Apresentação

Estas Notas de Aula, visam aumentar a produtividade dos alunos nas aulas teóricas, evitando a cópia das teorias expostas. Grande parte dos exemplos analisados em sala de aula e enunciados de exercícios constam nesta apostila, além da resolução de alguns destes. Os EC (exercícios de classe), serão normalmente baseados nos exercícios complementares de cada Aula a serem desenvolvidos preferencialmente no laboratório.

Habitualmente antes de resolvermos exemplos ou exercícios, elaboraremos o algoritmo, que nada mais é que uma seqüência de operações cuja execução produz um resultado que é a resposta de um problema proposto.

Um programa de computador nada mais é que a codificação de um algoritmo numa linguagem de programação. Linguagens como C, Pascal, BASIC, ALGOL, Clipper, COBOL, etc., são chamadas de procedurais, devido ao fato das instruções serem executadas de forma seqüencial, enquanto que as linguagens baseadas no conceito de eventos como C++, Visual BASIC, Visual Objects, utilizam outra estratégia de programação (Programação Orientada ao Objeto), a ser vista em outro módulo do curso (OOP), em C utilizaremos a **metodologia estruturada**.

Os algoritmos podem ser estruturados ou não, conforme exemplificamos a seguir no cálculo do máximo divisor comum entre dois números inteiros positivos, com operações elementares:

Linear

```
Leia m,n
(1)     se n = 0 então
imprima m
pare
k <- m - Int(m / n) * n
m <- n
n <- k
vá para (1)
```

Em Quick BASIC teríamos Em BASICA teríamos

```
   input m : input n          10 input m : input n
10 if n = 0 then              20 if n <> 0 then 50
      print m                 30 imprima m
   end if                     40 end
   k = m - Int(m / n) * n     50 k = m - Int(m / n) * n
   m = n                      60 m = n
   n = k                      70 n = k
   goto 10                    80 goto 20
```

O Algoritmo de Euclides anteriormente apresentado , apesar de muito simples, executado normalmente por qualquer criança de primeiro grau, ganha contornos de aparente dificuldade quando transcrito para o GW-BASIC, um dialeto BASIC, que exige numeração de linhas, que nos obrigou a alterar um pouco a estratégia de resolução do problema. Já a versão em Quick BASIC, poderia ser transcrita por qualquer pessoa com um mínimo de conhecimento em algoritmos e BASIC.

Estruturado

```
Inteiros m,n
Leia m,n
enquanto n <> 0 faça
k <- m - Int(m / n) * n
```

```
m <- n
n <- k
imprima m
pare
```

Em C teríamos Em Clipper teríamos

```
main() input to a

{ input to b

int m,n,k; do while n <> 0

scanf("%d",&m); k = m - Int(m / n) * n

scanf("%d",&n); m = n

while (n != 0) { n = k

k = m - m / n * n; enddo

               m = n;                    ? m
              n = k;
  }
printf("%d",m);
}
```

Nas Linguagens estruturadas a representação fica idêntica, quer em C quer em Clipper. Ficaria a mesma forma em Quick BASIC, Pascal ou COBOL Estruturado, daí a preferência na abordagem dita estruturada. No decorrer deste curso nos deteremos detalhadamente no estudo dos algoritmos estruturados e analisaremos alguns (poucos, porém importantes) casos onde a programação linear é mais adequada que a estruturada.

Simulação

Por melhor que seja o conhecimento do Analista/Programador, este só poderá ter certeza que sua "estratégia" foi bem sucedida após testar o programa num computador.

Bons programadores tem poucas surpresas quando testam seus programas, fazendo normalmente umas poucas correções para que o programa funcione de maneira adequada.

Programadores iniciantes criam uma estratégia (algoritmo) muitas vezes ineficiente e normalmente "correm" para o computador visando testar o programa e acabam por perder um tempo enorme. Não é raro ouvirmos de iniciantes a frase "Nunca conseguirei fazer um programa ...". Certamente, não conseguirá mesmo, caso não tenha uma estratégia definida.

Imagine uma "tarefa" muito simples, como por exemplo fritar um ovo. Caso você não soubesse "operar" o fogão provavelmente não conseguiria acendê-lo. Se nunca tivesse visto alguém quebrar um ovo, provavelmente não conseguiria fazê-lo sem perder parte de seu conteúdo. A fritura seria algo desastroso, caso você desconhecesse a utilidade do óleo. E o sabor seria frustrante, caso o tempero utilizado fosse açúcar.

O Programador iniciante que não simula seu algoritmo, se compara ao cozinheiro desastrado descrito acima. E como simular?

