Programação I
PRG29002

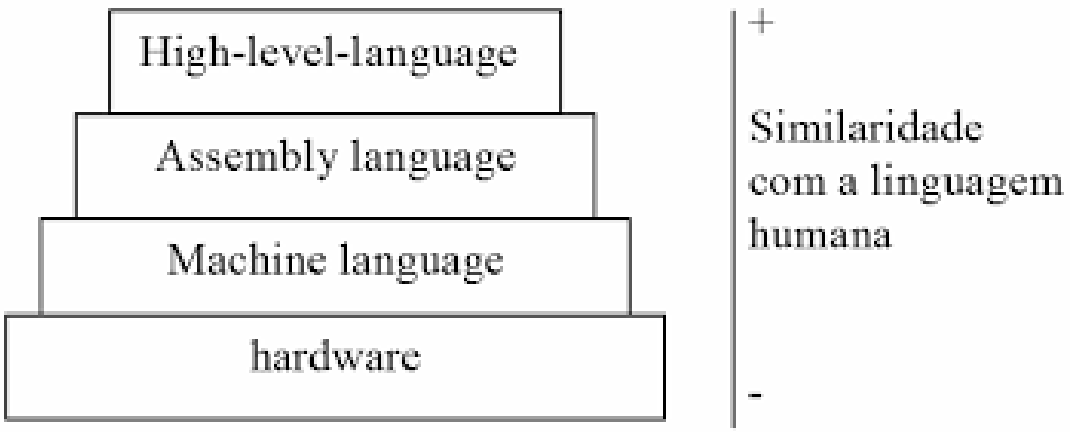
Engenharia de Telecomunicações 2ª Fase

Professor: Cleber Jorge Amaral

2016-2

INSTITUTO FEDERAL
SANTA CATARINA

Níveis de linguagens



Introdução ao C

- Um programa em C é composto por um conjunto de Funções.
- A função pela qual o programa começa a ser executado chama-se “main()”.
- Após cada comando em C deve-se colocar um ; (ponto-e-vírgula).
- É uma linguagem “tipada”, ou seja, os dados precisam ter tipos definidos
- Possui estruturas diversas de fluxo e controle como “if... else”, “Switch case”, etc.

Identificadores

- São os nomes que o programador dá a suas variáveis, constantes e funções
- Deve sempre iniciar com uma letra ou “_” (underscore)
- A partir do segundo caracter pode também conter números
- A linguagem não suporta caracteres especiais como letras acentuadas
- Identificadores não podem ser escritos com espaço, exemplo “buscarCodigo()”, não pode ser escrito como “buscar codigo()”
- A linguagem C é case-sensitive. Por exemplo, as variaveis “numero”, “Numero” e “NUMERO” são endereços diferentes
- Deve ter no máximo 31 caracteres (compatível com TurboC)

Boas práticas

- A identação adequada facilita a compreensão do código
- O uso de nomes auto-explicativos facilita a compreensão e manutencão futura
- É comum variaar maiusculas e minusculas para facilitar a leitura como “QtMedidas”, “ValorMedio”

Preparando o ambiante

- Como precisaremos digitar alguns comandos, vamos utilizar o terminal do linux, abra portanto o terminal
- Por padrão o linux inicia na pasta do usuário, ficará algo assim:

  aluno@sj-redes1-d1:~$, daqui em diante utilizaremos o ~$ para simbolizar a pasta do usuário
- Execute os seguintes comandos para criar e entrar na pasta do usuário:

  ~$ mkdir ExerciciosC

  ~$ cd ExerciciosC
- Observe que agora o terminal exibe algo como aluno@sj-redes1-d1:~/ExerciciosC$

