arquivo binário para leitura e escrita. O mesmo que "w+" acima, só que o arquivo é binário.
"a+b"	Acrescenta dados ou cria uma arquivo binário para leitura e escrita. O mesmo que "a+" acima, só que o arquivo é binário

Figura 5: Modos e significados de manipulação de ficheiros

Modo de acesso aos ficheiros:

Se o ficheiro tiver no modo a letra r, o ficheiro deve existir.

Se o ficheiro tiver no modo a letra w, o ficheiro pode não existir. Neste caso, vai ser criado. Se o arquivo já existe, seus antigos conteúdos serão perdidos.

Se o modo ficheiro tiver no modo a letra a, o ficheiro pode não existir. Neste caso, vai ser criado. Se o ficheiro já existe, os novos dados serão adicionados no final do ficheiro anterior.

Fechar um ficheiro em C

O fecho de um ficheiro elimina a ligação entre a variável e o ficheiro existente no disco.

A função fclose () serve para esvaziar da memória um ficheiro, que se associa diretamente ao nome lógico do ficheiro. Esta função retorna um inteiro que é zero se a operação for bem-sucedida (e diferente de zero em caso contrário).

Protótipo da função *fclose* ():

fclose ("ficheiro");

Opções de guardar e ler dados em ficheiros

Existem várias funções em C para a operação de gravação e leitura de dados em ficheiros conforme indicado nos exemplos abaixo:

putc () ou fputc (): grava um único caracter no ficheiro.

Sintaxe: putc (caracter, ficheiro);

fprintf (): grava dados formatados no ficheiro, de acordo com o tipo de dados (float, int, ...). esta função é semelhante à função printf, porém ao invés de imprimir na tela, grava no ficheiro.

Sintaxe: `fprintf (arquivo, "formatos", var1, var2 ...);`

fwrite (): grava um conjunto de dados heterogêneos (tipo struct) no ficheiro.

Sintaxe: `fwrite (buffer, tamanho em bytes, quantidade, ponteiro de ficheiro);`

fscanf(): retorna a quantidade de variáveis lidas com sucesso.

Sintaxe: `fscanf(ficheiro, "formatos", &var1, &var2 ...);`

Sempre que um programa em C é executado são abertos, pelo menos, 5 ficheiros standard.

- Ficheiros standard: stdin, stdout, stderr, stdprn e stdaux
- Stdin: representa o standard input e está normalmente associado ao teclado.
- Stdout: representa o standard output e está normalmente associado ao ecrã.
- Stderr: representa o standard error, local para onde devem ser enviadas as mensagens de erro de um programa.
- Stdaux: representa o denominado aux device e está definido como porta principal de comunicações (COM1 num PC).
- Stdprn: representa o standard printer.

O ponteiro do ficheiro em C

O ponteiro é um meio comum que une o sistema C ANSI de E/S. Um ponteiro de ficheiro é uma variável ponteiro do tipo FILE. Para ler ou escrever ficheiros, seu programa precisa usar ponteiros de ficheiros. Para obter uma variável ponteiro de ficheiro, use o comando a seguir:

*FILE * fp;*

Exemplo de uma utilização de funções em ficheiros

Este programa lê dez nomes do teclado e escreve no ficheiro nomes.txt.

```
1 #include <stdio.h>
2
3 main(int argc,char *argv[]) {
4  FILE *fp;
5  char nome[50];
6  if ((fp=fopen ("nomes.txt","w")) != NULL) {
7  for(int i=0; i<10; i++) {
8  printf("Escreva um nome: ");
9  gets(nome);
10 fprintf(fp, "Nome %d: %s\n", i+1, nome);
11 }
12 }
13 fclose(fp);
```

Recomenda-se a leitura do capítulo 10, do livro principal do curso (Damas, 2014, pág. 341).

Conclusão

Os ficheiros em C são vistos como sequência de bytes sem qualquer formato especial. Nesta unidade desenvolvemos o conceito de ficheiro e as suas principais funções/operações na linguagem C. Este mecanismo vai permitir o controle das operações de entrada e saída no computador uma vez que possibilita genericamente quatro operações importantes como: abertura, leitura e escrita e fecho. Vamos então praticar estes conceitos!

Avaliação

1. Escreva um programa em C, que abre um ficheiro alunos.txt, pede ao utilizador para introduzir dados de alunos como nome e nota (o utilizador define o número de alunos), abre o ficheiro alunos.txt e escreve estes dados neste ficheiro. Deve seguidamente fechar o ficheiro. Verifica os dados introduzidos no ficheiro alunos.txt.

2. Pretende-se guardar em disco, num ficheiro de texto, informação relativa a pessoas, utilizando para isso uma estrutura idêntica que permita armazenar nome, cidade, telefone e data de nascimento. Os elementos pertencentes a uma mesma pessoa devem ser guardados, como texto e na mesma linha do ficheiro, segundo o exemplo seguinte:

Ana Maria Lopes	Coimbra	239700900	1	6	1970
Clara Pinto dos Santos	Leiria	244890456	12	7	1975
Ricardo Teixeira	Figueira da Foz	233234790	26	8	1971
Sónia Costa	Lisboa	218998770	2	1	1976

São reservados 40 caracteres para o nome, 20 para a cidade, 10 para o telefone, 3 para dia e mês e 4 para ano.

a. Escreva uma função que peça ao utilizador os dados de uma pessoa e os grava num ficheiro.

b. Escreva uma função que apresente no ecrã os dados de todas as pessoas guardadas no ficheiro conforme a alínea a).