Basta "agirmos" como se fossemos o próprio computador, ou seja devemos "fingir" que nosso raciocíneo é baseado no **conteúdo de variáveis**. De fato, usualmente antes de tomarmos alguma decisão analisamos uma série de fatores (as tais variáveis dos programas). Desta forma, caso fizermos uma análise apurada sobre os conteúdos das variáveis, poderemos "descobrir" a função de cada programa.

Primeiramente vamos resolver o problema como aprendemos no primeiro grau.

Supondo M = 120 e N = 28.

Dividendo	=	120	28	8	
Divisor	=	28	8	4	MDC
Quociente	=	4	3	2	
Resto	=	8	4	0	

Simulação:

M	N	K	Impresso	Instruções	Comentários
120	28			Declarações e Leituras	
		8		Enquanto n <> 0 k = m - m / n * n	Cálculo do Resto
28	8			m = n; n = k	Novos Divisor e Dividendos
		4		Enquanto n <> 0 k = m - m / n * n	Voltando ao Teste Cálculo do Resto
4	0			m = n; n = k	Novos Divisor e Dividendos
				Enquanto n <> 0	N é Zero
			4		Achou MDC!

Observação Final: Obviamente caso você se sinta seguro a resolver diretamente o problema na linguagem diretamente no computador, faça isto. Porém se surgirem dificuldades, não se esqueça de fazer o algoritmo e a simulação. Lembre-se do cozinheiro desastrado, que se tivesse consultado uma simples receita (todo algoritmo não passa de uma receita sofisticada), poderia ao menos fritar o ovo!

Exercícios (Aula 01L)

Nunca esqueça que os comandos da linguagem C devem ser escritos SEMPRE em letras minúsculas!

Apresentação do Interpretador Classic C.

- 1- Elabore programa que apresente mensagem alô mundo!
- 2- Elabore programa para o cálculo do máximo divisor comum entre 2 números.
- 3- Elabore programa que imprima o seu nome.

Linguagem C

Objetivos:

- Tornar o Aluno apto a programar em C (pequenos programas).
- Conhecimento do C Clássico (Classic C).
- Conhecimento do Turbo C (compilador voltado ao aprendizado e também usado comercialmente).

Estrutura do Curso:

- Aulas Teóricas: Exposição, Exemplos e Exercícios.
- Aulas Práticas: Processar Programas Prontos e Elaboração de Exercícios.

Introdução

Desenvolvida nos laboratórios Bell na década de 70, a partir da Linguagem B (criada no final dos anos 60 por Ken Thompson), que foi reformulada por Brian Kernighan e Dennis M. Ritchie e posteriormente renomeada para C.

Podendo ser considerada como uma linguagem de médio nível, pois possui instruções que a tornam ora uma linguagem de alto nível e estruturada como o Pascal, se assim se fizer necessário, ora uma linguagem de baixo nível pois possui instruções tão próximas da máquina, que só o Assembler possui.

De fato com a linguagem C podemos construir programas organizados e concisos (como o Pascal), ocupando pouco espaço de memória com alta velocidade de execução (como o Assembler). Infelizmente, dada toda a flexibilidade da linguagem, também poderemos escrever programas desorganizados e difíceis de serem compreendidos (como usualmente são os programas em BASIC).

Devemos lembrar que a linguagem C foi desenvolvida a partir da necessidade de se escrever programas que utilizassem recursos próprios da linguagem de máquina de uma forma mais simples e portável que o assembler.

Uma análise superficial dos programas escritos em C e Clipper (Aula 01 - Algoritmos Estruturados), nos permite perceber que a linguagem C supera em muito em dificuldade o programa análogo em Clipper. Ora, então porque não desenvolvermos programas somente em Clipper?

A inúmeras razões para a escolha da linguagem C como a predileta para os desenvolvedores "profissionais". As características da Linguagem C servirão para mostrar o porquê de sua ampla utilização.

Características da Linguagem C

- Portabilidade entre máquinas e sistemas operacionais.
- Dados compostos em forma estruturada.
- Programas Estruturados.
- Total interação com o Sistema Operacional.
    - Código compacto e rápido, quando comparado ao código de outras linguagem de complexidade análoga.

Aplicações Escritas em C

Atualmente, nos USA, C é a linguagem mais utilizada pelos programadores, por permitir, dadas suas características, a escrita de programas típicos do Assembler, BASIC, COBOL e Clipper, sempre com maior eficiência e portabilidade, como podemos constatar pelos exemplos abaixo relacionados:

- Sistema Operacional: UNIX (Sistema Operacional executável em micro computadores e em mainframes).
- Montadores: Clipper (O utilitário de banco de dados mais usado no Brasil).
- Planilhas: 1,2,3 e Excel (A planilha eletrônica com maior volume de vendas mundial).
- Banco de Dados: dBase III, IV e Access (o gerenciador de base de dados mais utilizado no mundo).
- InfoStar: O Editor de Texto mais utilizado nos USA no Sistema Operacional UNIX.
- Utilitários: FormTool (Editor de formulário mais vendido no mundo).
- Aplicações Gráficas: Efeitos Especiais de filmes com Star Trek e Star War.
- Linguagens como o Power Builder e o Visual Basic, respectivamente as linguagens mais utilizadas nos EUA e no Brasil.

