Variáveis e Operadores

Todas as variáveis devem ser delaradas desta forma: "tipo lista_de_variáveis;"

Exemplos:

- int i,j,l;
- short int si;
- unsigned int ui;
- double balanco, consolidacao;

De maneira semelhante ao que ocorre no Pascal e demais linguagens estruturadas, em C as variáveis tem seus valores localizados nas rotinas onde foram declaradas (escopo).

Basicamente, as variáveis podem ser declaradas fora das funções (globais) que valem para todas as funções do programa. Podem ser declaradas dentro de uma função (locais) sendo desconhecida no restante do programa. Além disso podem ser usadas para passagem de valores entre funções (parâmetros).

- Exemplo 1: Variáveis locais e globais; parâmetros.

```
int soma;                /* global */
main()
{
int cont;               /* local */
soma = 0;               /* soma(global) = 0 */
for (cont = 0; cont < 10; cont ++) {
        total(cont);
        display();
  }
}

total(x)
int x;                   /* x é parâmetro e vale cont */
{
soma = x + soma;
}

display()
{
int cont;            /* cont é local e difere de cont da função main() */
for (cont = 0; cont < 10; cont++) printf("-");
cont++;              /* equivale a cont = cont + 1    */
printf("A Soma atual é %d\n",soma);
}
```

Resultando em 0 1 3 6 10 15 21 28 36 45

Operadores

C é uma das linguagens com maior número de operadores, devido possuir todos os operadores comuns de uma linguagem de alto nível, porém também possuindo os operadores mais usuais a linguagens de baixo nível. Para fins didáticos, dividiremos os operadores em aritméticos, lógicos e de bits. No momento abordaremos apenas as duas primeiras classes.

Operadores Aritméticos

Operador	Ação
+	Adição
*	Multiplicação
/	Divisão
%	Resto de Divisão Inteira
-	Subtração o menos unário
--	Decremento
++	Incremento

Operadores Relacionais e Lógicos

Operador	Ação
>	Maior que
>=	Maior ou igual que
<	Menor que
<=	Menor ou igual que
==	Igual a
!=	Diferente de
&&	Condição "E"
||	Condição "OU"
!	Não

Observação: Em C o resultado da comparação será ZERO se resultar em FALSO e DIFERENTE DE ZERO no caso de obtermos VERDADEIRO num teste qualquer. Programadores experientes utilizam-se desta conclusão em alguns programas, onde "inexplicavelmente" algo é testado contra ZERO.

Comparações e Testes

Observemos antes de mais nada que ++x é diferente de x++!

Se

```
x = 10;

y = ++x;
```

então

x = 11 (pois x foi incrementado) e y = 11

porém Se

```
x = 10;

y = x++;
```

então

x = 11 e y = 10 (pois x foi atribuído a y ANTES de ser incrementado)

Se

x = 1;

y = 2;

printf("%d == %d e' %d\n",x,y,x==y);

então resultaria em 1 == 2 0 (pois a expressão é falsa)

if (10 > 4 && !(10 < 9) || 3 <= 4)

resultaria em Verdadeiro pois dez é maior que quatro E dez não é menor que nove OU três é menor ou igual a quatro

Operador Sizeof

Este operador retorna o tamanho da variável ou tipo que está em seu operando.

Por exemplo "sizeof(char)" resultaria em 1.

Conversões de Tipos

Quando forem misturadas variáveis de diferentes tipos, o compilador C converterá os operandos para o tipo de operando maior, de acordo com as regras descritas a seguir:

- 1- Todo char e short int é convertido para int. Todo float é convertido para double.
- 2- Para os demais pares de operandos valem as seguintes regras em seqüência:

- 2.1- Se um operando for long double, o outro também o será.
- 2.2- Se um operando for double, o outro também o será.
- 2.3- Se um operando for long, o outro também o será.
- 2.4- Se um operando for unsigned, o outro também o será.

Nota: Devemos observar que o compilador C é bastante flexível e pouco vigilante, comportando-se de maneira muito diferente de um compilador Clipper ou Pascal, sempre vigilantes com relação aos tipos das variáveis. De fato aqueles compiladores podem gerar executáveis misturando tipos, porém a ocorrência de erros de execução é quase inevitável. Ao contrário destes compiladores, os compiladores C "ajeitam" as coisas para o programa funcionar da "melhor maneira possível", o que não significa em hipótese alguma que os resultados serão os esperados por programadores "relapsos". Assim esta boa característica dos compiladores C, pode transformar-se numa autêntica "bomba relógio" para programas não muito bem elaborados.

Laboratório (Aula 07L)

1- Observe o programa a seguir.

```
main()
{
int i=1,j=2,k=3,l=4;
i++;
k=++i;
l=j++;
++j;
printf("%d %d %d %d",i,j,k,l);
```

}

Constate os resultados do programa acima.

2- Elabore programa contendo 2 variáveis x e y, que atribuam valores a i e j desta forma: i = ++x e j = y++. Constate os resultados.

3- Dados 3 números, os imprima em ordem crescente usando apenas 1 comando printf.

4- Dados 2 números inteiros, imprima 1 se ambos forem positivos ou negativos, 2 se tiverem sinais opostos ou 3 se um deles for zero.