Editando um arquivo

- O arquivo “.c” é o código-fonte de nosso projeto, é onde digitaremos o código na linguagem C
- Depois precisaremos compilar este código para transformá-lo em executável e finalmente poder rodá-lo, para compilar utilizaremos o compilador gcc do linux
- Para editar o arquivo “.c” podemos utilizar qualquer editor como o “gedit” do linux que é bem parecido com o “bloco de notas” do windows.
- Porém como teremos que entrar com linhas de comando para compilar, vamos utilizar um editor chamado pico que roda no terminal do linux

Passo-a-passo criando o OlaMundo.c

- 1) Crie o arquivo através do pico “nome do arquivo.c”

  ~/ExerciciosC$ pico OlaMundo.c

- 2) Digite dentro do arquivo em brando criado o seguinte código:

```
#include <stdio.h>

main()

{

  printf("Olá Mundo!\n");

}
```

- 3) Compile o código (nenhuma linha impressa significa que não deu erro de compilação)

  ~/ExerciciosC$ gcc OlaMundo.c -o OlaMundo

criando o OlaMundo.c (cont.)

- 4) Observe que na sua pasta agora tem dois arquivos um é o código fonte (arquivo .c) e o outro é o executável

  ~/ExerciciosC$ ls -l

- 5) Execute o programa criado

  ~/ExerciciosC$ ./OlaMundo

- 6) Como resultado você deve ver a mensagem na tela:

  Olá Mundo!

- 7) O sistema volta para a linha de comando

  ~/ExerciciosC$

Pratique

- Crie novos arquivos e experimente escrever diversas mensagens de diferentes textos.
- Teste também o uso do “\n”

Variáveis

- Por uma questão de eficiência de uso de memória e processamento o C possui diversos tipos de variáveis, vamos agora trabalhar com alguns deles que servirão para praticamente todas as nossas necessidades
  - char: ocupa 1 byte na memória e varia de -127 a +127
  - int: ocupa 4 bytes e varia de -2.147.483.648 a +2.147.483.647
  - double: ocupa 8 bytes e possui dez dígitos de precisão
  - char[]: esta é o mesmo char descrito acima mas aqui simbolizando uma cadeia/vetor de caracteres (string)
- Apenas para conhecimento neste momento, há outros tipos como short, float e long double e os tipos que não são de precisão podem ainda ser signed ou unsigned

Passo-a-passo criando o exemplo2.c

- 1) Crie o arquivo através do pico “nome do arquivo.c”

  ~/ExerciciosC$ pico UsaVariavel.c

- 2) Digite dentro do arquivo em brando criado o seguinte código:

```
#include <stdio.h>

main()
{
    int x; /* declaração de uma variável inteira */
    x=5;/* atribuindo o valor 5 (constante) a variável x */
    printf ("O valor de x é %d\n",x);
}
```

- 3) Compile e execute

Introdução ao C (continuação)

- C é uma linguagem “compilada”, ou seja, de um código fonte (escrito em C) são gerados códigos de máquina formando um ou mais arquivos executáveis e inteligíveis apenas para o computador
- Há diversos compiladores e estes podem ter algumas diferenças de comportamento e aceitarem diferentes parametrizações
- Um código é compilado para um sistema operacional específico e uma arquitetura de processador, portanto, um código compilado para um S.O. não tem qualquer garantia de funcionamento em outros sistemas. Da mesma forma um código que roda em um PC, não tem qualquer garantia de rodar em outras arquiteturas diversas
- Em oposição ao código compilado temos o código interpretado

Introdução ao C (continuação)

- Sempre que um código fonte é modificado se faz necessário nova compilação para que as modificações façam efeito na execução
- As variáveis que serão utilizadas pelo programa devem ser listadas antecipadamente
- A linguagem C tem um conjunto de palavras reservadas, que não podem ser utilizadas para outro propósito se não o que está definido na estrutura da linguagem
  - Exemplos: break, case, if, for, while, return,...