Atividade 4.2 - Teste e depuração de programa

Introdução

Teste e depuração de um programa já foi trabalhado na unidade 4 do módulo "princípios de programação". Nesta unidade vais apenas refrescar estes conceitos e utilizá-los nos seus programas.

Detalhes da atividade

Teste de programa

O teste de um programa permite verificar se o programa contém algum erro/bug que pode ser encontrada na instrução ou em comando incorretos ou alguma falha encontrada na saída do programa, no resultado.

Depuração/Debugging

É um processo que permite-lhe encontrar e corrigir defeitos num programa. O processo de depuração deve passar antes pela compilação do programa.

Para esta unidade propomos assim a realização das atividades propostas no módulo anterior bem como a realização do processo de depuração no DEV C++.

Podes consultar estes links com as informações sobre o DEV C++ e o processo de depuração:

http://wiki.icmc.usp.br/images/e/e3/Manual_Dev_C.pdf

https://www.youtube.com/watch?v=kHFpzxMFB3E

http://www.gnu.org/software/gdb/ , GNU debugger online.

Conclusão

Concluímos assim este módulo com os processos de teste de programa e depuração no DEV C++ e em síntese já temos todas as ferramentas para o desenvolvimento e teste de programas mais complexos em C.

Avaliação

1. O que é o makefile. Dê um exemplo.
2. Faça uma pesquisa no google e explica por tuas palavras o que entendes por teste de programa. Dê um exemplo.
3. Abra o Dev-C++ e abre a opção: Tools/Compiler Options
4. Ative a opção "Add the following commands when calling compiler" e digite o argumento -O0
5. Ative no Guia Settings a opção Linker e marque para Yes a opção "Generate debugging information"

   Nota: antes de depurar o seu programa já sabes que é necessário que ele deve passar primeiro pelo processo de compilação. Podes escolher um dos programas apresentados nos exemplos nesta unidade para depurar.

6. Registe todos os passos necessários para a depuração do seu programa, podes acompanhar a depuração do programa nos links apresentados acima.

Resumo da Unidade

Nesta unidade aprendeste a utilizar opções de entrada e saída no computador nomeadamente os ficheiros tendo conhecimento de várias funções de manipulação de ficheiros para a sua utilização na construção de programas na linguagem de programação C.

Os conceitos de teste de programa e depuração foram também relembrados nesta unidade através da sua aplicação prática. Portanto, agora já podes após criar os teus programas em C, analisar e corrigir os erros e falhas.

Avaliação da Unidade

Verifique a tua compreensão!

Trabalho de casa 4

Instruções

Faça todos os exercícios propostos a seguir, a avaliação tem uma duração de aproximadamente 2 horas.

Critérios de Avaliação

A pontuação total é de 20 pontos distribuídos em cada uma das questões a seguir colocadas.

Avaliação

1. Defina uma estrutura Professor com quatro campos: Nome, curso, NIF e Salário, que guarda os dados de 50 professores.
2. Implemente em C um programa que abre o ficheiro registro.txt, lê os seus valores e armazena-os no ficheiro. O ficheiro registro.txt contém informações sobre os professores de uma Universidade que estão no formato. Nome, Curso, NIF e Salário.
3. Implemente em C uma função que pede um salário ao utilizador e verifica quantas vezes esse valor aparece no ficheiro.
4. Implemente uma função em C que pede um nome e um NIF ao utilizador, e os insere no ficheiro dados.txt. Essa inserção deverá ser feita, somente se esse NIF não existir no ficheiro. Caso já exista, o programa deverá dar a mensagem de erro correspondente. Implemente uma função C que cria um novo ficheiro (backup.txt) e escreve nele todos os nomes dos professores do registo anterior.

Leituras e outros Recursos

As leituras e outros recursos desta unidade encontram-se na lista de Leituras e Outros Recursos do curso.

Trabalho Laboratorial da unidade 4

Laboratório de depuração de um programa em C

Objetivos: identificar todos os passos de depuração, depurar um programa completo, diferenciar erros de falhas num programa em C.

Atividades: vamos nesta atividade depurar um programa, mas antes disso vamos recordar alguns conceitos importantes sobre o mesmo.

Exercício 1: por favor abra o bloco de notas no teu computador e regista todos os passos para a depuração de um programa em C? Guarde este ficheiro com o nome de depuraçao_notas na tua pasta de trabalho para outras consultas.

Exercício 2: uma vez conhecida e registada todos os passos do processo de depuração, vamos praticar? Vamos abrir o DEV C++ para depurar o nosso programa.

Recursos: Compilador Dev C++, tutoriais sobre depuração e IDE, acesso à internet.

Duração: este laboratório tem duração de 2h.