No Brasil utilizada por empresas especializadas na elaboração de vinhetas e outros efeitos especiais.

C comparado a outras linguagens

Devemos entender Nível Alto como sendo a capacidade da linguagem em compreender instruções escritas em "dialetos" próximos do inglês (Ada e Pascal, por exemplo) e Nível Baixo para aquelas linguagens que se aproximam do assembly, que é a linguagem própria da máquina, compostas por instruções binárias e outras incompreensíveis para o ser humano não treinado para este propósito. Infelizmente, quanto mais clara uma linguagem for para o humano (simplicidade >) mais obscura o será para a máquina (velocidade <).

Observemos o esquema a seguir:

Nível Baixo Nível Médio Nível Alto

VELOCIDADE CLAREZA

Macro Assembler Basic Ada

Assembler C Fortran Pascal

Forth COBOL MODULA-2

Antes da linguagem C tornar-se um padrão de fato (meados de 1.988, nos USA), tínhamos aproximadamente, o seguinte perfil de mercado:

- Aplicações de Banco de Dados
- Mainframe: COBOL e gerenciadores
- Micros: dBase, Clipper e BASIC e gerenciadores como Btrieve
- Aplicações Gráficas: Pascal.
- Aplicações Científicas: FORTRAN e Pascal.
- Utilitários, Sistemas Operacionais e Compiladores: Assembler.

A chegada de poderosos compiladores C (Borland, Microsoft e Zortech-Symantec), revolucionou totalmente estes conceitos pois passou a permitir a construção de praticamente qualquer tipo de aplicação na Linguagem C, normalmente mais rápidas do que na linguagem original e portável entre os diversos ambientes ("roda" em DOS, UNIX, etc. com poucas mudanças). Devemos entender no entanto, que apenas temos uma *relativa portabilidade*, pois a verdadeira portabilidade depende necessariamente da implementação do sistema operacional, necessariamento aberto, o que não existe fora do mundo Unix.

Quadro de características de linguagens:

Linguagens Características Ideais	Assembler	BASIC Pascal	Clipper	COBOL	C
Executáveis Curtos	ótimo	fraco	péssimo	fraco	ótimo
Executáveis Rápidos	ótimo	bom	razoável	fraco	bom
Portáveis	péssimo	bom	ótimo	ótimo	bom
Manipulação de Bits	ótimo	razoável	péssimo	fraco	ótimo

O quadro anterior, deixa claro o porquê da revolução causada pela Linguagem C, dados os inúmeros pontos fortes da linguagem e a inexistência de pontos fracos da mesma. Não devemos concluir apressadamente que poderemos desenvolver tudo em C e abandonarmos todas as outras linguagens, pelos seguintes motivos:

- Alta base de programas escritos em Assembler, COBOL, BASIC, Pascal.
- Conhecimento amplo de linguagens concorrentes como COBOL, Clipper e BASIC.
- Melhor adaptação de alguma linguagem para tarefa específica como Gráficos (Pascal), Inteligência Artificial (Prolog, LISP), matemáticas (Pascal e FORTRAN), aplicações comerciais (COBOL, Clipper e BASIC).
- As versões atuais das linguagens BASIC, C, Clipper, Pascal e COBOL, estão muito mais semelhantes do que o eram ao tempo em que este quadro foi elaborado, portanto na prática muitas vezes este quadro, meramente teórico, pode tornar-se inaplicável.

A médio prazo, podemos afirmar que a linguagem C deverá ir desalojando as outras linguagens podendo até mesmo tornar-se um padrão de direito, porém devemos lembrar que algumas linguagens foram propostas para isto (Algol, PL/1) e não só não conseguiram atingir seus objetivos como praticamente desapareceram.

A tabela anteior não esgota todas as possibilidades, pois alguns dialetos de linguagem destacam recursos que na maior parte das implementações da linguagem não são muito fortes. O inverso também pode ocorrer, de modo que as vezes uma implementação corrige algum defeito da linguagem (por exemplo aumenta drasticamente seu desempenho) e piora outras (portabilidade inexistente). Este problema afeta sensivelmente o BASIC e as implementações xBase (Fox, dBase e Clipper) e em menor grau o Pascal e a própria linguagem C.