Introdução ao C (continuação)

- O C permite que trabalhemos com bibliotecas (lib) que são conjuntos de funções que realizam certas tarefas
- Além de podermos criar nossas próprias bibliotecas com funções úteis que podemos reutilizar em vários programas, também podemos nos apropriar de diversas libs já desenvolvidas, sejam padrão ANSI (libc) ou não, desta forma não precisamos “reinventar a roda” e já sair de largada com várias funcionalidades
  - Exemplos: <stdio.h>, <math.h>, <complex.h>, <float.h>, <string.h>, etc. (são 24 padrão ANSI no total)

Passo a passo da compilação de um projeto em C

1) Edição: atividade feita pelo programador
2) Preprocessamento: compilador processa o código e ignorando comentários, fazendo associações de constantes e controle de código através de diretivas especiais de compilação
3) Compilação: criação do código-objeto, é a tradução da linguagem C em linguagem de máquina
4) Linkagem: associação de diferentes código-objeto e bibliotecas
5) Carregamento: carrega o programa em memória
6) Execução: cpu realiza a execução das instruções passo a passo, armazenando os resultados em memórias definidas pelo programa e pilhas de dados para controle

Comentários

- Como vimos podemos incluir no programa fonte textos livres que ajudam na compreensão do código
- Os comentários são ignorados pelo compilador, não se tornam código de máquina
- Para incluir comentários inicie com /* digitando então o comentário aqui e terminando com */
  - Este formato permite que digitemos varias linhas de comentários, normalmente é utilizado para textos mais extensos
- A maioria dos compiladores também aceita o formado //comentário, que serve para incluir um comentário de apenas uma linha, apenas os caracteres depois do // serão ignorados e neste caso o terminador é o sinal de nova linha que normalmente está oculto

Operadores aritméticos

- Operadores aritméticos
  - “+” adição
  - “-” subtração
  - “*” multiplicação
  - “/” divisão
  - “%” resto da divisão
- Por padrão, multiplicações e divisões são operadas antes de somas e subtrações

Uso de parênteses

- Devemos utilizar parênteses para agrupar operações e definir a sequencia mais adequada. O compilador vai sempre resolver o que está dentro dos parênteses primeiro, de “dentro para fora” quando houver mais de um nível
- Exemplos
  - 1+2*3 = 7 é o mesmo que 1+(2*3)
  - (1+2)*3 = 9
  - 1+2*3+4*5 = 27 é o mesmo que 1+(2*3)+(4*5)
  - (((1+2)*3)+4)*5 = 65

Escrevendo mensagens na tela

- A função printf da lib stdio é bastante completa para esta tarefa, permite escrever mensagens com múltiplos argumentos.
- Formato printf (“string de controle”, lista de argumentos);
- Exemplo:
  - printf(“Olá Mundo!\n”);
  - printf(“Digite sua idade:\n”);
  - printf(“Sua idade é: %d”,idade);

Algumas strings de controle
%c caractere
%d decimal
%e notação científica
%f ponto flutuante
%o formato octal
%x hexadecimal
%s cadeia de caracteres
%lf double

Lendo o teclado do usuário

- A função scanf da lib stdio é bastante útil para esta tarefa, ela aguarda que o usuário entre com uma informação e tecle [ENTER] no final.
- Está função é blocante, ou seja, o programa fica parado esperando a entrada de dados para então dar continuidade a execução
- Formato scanf (“string de controle”, lista de argumentos);
- Exemplo:
  - scanf(“%d”,&idade);

Reflexão

- Linguagens compiladas vs interpretadas, qual a diferença?
- Para que servem as bibliotecas? O que é a libc?
- Para que servem e como se utilizam comentários em C?
- Quais os operadores aritméticos que estudamos? Qual a impirtância da precedência?
- Que funções utilizamos para imprimir mensagens e tela e para capturar dados? Qual a lib que contém estas funções?