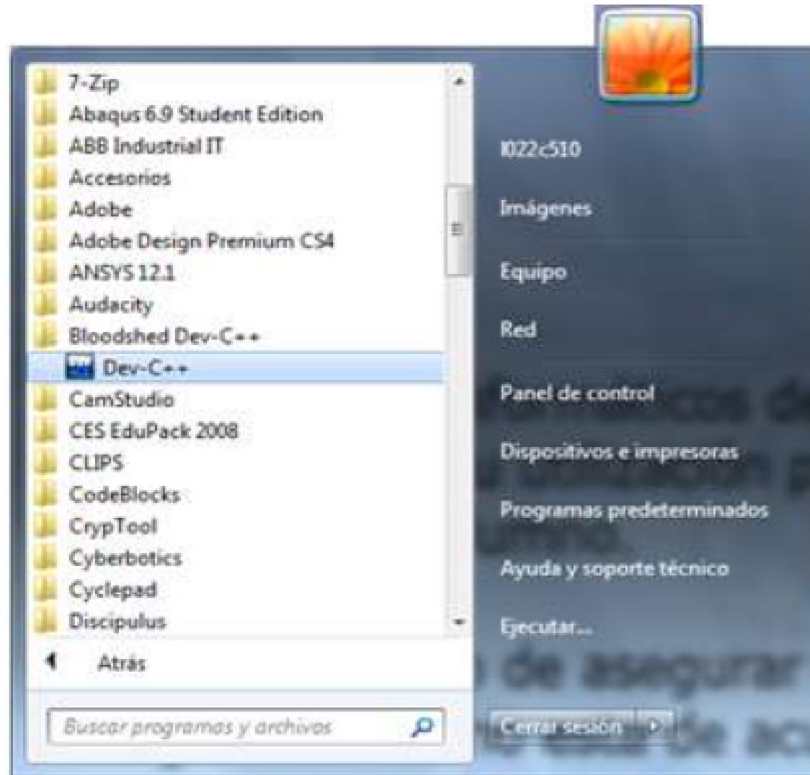
Respostas:

Exercício 1: os passos para a depuração de um programa são:

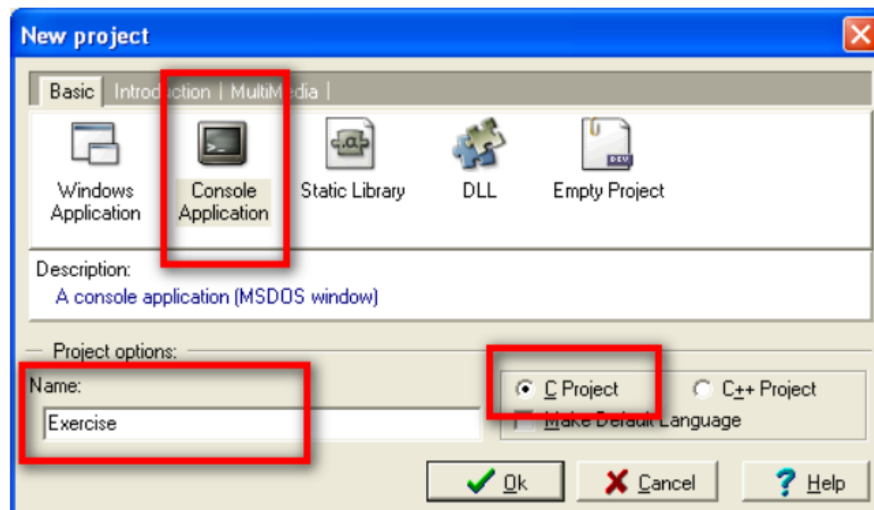
1. Criar o projeto
2. Escrever o código (o programa) e guardar o ficheiro.
3. Compilar e linkar o programa, para gerar o programa executável.
4. Fixar os erros de compilação (quando a sintaxe de um programa não estiver correto o processo de compilação falha e o compilador apresenta mensagem de erros. O programador deve corrigir os erros.
5. Executar o programa para avaliar a sua funcionalidade.
6. Fixar os erros de execução, se as ações executadas pelo programa não são como esperadas, é necessário corrigir o código-fonte, também pode ser conveniente usar o depurador para encontrar erros complexos.