Nota de Revisão: Apesar da Linguagem C ter crescido de cerca de 3% das instalações no Brasil, quando da escrita da primeira versão desta apostila (1989), para cerca de 20% das instalações atualmente (1997), o número atual esconde uma realidade que deve ser mencionada. Muitas das instalações usuárias da linguagem C, também são usuárias do VB (Visual Basic) da Microsoft.Desta forma, a linguagem C está presente normalmente para construção de rotinas mais sofisticadas impossíveis de implementação em VB. Contrariamente as empresas usuárias do Delphi (Borland) e do Power Builder (Sybase), geralmente estão abrindo mão em suas instalações da Linguagem C, tendo em vista a capacidade destas implementações em realizar tarefas geralmente destinadas a linguagem C. A linguagem Java, que é claramente baseada em C, também começa a penetrar em áreas onde C reinava abosulta.

Exemplo: Desejo desenvolver um processador de textos, a partir do início (não possuo qualquer código de programa pronto). Qual linguagem escolherei?

Solução:

Características Ideais	**Classe**	**Valor**
Executáveis Curtos	Não Importa	0
Executáveis Rápidos	Fundamental	3
Portáveis	Desejável	1
Simplicidade	Desejável	1
Manipulação de Bits	Necessário	2

Onde 0- Não Importa, 1- Desejável, 2- Necessário, 3- Fundamental

Juntando as Características do Projeto as Características da Linguagem, temos:

Linguagens Características Ideais	Pt	Assembler	Basic Pascal	Clipper	COBOL	C
Executáveis Curtos	0	7x0=0	1x0=0	0x0=0	0*0=0	7*0=0
Executáveis Rápidos	3	7*3=21	5x3=15	3x3=9	1x3=3	5x3=15
Portáveis	1	0*1=0	3*1=3	1*1=1	5*1=5	7*1=7
Simplicidade	1	0*1=0	5*1=5	7*1=7	7*1=7	5*1=5
Manipulação de Bits	2	7*2=14	3*2=5	1*2=2	0*2=0	7*2=14
		35	27	19	15	41

Onde 0- Péssimo, 1- Fraco, 3- Razoável, 5- Bom e 7- Ótimo

Resposta: Linguagens Adequadas C ou Assembler. Caso se dê prioridade a portabilidade C é ainda mais adequada. O número talvez surpreendentemente alto para Pascal e BASIC serve para demonstrar o porquê da existência de Editores feitos total ou parcialmente em Pascal. O fraco desempenho do Clipper e do COBOL era esperado pois são linguagens mais apropriadas para manipulação de dados. Na prática devemos observar a existência de dois critérios também importantes, que são a existência de programadores especialistas na linguagem adequada e a possibilidade de junção de mais de uma linguagem numa implantação qualquer.

Exercícios

1- Desejo criar um vírus de computador, qual a linguagem ideal?

2- Desejo criar um utilitário de Banco de Dados, semelhante a dBase. Qual a linguagem ideal?

3- Supondo ter uma empresa concluído ser ideal desenvolver internamente sua folha de pagamento. O Gerente de Informática, atento ao movimento do Mercado definiu a escrita dos programas em C, visando poder migrar a aplicação entre ambientes. Supondo ainda que a equipe desconhece totalmente a Linguagem C, mas que é bastante gabaritada em COBOL, em sua opinião nosso Gerente foi feliz em sua decisão? Justifique.

Exercícios (Aula 02L)

- 1- Leia seu nome e o imprima.
- 2- Leia dois números e apresente seu produto.
- 3- Leia três números e apresente sua média.

C- Uma visão Geral-Intruções de entrada e de saída.

Toda linguagem de programação de alto nível suporta o conceito de "Tipo de Dado", que define um conjunto de valores que a variável pode armazenar, e os tipos mais comuns encontrados nas linguagens de programação, ou seja, inteiro, real e caractere. Diferentemente do Pascal que é fortemente tipada onde a mistura entre um número inteiro e um real podem causar erros, C suporta livremente tipos caracteres e inteiros na maioria das expressões!

Em geral os compiladores C realizam pouca verificação de erros em tempo de execução, verificação de limites de matrizes ou compatibilidade entre tipos de argumentos, cabendo esta responsabilidade ao programador. Assim você decide onde uma verificação de erro é ou não mais necessário.

Por ser capaz de manipular bits, bytes e endereços, C se adapta bem a programação a nível de sistema. E tudo isto é realizado por apenas 43 palavras reservadas no Turbo C, 32 nos compiladores padrão ANSI e 28 no C Padrão. Como curiosidade, o IBM BASIC que é um interpretador BASIC com fins puramente educativos tem 159 comandos.

Como curiosidade apresentamos a seguir quadro com as palavras reservadas do C Padrão.