Operadores relacionais e lógicos

- Relacionais
  - > maior que, ex.: Se (i > j) printf(“i é maior que j”);
  - >= maior ou igual que, ex.: Se (i >= j) printf(“i é maior ou igual a j”);
  - < menor que, ex.: Se (i < j) printf(“i é menor que j”);
  - <= menor ou igual que, ex.: Se (i <= j) printf(“i é menor ou igual a j”);
- Igualdade
  - == igual a, ex.: Se (i == j) printf(“i é igual a j”);
  - != diferente de, ex.: Se (i != j) printf(“i é diferente de j”);

Operadores relacionais e lógicos

- Lógicos
  - && Lógica E (AND), ex.: Se (i > j) && (i > 0) printf(“i é maior que j e positivo”);
  - || lógica OU (OR), ex.: Se (i > j) || (i == 0) printf(“i é maior que j ou é igual a zero”);
  - ! Lógia negação (NOT), ex.: Se !(i > j) printf(“i não é maior que j”);

A declaração “if (expressão) corpo”

- Permite o programa escolher por duas alternativas, executando o procedimento presente no corpo ou não
  - O parênteses é obrigatório
  - A expressão pode conter múltiplos testes
  - “if” se escreve com letras minúsculas
- O corpo com múltiplos comandos deve ficar dentro de {chaves}
- Exemplos:

```
if (i > 0) printf(“i é maior que zero”);
if ((i > 0) && (j == -1)) {
    j = i;
    printf(“o novo valor de j é %d”, j);
}
```

A cláusula “else”

- Permite o programa escolher por duas alternativas, executando apenas o conteúdo do corpo do if ou o conteúdo do do eles

  if (expressão) corpo_if else corpo_else

- Mesmas regras citadas para o if, observe também que os comandos sempre terminam com ;

- Exemplos:

```
if (i > j)
    max = i;
else
    max = j;
```

```
if (i > j)
    if (i > k) max = i; else max =
    k;
else
    if (j > k) max = j; eles max =
    k;
```

if em cascata

- É possível realizar séries de testes parando assim que uma for verdadeira.
- Exemplo:

```
if ((i >= 0) && (i < 6))
    printf("Conceito insuficiente");
else if ((i >= 6) && (i < 9))
    printf("Conceito suficiente/proficiente");
else if ((i >= 9) && (i <= 10))
    printf("Conceito excelente");
else
    printf("Conceito inválido");
```

Boas práticas

- Idente adequadamente o código para facilitar a leitura
- Utilize parênteses mesmo em testes que poderia ser dispensado para facilitar o entendimento
- Procure utilizar {chaves} em declarações onde haja if dentro de if para evitar confusão sobre qual é o if dono do else (problema do eles pendente)

```
if (y != 0)
{
    if (x != 0) {
        result = x / y;
    } else {
        printf(“erro: y é zero”);
    }
}
```

Expressões condicionais

- Há uma forma mais compacta de se declarar uma condição que vai funcionar como um if... else

  expressão1 ? expressão3 : expressão3

- Exemplo de substituição:

```
if (i > j)
    return i;
else
    return j;
```

Equivalente: return (i > j) ? i : j;

Usando printf(“%d”, i > j ? i : j);

Comando switch

- Controle de fluxo para testes com números inteiros (ponto flutuante e strings não podem ser utilizados)
- Não é necessário incluir {chaves} os comandos já são tratados em blocos
- Não se pode duplicar a constante
- A cláusula defaut não é obrigatória
- Estrutura:

```
switch ( expressão ) {
    case constante1: corpo1
    case constante2: corpo2
    default: corpo_default
}
```

Comando switch (exemplo)

- Exemplo:

```
switch ( nota ) {
    case 4:
    case 3:
    case 2:
    case 1: printf(“passou!”);
        break;
    case 0: printf(“não passou!”);
        break;
    default: printf(“nota inválida”);
        break;
}
```

Boas práticas

- Caso a omissão do break é proposital, deixe um comentário descrevendo isso
- Utilize o default para tratamento de situações inesperadas

Obrigado pela atenção e participação!