Exercício 2: processo de depuração de um programa

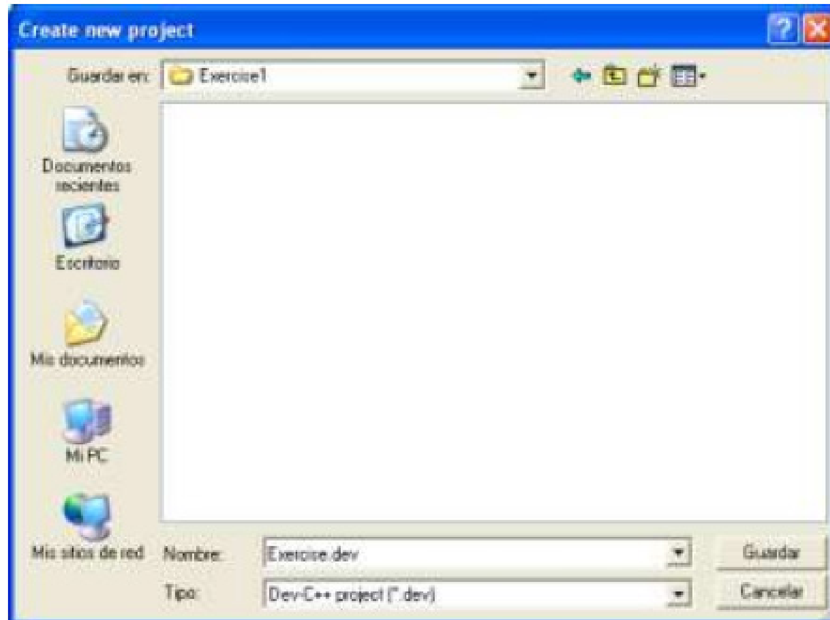
1. Abra o Dev C++



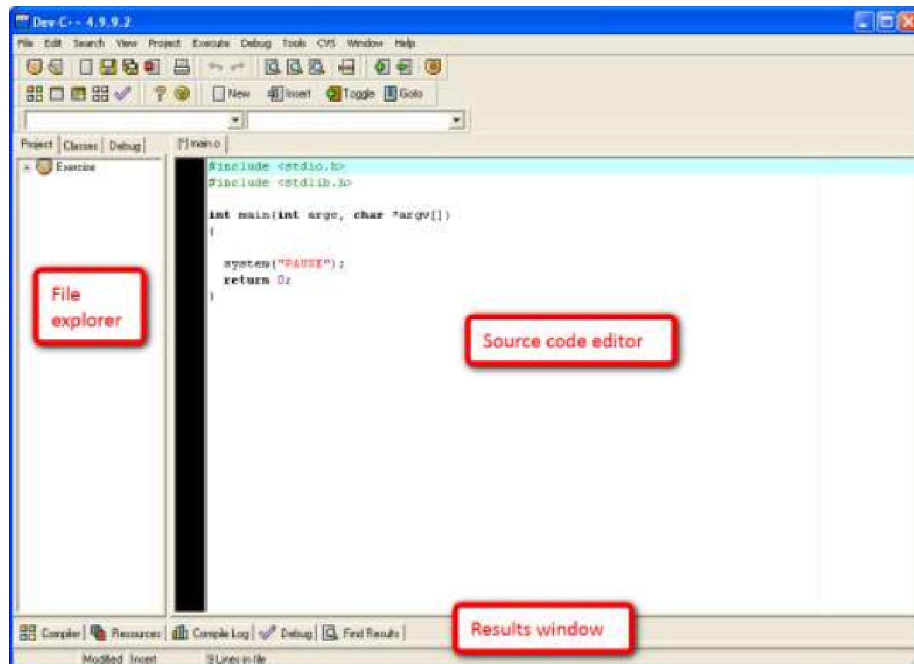
2. Crie um novo projeto através da ativação das opções abaixo:



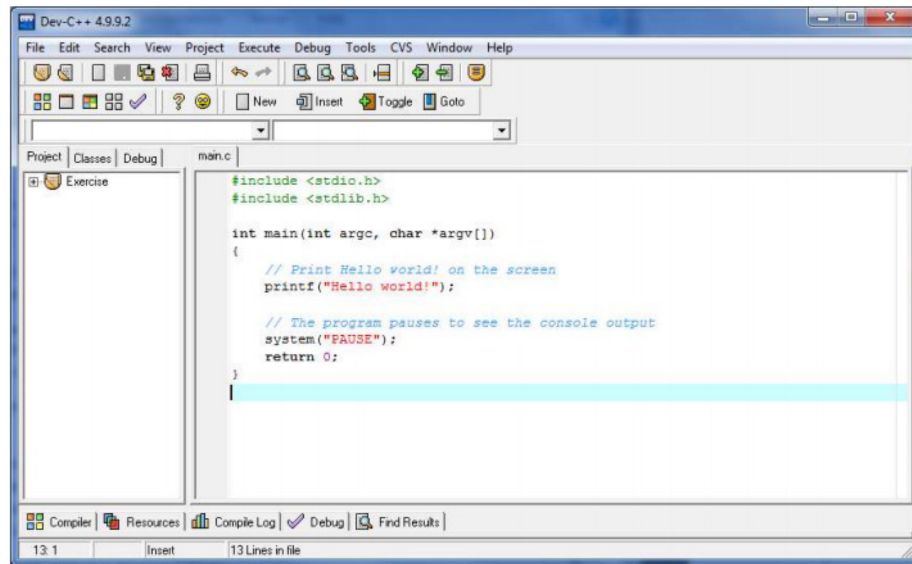
3. Selecione a pasta onde vais guardar o teu programa:



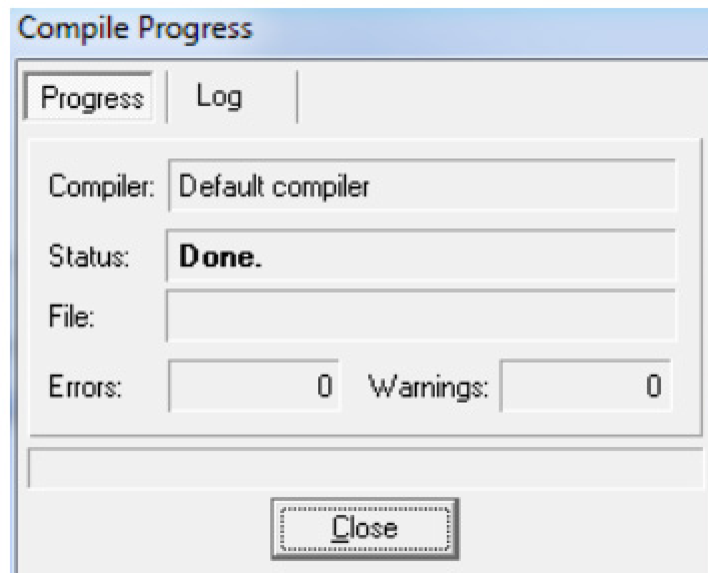
4. Após a indicação da pasta onde vais guardar o teu projeto, o IDE gera um código básico por defeito o main.c conforme apresentado a seguir.