Auto	double	if	static
break	else	int	struct
case	entry	long	switch
char	extern	register	typedef
continue	float	return	union
default	for	sizeof	unsigned
do	goto	short	while

Fundamentos de C

Em se tratando de programação a expressão "Se você não souber, jamais irá aprender." é uma verdade absoluta. É muito comum ouvirmos queixas do tipo "nunca conseguirei escrever um programa", ou "só sendo louco ou gênio para descobrir a solução". Estas expressões geralmente são ditas por estudantes que desconhecem o fato de que cada elemento da linguagem (comandos, funções) não existe sozinho, mas somente combinados a outros elementos.

Desta forma a orientação que adotaremos neste início do curso se deterá mais na compreensão geral do programa, do que a análise detalhada de cada comando ou função utilizada. De fato apresentaremos alguns comandos fundamentais para a escrita de programas básicos e apenas nos utilizaremos de sua sintaxe mais elementar (posteriormente estudaremos cada um deles mais detidamente), construiremos os primeiros programas do curso.

- Exemplo 1: Programa mostra a idade.

```
/* Exemplo Idade */
main()
```

```
{
int idade;
idade = 40;
printf("Sua idade e' %d anos. \n", idade);
}
```

Este programa simplesmente imprime "Sua idade e' 40 anos." saltando uma linha (/n) em seu término.

Comandos Básicos - 1ª. Parte

Instruções de Entrada e Saída

O objetivo de escrevermos programas é em última análise, a obtenção de resultados (Saídas) depois da elaboração de cálculos ou pesquisas (Processamento) através do fornecimento de um conjunto de dados ou informações conhecidas (Entradas).

Para que nosso programa possa receber dados e alocá-los em variáveis, que serão responsáveis por armazenar as informações iniciais, nossa linguagem deverá conter um conjunto de instruções que permitam ao operador interagir com o programa fornecendo os dados quando estes forem necessários.

scanf()

Uma das mais importantes e poderosas instruções, servirá basicamente para promover leitura de dados (tipados) via teclado.

Sua forma geral será: scanf("string de controle", lista de argumentos);

Posteriormente ao vermos sua sintaxe completa, abordaremos os recursos mais poderosos da <string de controle>, no momento bastará saber que:

- %c - leitura de caracter
- %d - leitura de números inteiros
- %f - leitura de números reais
- %s - leitura de caracteres

A lista de argumentos deve conter exatamente o mesmo número de argumentos quantos forem os códigos de formatação na <string de controle>. Se este não for o caso, diversos problemas poderão ocorrer - incluindo até mesmo a queda do sistema - quando estivermos utilizando programas compilados escritos em C. Felizmente ao utilizarmos o Classic C, apenas uma mensagem de erro será apresentada, para que possamos corrigir o programa sem outros inconvenientes.

Cada variável a ser lida, deverá ser precedida pelo caracter &, por razões que no momento não convém explicarmos, mas que serão esclarecidas no decorrer do curso. Para seqüência de caracteres (%s), o caracter & não deverá ser usado.

- Exemplo: Programa para ler e mostrar uma idade

```
/* Exemplo Le e Mostra Idade */
main()
{
int idade;
char nome[30];
printf("Digite sua Idade: ");
scanf("%d",&idade);
printf("Seu Nome: ");
scanf("%s",nome);    /* Strings não utilizar '&' na leitura */
printf("%s Sua idade e' %d anos. \n", nome, idade);
}
```

Laboratório (Aula 03L)

- 1- Leia o número e o nome dos elementos do grupo e os apresente.
- 2- Leia o nome e as notas de um aluno. Apresente seu nome e sua média.
- 3- Leia uma letra, um número inteiro, um número com casas decimais e uma string, depois os apresente.

Instruções de entrada e de saída (continuação)

O Comando printf, usado anteriormente, segue o mesmo padrão de scanf(), porém é destinado a apresentação dos dados, enquanto aquele destina-se a leitura dos dados.

printf()

É outro dos mais poderosos recursos da linguagem C, printf() servirá basicamente para a apresentação de dados no monitor.

Sua forma geral será: printf("string de controle", lista de argumentos);

Necessariamente você precisará ter tantos argumentos quantos forem os comandos de formatação na "string de controle". Se isto não ocorrer, a tela exibirá sujeira ou não exibirá qualquer dado.

Os caracteres a serem utilizados pelo printf() em sua <string de controle>, no momento serão os mesmos de scanf().

- Exemplo: Dado um número, calcule seu quadrado.

```
main()
{
int numero;
printf("Digite um Numero: ");
scanf("%d",&numero);
printf("O %d elevado ao quadrado resulta em %d. \n",                    numero,numero*numero);
}
```

Funções em C

Conceitualmente, C é baseada em blocos de construção. Assim sendo, um programa em C nada mais é que um conjunto de funções básicas ordenadas pelo programador. As instruções printf() e scanf(), vistas anteriormente, não fazem parte do conjunto de palavras padrões da linguagem (instruções), pois não passam elas mesmas de funções escritas em C! Esta abordagem, permite a portabilidade da linguagem, pois seus comandos de entrada e saída, não são parte do conjunto básico da linguagem, livrando-a desta forma dos problemas de suporte aos diversos padrões de vídeos, teclados e sistemas operacionais existentes.