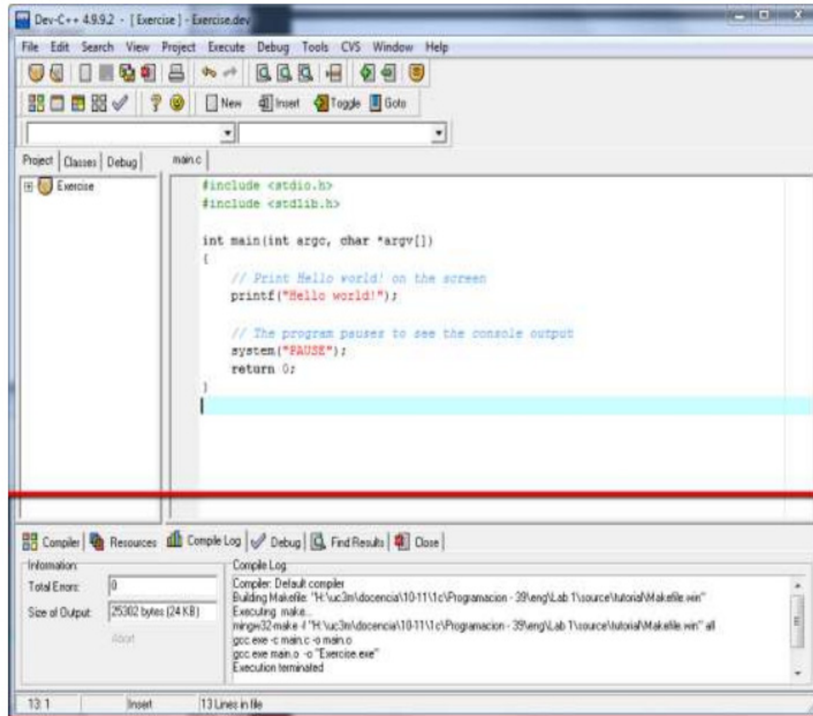


5. Crie um programa, vamos utilizar este exemplo simples a seguir:

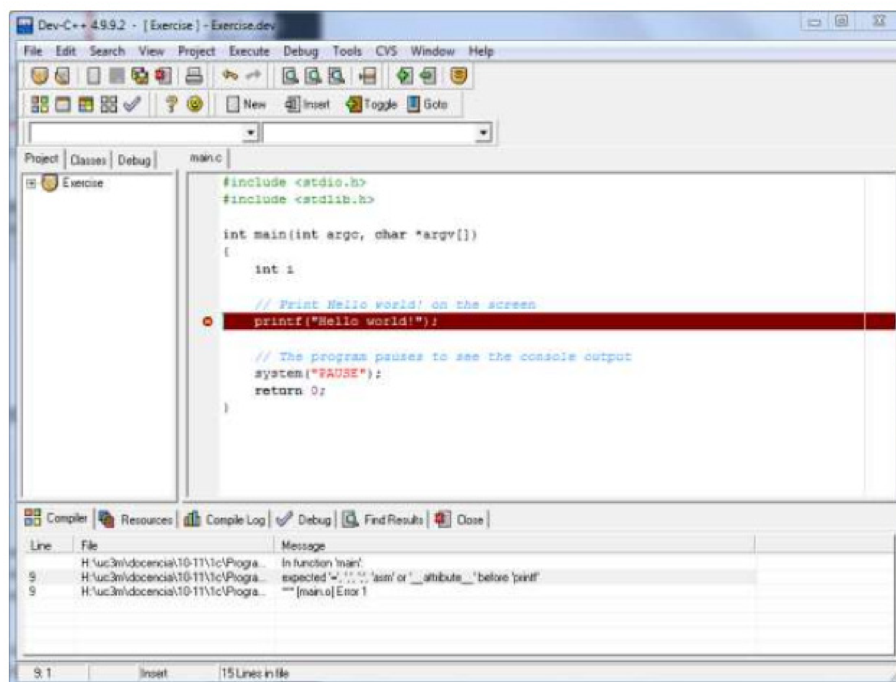


6. Compile o programa através da opção compilar ou da conjugação da tecla Ctrl+9





7. Fixe os erros de compilação



8. Execute o programa (usa o botão compilar + executar) altera o programa "Hello world" para poder introduzir um erro de compilação; por exemplo, digitar incorretamente a função printf ou remover uma das marcas de aspas duplas " " no caráter.

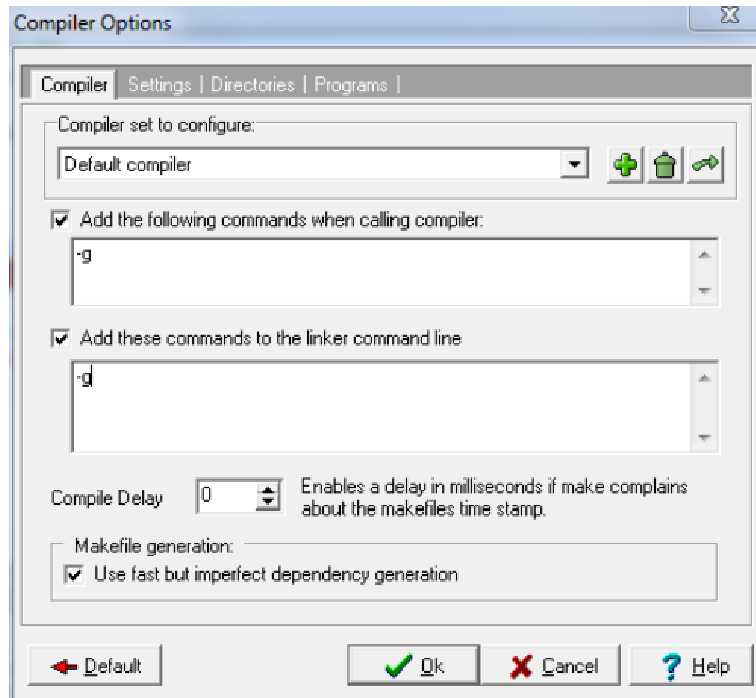
9. corda. Compilar e executar o programa (use o botão Compilar & Run). Verifique mensagens o erro e fixa os erros.

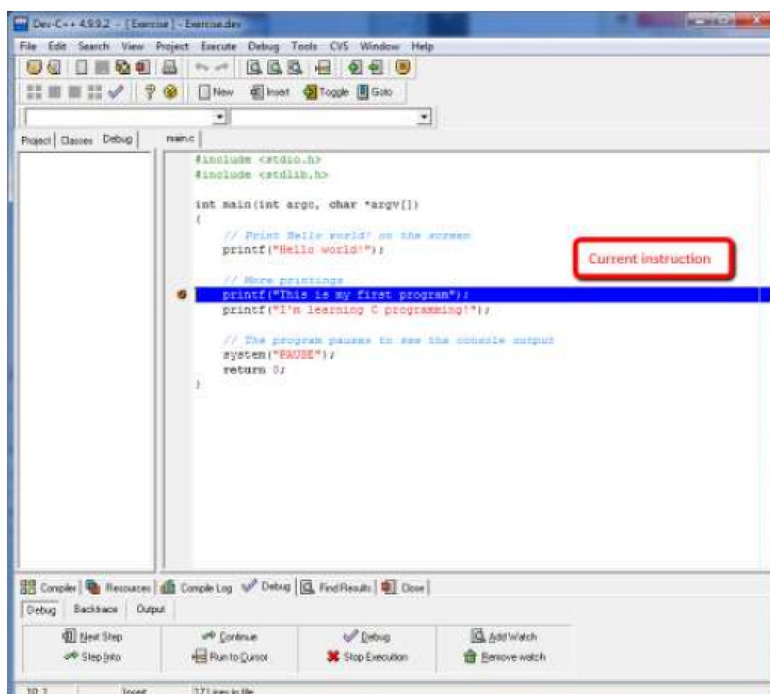
a. Realize o processo de debuger da seguinte forma:

b. No Dev-C++ abra a opção: Tools/Compiler Options

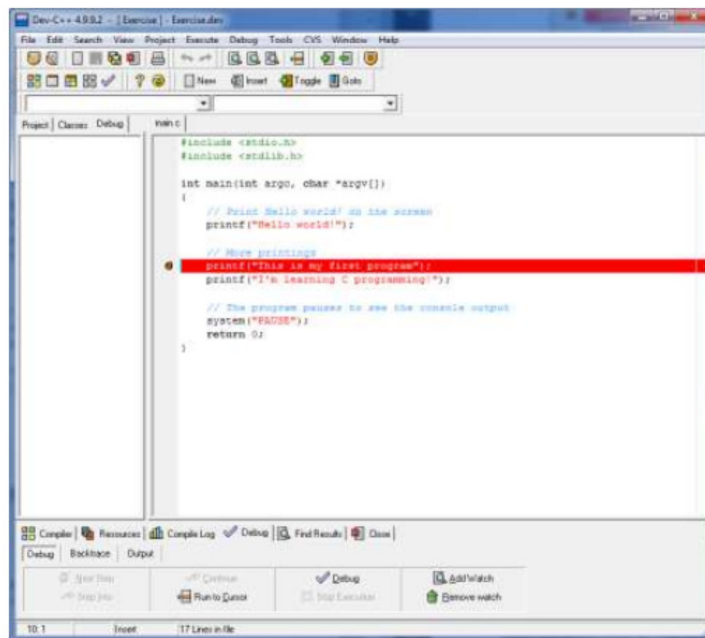
c. Ativa a opção "Add the following commands when calling compiler" e digite o argumento -O0