Cada função C é na verdade uma sub-rotina que contém um ou mais comandos em C e que executa uma ou mais tarefas. Em um programa bem escrito, cada função deve executar uma tarefa. Esta função deverá possuir um nome e a lista de argumentos que receberá. As funções em C são muito semelhantes as usadas no Pascal, com a diferença que o próprio programa principal é apenas uma função que se inicia com a palavra reservada main() podendo receber parâmetros diretamente do DOS, por exemplo.

- Exemplo: Programa principal chamando função alo.

```
main()
{
alo();
}
alo()
```

```
{
printf("Alô!\n\n");
}
```

Retomemos o exemplo do cálculo de um número elevado ao quadrado.

- Exemplo: Quadrado com função

```
main()
{
int num;
printf("Digite um numero: ");
scanf("%d",&num);
sqr(num);   /* sqr recebe "num" do programa principal */
}
sqr()
int x;  /* x é um "parâmetro" recebido do programa principal
no caso x "vale" o conteúdo de num */
{
printf("%d ao quadrado e' %d ",x,x*x);
}
```

Nota: O argumento simplesmente é o valor (em "num") digitado no programa principal (em scanf) e enviado a função sqr.

Um conceito importante e normalmente confundido é a diferença conceitual entre "argumento" e "parâmetro" que em resumo pode ser definido da seguinte forma: "Argumento" se refere ao *valor* que é usado para chamar uma função. O termo "Parâmetro" se refere à variável em uma função que recebe o valor dos argumentos usados na função. A distinção que deve ser compreendida é que a variável usada como argumento na chamada de uma função não tem nenhuma relação com o parâmetro formal que recebe o valor dessa variável.

- Exercício: Passagem de variáveis entre rotinas.

```
int x;
main()
{
int a;
printf("Digite um valor: ");
scanf("%d",&a);
x = 2 * a + 3;
printf("%d e %d",x,soma(a));
}

soma(z)
int z;
{
x = 2 * x + z;
return(x);
}
```

Laboratório (Aula 04L)

Retome os exercícios das Aulas 1L, 2L e 3L refazendo-os usando funções.

Utilize os caracteres abaixo para apresentação de forma mais sofisticada dos resultados.

Utilize a função cls(), disponível no Classic C para apagar a tela.

Observemos o Quadro de Operadores Especiais suportados por printf()

Código Significado

\b Retrocesso (BackSpace)

\f Salto de Página (Form Feed)

\n Linha Nova (Line Feed)

\r Retorno do Carro (cr)

\t Tabulação Horizontal (TAB)

\' Caracter com apóstrofo

\0 Caracter Nulo ou Fim de String (Seqüência)

\x Representação de byte na base hexadecimal

Exemplo: printf("\x41"); causa a impressão da letra A na tela.

Tomada de decisão

- Comandos Básicos - 2ª. Parte
- Análogo a outras linguagens, sua forma geral será

- *if <condição>*
- *<comando>;*
- *else*
- *<comando>;*

- Exemplo 1: Programa Adulto, Jovem ou Velho.

```
main()
{
int i;
printf("Digite sua idade: ");
scanf("%d",&i);
if (i > 70)
        printf("Esta Velho!");
else
        if (i > 21)
                printf("Adulto");
        else
                printf("Jovem");
}
```

- Observação: A expressão avaliada, deverá obrigatoriamente estar entre parênteses.

Exemplo 2: Maior entre três números

```
main()
{
int a,b,c;
cls();
printf("Digite o 1° Número: ");
scanf("%d",&a);
printf("\nDigite o 2° Número: ");
scanf("%d",&b);
printf("\nDigite o 3° Número: ");
scanf("%d",&c);
if (a > b)
        if (a > c)
                        printf("\nO Maior é %d",a);
        else
                        printf("\nO Maior é %d",c);
```

```
else
        if (b > c)
                printf("\nO Maior é %d",b);
        else
                printf("\nO Maior é %d",c);
}
```

Exemplo 3: Maior entre três números (Segunda Solução)

```
main()
{
int a,b,c,d;
cls();
printf("Digite o 1° Número: ");
scanf("%d",&a);
printf("\nDigite o 2° Número: ");
scanf("%d",&b);
printf("\nDigite o 3° Número: ");
scanf("%d",&c);
if (a > b)
        d = a;
else
        d = b;
if (c > d)
        printf("\nO Maior é %d",c);
else
        printf("\nO Maior é %d",d);
}
```

- Exemplo 4: Dados 2 números apresente-os ordenados.

```
main()
{
int a,b,t;
cls();
printf("Digite o 1° Número: ");
scanf("%d",&a);
printf("\nDigite o 2° Número: ");
scanf("%d",&b);
if (a < b) {
        t = a;
        a = b;
        b = t;
        }
printf("\nOrdenados: %d e %d ",b,a);
}
```

Contagem

Um dos grandes benefícios dos sistemas de processamento de dados está em sua confiabilidade (precisão nos cálculos) e rapidez (infinitamente superior ao ser humano), desta forma é ideal para processamento de elevado número de operações **repetitivas**. O processamento de uma Folha de Pagamentos, a Emissão de Notas Fiscais, a Geração de Estatísticas de Faturamento, são típicas tarefas a serem realizadas por processamento eletrônico de dados.