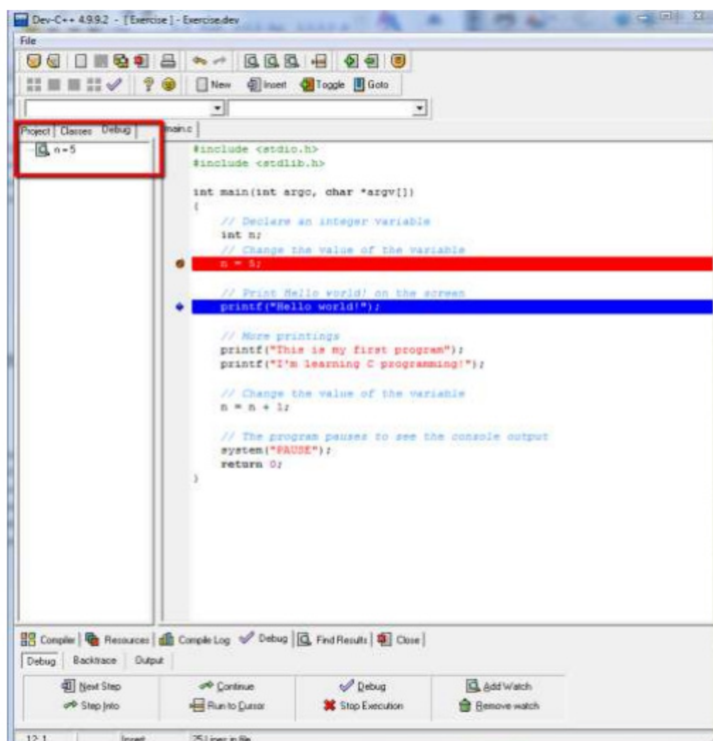
d. Ativa no Guia Settings a opção Linker e marque para Yes a opção «Generate debugging information».





Opções de debug:





Critério de avaliação: este laboratório tem uma cotação de 10 pontos.

Referências e links:

Programação com IDE no DEV C++. disponível em: http://ocw.uc3m.es/ingenieria-informatica/programming-in-c-language-2013/IntroductiontoDevCIDE.pdf

Depuração e teste de programas em C. disponível em http://wiki.icmc.usp.br/images/1/18/Aula12-Arquivos-Parte1.pdf

C programming and C++ Programming. disponível em: http://www.cprogramming.com/

The GNU Project Debugger. Disponível em http://www.gnu.org/software/gd

Avaliação do Curso

Exame 1

Instruções

Faça todos os exercícios propostos a seguir, a avaliação tem uma duração de aproximadamente 2 horas.

Critérios de avaliação

A pontuação total é de 20 pontos distribuídos em cada uma das questões a seguir colocadas.

Avaliação

Exercício 1: Vetores em C (6 pontos total)

1. Implemente uma função em C que pede ao utilizador as notas de 10 alunos de IP, armazena num vetor e retorna a média. (2 pontos)
2. Desenvolva um programa para ler uma variável de 10 elementos numéricos e verificar se existem elementos iguais a 30. Se existirem, escreva a posições em que estão armazenadas. (2 pontos)
3. Faça um programa em C que leia um vetor N [20]. A seguir, encontre o menor elemento do vetor N e a sua posição dentro do vetor, mostrando, por exemplo: "O menor elemento de N é", M, "e sua posição dentro do vetor é:", P (2 pontos)

Exercício 2: Matriz em C (14 pontos)

4. Crie uma matriz 4x4 com valores no intervalo [1,20]. Escreva um programa que transforme a matriz gerada numa matriz triangular inferior, ou seja, atribuindo zero a todos os elementos acima da diagonal principal. Imprime a matriz original e a matriz transformada? (6 pontos)

5. Crie uma função para somar duas matrizes. Esta função deve receber duas matrizes e retornar a soma em uma terceira matriz. Caso o tamanho da primeira e segunda matriz seja diferente a função retornará um erro. Caso a função seja concluída com sucesso a mesma deve retornar o valor zero (0). Utilize aritmética de ponteiros para manipulação das matrizes. Mostre o uso dessa função em um programa feito em C. (8 pontos)

Exame 2

Instruções

Faça todos os exercícios propostos a seguir, a avaliação tem uma duração de aproximadamente 3 horas.

Critérios de avaliação

A pontuação total é de 20 pontos distribuídos em cada uma das questões a seguir colocadas.

Avaliação

Exercício 1: Cadeia de caracteres/string em C (4 pontos)

6. Escreva um programa que lê duas cadeias de caracteres de tamanho 10 e mostra-as concatenadas na tela? (2 pontos)

7. Escreva um programa que lê uma cadeia de tamanho 3 e mostra na tela a soma dos códigos ASCII dos caracteres da cadeia? (2 pontos)

Exercício 2: Ponteiro em C (8 pontos)

8. Explique a saída do seguinte programa:

```
#include <stdio.h>

#include <string.h>

int main(void)

{

char t[];

strcpy(t,”abcde”);

printf(“\n%ld %c”, t, *t);

printf(“\n%ld %c”, t+1, *(t+1));

printf (“\n%ld %c”, t+, *(t+));

printf (“\n%ld %c”, t+, *(t+));

printf (“\n%ld %c”, t+4, *(t+4));

return 0;

}
```

Exercício 3: Funções em C (8 pontos)

9. Escreva um programa utilizando funções, que calcula o valor das seguintes séries, com n a ser introduzida pelo utilizador:

$$Q(n) = 1 + \frac{1}{4} + \frac{1}{9} + \ldots + \frac{1}{n^2}$$

Exame 3

Instruções

Faça todos os exercícios propostos a seguir, a avaliação tem uma duração de aproximadamente 2 horas.

Critérios de avaliação

A pontuação total é de 20 pontos distribuídos em cada uma das questões a seguir colocadas.