Todas as linguagens importantes dispõe de recursos destinados a forçar a repetição de execução de um determinado número de instruções. Diferentemente do BASIC e do Pascal, o "loop" *for* é a instrução mais poderosa na criação de estruturas de repetição. Neste momento, abordaremos apenas sua sintaxe simplificada, que lembra bastante a sintaxe do FORTRAN (comando DO) e semelhante àquelas linguagens citadas anteriormente.

Sua forma mais simples é:

for (<início>;<condição>;<incremento>) comando;

- Exemplo 1: Contagem de 1 a 100 ficaria

```
main()
{
int cont;
for (cont = 1; cont <= 100; cont++)
        printf("%d",cont);
}
```

- Exemplo 2: Elaborar programa que imprima a tabuada de um número dado.

```
main()
{
int cont,num;
printf("Digite um Numero: "); scanf("%d",&num);
for (cont = 0; cont <= 10; cont++)
        printf("%2d * %2d = %2d \n",num,cont,num * cont);
}
```

Nota: O número '2' antes do 'd' causa a representação em vídeo de 2 casas, permitindo o alinhamento da tabuada!

Strings (Seqüências)

Em C uma string é simplesmente uma matriz de caracteres finalizada por um código ASCII zero. Sua declaração será:

char str[<inteiro>];

onde a variável str poderá conter no máximo o número de letras igual ao especificado. Por exemplo se declararmos

char a[80];

A variável "a" poderá conter 80 elementos onde a primeira letra estará em a[0] e a octogésima em a[79].

Algo muito importante a ser lembrado é que C não verifica comprimento de strings como o BASIC ou Pascal. A forma mais simples de evitar-se erros é que a string seja suficientemente grande para conter o que você quiser colocar nela. Um caractere zero binário é colocado no final de sua string, portanto se você digitar a palavra alô para uma variável chamada "saudar", esta deverá ter sido declarada no mínimo como:

"char saudar[03];".

A l ô \0

1 2 3 4

- Exemplo 3: Apresente um nome digitado pelo usuário.

```
main()
{
int cont;
char nome[10];
printf("Digite um Nome: "); scanf("%s",nome);
for (cont = 0; cont <= 10; cont++)
        printf("%c",nome[cont]);
printf("\n\n%s",nome);
}
```

Laboratório (Aula 05L)

- 1- Dados 3 números, imprima o maior deles e o menor deles.

- 2- Dados 3 números, imprima a média destes números. Verifique se todos são positivos. Caso algum número seja negativo, indique ao usuário que a média não será calculada.
- 3- Elabore programa que leia "n" números digitados e apresente sua média.
- 4- Elabore programa que imprima a média de "n" números, excluindo o menor deles.
- 5- Dados "n" números apresente a média entre o maior e o menor número da seqüência fornecida.

Tipos de dados

No momento dispomos de conhecimento para elaboração de programas básicos para construção de pequenos programas, pois conhecemos instruções de entrada de dados (scanf), de saída (printf), tomada de decisão (if) e de contagem (for).

Veremos a seguir Tipos de Dados da linguagem C, variáveis, operadores, e demais instruções básicas. Devemos procurar compreender a utilidade das declarações que serão exibidas a seguir, relacionando estes recursos com os exemplos e exercícios vistos anteriormente.

Semelhante ao BASIC, Pascal e COBOL, a linguagem C necessita que todas as variáveis tenham seus tipos definitos. C aceita tipos básicos (caractere, inteiro, ponto flutuante, dupla precisão e sem valor) e modificadores (sinal,sem sinal, longo e curto) que podem alterar os tipos básicos. Observemos as tabelas abaixo para melhor entendimento:

Tabela de Tamanhos e Escala de Tipos Básicos

Tipo	Extensão	Escala Numérica em bits
char	8	0 a 255
int	16	-32768 a 32767
float	32	3.4E-38 a 3.4E+38
double	64	1.7E-308 a 1.7E+308
void	0	sem valor

Tabela com Combinações possíveis de Tipos Básicos e seus Modificadores

Tipo Extensão Escala Obs.

char 8 -128 a 127

unsigned char 8 0 a 255

signed char 8 -128 a 127 ver char

int 16 -32768 a 32767

unsigned int 16 0 a 65535

signed int 16 -32768 a 32767 ver int

short int 16 -32768 a 32767 ver int

unsigned short int 16 0 a 65535 ver unsigned int

signed short int 16 -32768 a 32767 ver int

long int 32 -2147483648 a 2147483647

signed short int 32 -2147483648 a 2147483647 ver long int

unsigned short int 32 0 a 4294967295

float 32 3.4E-38 a 3.4E+38

double 64 1.7E-308 a 1.7E+308

long double 64 1.7E-308 a 1.7E+308 ver double

Nota: Alguns modificadores resultam em combinações exatamente iguais a de outras declarações. Algumas declarações apresentadas acima não são suportadas pelo Classic C, notadamente aquelas com 64 bits, só disponíveis nas mais potentes implementações de compiladores C.

- Exemplo 1: Mesmo número com 2 representações diferentes e Erro.

```
main()
{
int a;
printf("Digite um numero: ");
scanf("%d",&a);
printf("%d %5d %f",a,a,a);
}
```

- Para a = 67, teremos:" 67 67 0.000000"! Observe que a representação em ponto flutuante foi truncada!

- Exemplo 2: Mesmo número com 2 representações diferentes Corretas.

```
main()
{
float a;
printf("Digite um numero: ");
scanf("%f",&a);
printf("%f %e",a,a);
}
```

Simulando obtemos:

Digite um numero: 65

65.000000 6.500000E+01

Exemplo 3: Caracter sendo tratado como número, sem qualquer ERRO de execução ou de compilação.

```
main()
{
int a;
printf("Digite um numero: ");
scanf("%d",&a);
printf("%d %u %c",a,a,a);
}
```

- Para a = 67, teremos 67 67 C
- Para a = -191, teremos -191 65345 A

Exercício4: Elabore tabela de Conversão de Temperaturas entre as Escalas Celsius e Fahreint.

Algoritmo:
inteiro fahr

```
flutuante celsius
para fahr de 0 até 300 passo 20 faça
        imprima fahr
        celsius = 5.0/9.0*( fahr-32)
        imprima celsius
```

- Programa:

```
main()
{
int fahr;
float celsius;
for (fahr = 0; fahr <= 300; fahr = fahr + 20) {
        printf("%4d", fahr);
        celsius = (5.0/9.0)*(fahr-32);
        printf("\t%6.1f\n",celsius);
  }
}
```

Note que quando dois ou mais comandos devam ser executados, devemos **obrigatoriamente** utilizar chaves para delimitar a seqüência de instruções a ser observada. Analisando o programa a seguir, percebemos que:

```
if (x == 0) {
        printf("A");
        printf("B");
  }
else
        printf("C");
```

Se x = 0 então serão impressos AB senão impresso C

```
if (x == 0)
        printf("A");
printf("B");
```

Se x = 0 então serão impressos AB senão impresso B

- **Exercício:** Programa Fatorial.
- Algoritmo:
    - *inteiro p,r,s*
    - *s = 1*
    - *leia r*
    - *para p de r até 1 faça*
    - *s = s * r*
    - *imprima s*

Linguagem C:

```
main()
{
int i,j;
float f=1;
printf("Digite um numero: ");
scanf("%d",&j);
for(i=1;i<=j;i++)
        f=f*i;
printf("O fatorial de %d e' %7.0f.",j,f);
}
```

Alternativamente poderíamos resolver este mesmo problema utilizando funções.

```
main()
{
int j;
```

```
printf("Digite um numero: ");
scanf("%d",&j);
printf("O fatorial de %d e' %d",j,fat(j));
}
int fat(n)
int n;
{
int i,f=1,z=3;
for(i=1;i<=n;i++)
        f = f*i;
return f;
}
```

Recursividade

A Recursividade é o ato da função chamar a si mesma com a finalidade de resolver determinado problema. O problema do fatorial também pode ser utilizado para demonstrar a recursividade como segue:

```
main()
{
int j;
printf("Digite um numero: ");
scanf("%d",&j);
printf("O fatorial de %d e' %d",j,fat(j));
}

int fat(n)
int n;
{
int i,f=1,z=3;
if (n == 1)
        return 1;
else
        return fat(n-1) * n;
}
```

Laboratório (Aula 06L)

1- Dado um número inteiro e uma razão, calcule a somatória dos primeiros 5 termos da Progressão Aritmética, determinada por estes números.

2- Processe os 3 exemplos do fatorial vistos na aula de teoria.

3- Exiba os primeiros "n" termos da PA de razão 2 e termo inicial 4.

4- Exiba os primeiros "n" termos da PA de razão 3 e termo inicial 3.