Avaliação

Exercício 1: Questões sobre estruturas em C (8 pontos)

10. Seja uma estrutura para descrever os carros de uma determinada revendedora, contendo os seguintes campos:

    marca: string de tamanho 15

    ano: inteiro

    cor: string de tamanho 10

    preco: real

    1. Defina a estrutura Carro de tamanho 20.
    2. Defina a função para ler o vetor carros.
    3. Defina uma função que receba um preço e imprima os carros (marca, cor e ano) que tenham preço igual ou menor ao preço recebido.
    4. Defina uma função que leia a marca de um carro e imprima as informações de todos os carros dessa marca (preço, ano e cor).
    5. Defina uma função que leia uma marca, ano e cor e informe se existe ou não um carro com essas características. Se existir, informar o preço.

Exercício 2: Questões sobre ficheiros em C (8 pontos)

11. Uma revista musical organiza, semanalmente, uma pesquisa de opinião sobre a popularidade de discos. Os resultados da pesquisa são guardados num ficheiro de texto, discos.txt, com a seguinte informação por linha: nome do disco, nome do autor, discos vendidos. Escreva um programa que leia esta informação para um vetor de registos e escreva os discos que têm vendas superiores a um certo valor.

Exercício 3: Depuração em C (4 pontos)

12. Faça um documento em MS power point explicando todos os passos para a depuração do programa?
13. Envie este documento pelo email do teu professo r?

Referências do Curso

Livros

- Backes, A. (2013). Linguagem C: Completa e Descomplicada. Campus. Elsevier 1ª edição.
- Cormen, T. H…[et al.] (2002). Algoritmos: Teoria e Prática. Rio de Janeiro. Elsevier 2ª edição.
- Damas, L. (1999). Linguagem C.   Rio de Janeiro. FCA 10ª edição.
- Damas, L. (2001). Linguagem C. Lisboa. FCA.
- Damas, L. (2014). Linguagem C. Lisboa. FCA 24ª edição.
- Guerreiro, P. (2006). Elementos de programação com C. FCA.
- Holzner, S. (1991). C Programming. Brady.
- Jamsa, K. (1989). Microsoft C – Secrets, Shortcuts and Solutions, Microsoft.
- Jackson, M. A. (1975). Principles of Program Design. Academic Press, Inc. Orlando, FL, USA.
- Mizrahi, V. V. (s/d). Treinamento em Linguagem C: curso completo, módulo I. São Paulo. Makron Books.
- Pereira, S. L. (s/d). Linguagem C. Disponível em http://www.ime.usp.br/~slago/slago-C.pdf
- Kernighan, B. W., Ritchie, D. (1988). The C Programming Language -the ANSI edition, Prentice-Hall
- Rocha, A. A (2014). Estruturas de dados e algoritmos em C. Lisboa. FCA 3ª edição.
- Rocha, A. A. (2006). Introdução á programação usando C. FCA.
- Schildt, H. (1995). C – The Complete Reference, McGraw-Hill. 3rd edition.
- Seymour, L. (1986). Data structures. McGraw Hill Companies.
- Schildt., H. (1995). C Completo e Total. São Paulo. São Paulo. Makron Books 3ª edição.
- Tenenbaum, A. A., Langsam, Y., Augenstein. M.J. (1995). Estruturas de dados usando C. São Paulo. Makron Books.
- Vasconcelos, J. B., Carvalho, V. J. (2005). Algoritmia e estruturas de dados: Programação nas linguagens C e JAVA. Lisboa. Centro Atlântico.

Sebentas/tutorias/Apontamentos Aulas

- Dias, A. M. (2017). Introdução à Programação. FCT Nova Lisboa.
- Endereços e Ponteiros. Disponível em http://www.ime.usp.br/~pf/algoritmos/aulas/pont.html
- Operadores. Site da eletrónica de programação. Disponível em: http://www.li.facens.br/eletronica

- Programação estruturada. Site da eletrónica de programação. Disponível em: http://www.li.facens.br/eletronica
- Sampaio, I., Sampaio, A. (s/d). Sebenta Linguagem C. ISEP.
- Santos. J., Baltarejo, P. (2006). Apontamentos de Programação em C/C++. Porto. Departamento de Engenharia Informática. Disponível em: http://www.ebah.pt/content/ABAAAgZ4UAH/sebenta-cpp-03-2006
- Sá, M. R. T (s/d). Apostila de Introdução à Linguagem C, Versão 2.0 unicamp apostilas. Disponível em: http://www.ufjf.br/petcivil/files/2009/02/Apostila-de-Introdu%C3%A7%C3%A3o-%C3%A0-Linguagem-C.pdf
- Sebenta da disciplina de Introdução à programação. Escola Superior de Tecnologia de Setúbal. 2004-2005.
- Sousa, J. F. (s/d). Introdução Manipulação de arquivos em C Estrutura de Dados II. Disponível em: http://www.ufjf.br/jairo_souza/files/2009/12/3-Arquivos-Manipula%C3%A7%C3%A3o-de-arquivos-em-C.pdf
- Vilela, V. V. (1999). 300 ideias para programar computadores. Brasília.

Sede da Universidade Virtual africana

The African Virtual University
Headquarters

Cape Office Park

Ring Road Kilimani

PO Box 25405-00603

Nairobi, Kenya

Tel: +254 20 25283333

contact@avu.org

oer@avu.org

Escritório Regional da Universidade Virtual Africana em Dakar

Université Virtuelle Africaine

Bureau Régional de l'Afrique de l'Ouest

Sicap Liberté VI Extension

Villa No.8 VDN

B.P. 50609 Dakar, Sénégal

Tel: +221 338670324

bureauregional@avu.org

cc BY SA 2017 UVA