5- Observe o programa a seguir.

```
main()
{
int i=1,j=2,k,l;
i++;
k=++i+k;;
l=j++ +l;
++j;
printf("%d %d %d %d",i,j,k,l);
}
```

Processe o programa 3 vezes. Justifique os resultados.

- 6- Utilize função DOSTIME, disponível no Classic C.
- 7- Processe o programa a seguir e constate os resultados

```
main()
{
int i=1,j=2,k,l;
i++;
k=++i +k;
l=j++ +l;
++j;
printf("%d %d %d %d",i,j,k,l);
}
```

Tomadas de Decisão - Parte II

Analisaremos mais detidamente o comando "if" e também os comandos "switch" e "?", destinados a promover desvios condicionais em nossos programas.

if

Sintaxe

- *if <condição>*
- *<bloco de comandos>;*
- *[else*
- *<bloco de comandos>;]*

- Exemplos: Comparações Simples, uso ou não do else.

```
if (t == 0)
        printf("T vale Zero");
if (t == 2 || t = 4) {
        printf("T vale Dois\n");
        printf("ou T vale Quatro");
        }
```

```
else {
        printf("T nao e'2 ou 4\n");
        if (t > 10)
                printf("E supera 10");
        else
                printf("Mas nao supera 10");
        }
```

Exemplo: Evitar-se divisões por Zero, usando recursos do comando if.

```
main()
{
int a,b;
printf("Digite 2 números: ");
scanf("%d %d",&a,&b);
if (b)
        printf("%f",a/b);
else
        printf("Nao posso dividir por zero\n");
}
```

Operador ?

A declaração Se-Então-Senão pode ser substituída por:

Exp1 ? Exp2 : Exp3

Se Exp1 for verdadeira, então Exp2 é avaliada tornando-se o valor de Exp1, senão Exp3 é que será avaliada e tornar-se-á o valor da expressão.

- Exemplo: Uso do ?.
- x = 10;
- y = (x > 20) ? 200 : 100;
- assim y valerá 100

Comando switch

Diversas vezes precisamos determinar se um valor encontra-se numa lista de valores. Apesar de podermos usar uma seqüência de ifs, este recurso além de não ser elegante, por vezes confunde o entendimento do programa. Vejamos uma opção melhor: o comando switch.

Sintaxe:

switch <variável> {

case <constante 1> :

<comandos>;

[break;]

case <constante 2> :

<comandos>;

[break;]

case <constante 3> :

<comandos>;

[break;]

[default :

<comandos>;]

}

Observe que "break" serve para terminar a seqüência de comandos em execução, por serem opcionais, se forem suprimidos permitem que o "case" a seguir seja executado, sem haver qualquer quebra na seqüência do processamento.

- Exemplo: Frases Montadas

```
main()
{
int t;
for (t = 0; t < 10; t ++)
        switch (t) {
                case 1:
                        printf("Agora");
                        break;
                case 2:
                        printf("e'");
                case 3:
                case 4:
                        printf("hora ");
                        printf("de todos os homens bons\n");
                        break;
                case 5:
                case 6:
                        printf("trabalharem");
                        break;
                case 7:
                case 8:
                case 9:
                        printf("-");
                }
}
```

Resultará em:

- Agora é hora de todos os homens bons
- hora de todos os homens bons
- hora de todos os homens bons
- trabalharem trabalharem ---

Laboratório (Aula 08L)

1- Elabore programa que solicite 2 números e uma operação matemática elementar (+-*/) e a execute.

2- Elabore programa que imprima a seqüência de frases abaixo, usando switch.

Verde Verde Vermelho

Amarelo

Amarelo

Amarelo

Azul Branco e Preto

Branco e Preto

3- Elabore programa que 'z' termine com o valor 4 se 'a' for maior que 10 ou 5 se 'a' for menor ou igual a 10. Os comandos if e switch não poderão ser utilizados.

4- Elabore um menu de opções com 4 situações diversas, utilizando switch.

5- Elabore programa que permita a 5 pessoas escolherem sua cor favorita entre Verde, Vermelho, Amarelo, Azul, Laranja ou Roxo e exiba os resultados.

6- Elabore programa que permita a escolha entre 1, 2 e 3 ou indique erro de escolha.

Loops

Estruturas de repetição normalmente usadas nos programas, terão em C três sintaxes distintas (for, while e do-while), cada uma delas indicada para determinado tipo de necessidade.

for

Sintaxe:

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

Além da sintaxe vista anteriormente, "for" permite a escrita de expressões mais elaboradas, sem qualquer paralelo nas linguagens BASIC, Pascal e COBOL, como pode ser vista a seguir:

```
for (x=0,y=0;x+y<100;++x,y=y+x)
printf("%d",x+y);
```

Esta instrução inicializaria x e y com zero, incrementando x de 1 em 1 e y receberia seu valor acrescido do de x. O resultado a cada iteração seria impresso desta forma: 0 (x=0 e y=0) 2 (x=1 e y=1) 5 (x=2 e y=3) 9 14 e assim sucessivamente.

- Exemplo 1: Contagem simples com condição no teste "for".

```
main()
{
int i,j,resposta;
char feito = ' ';
for (i=1;i<100 && feito != 'N';i++) {
        for (j=1;j<10;j++) {
                printf("Quanto e' %d + %d? ",i,j);
                scanf("%d",&resposta);
                if (resposta != i+j)
                        printf("Errou!\n);
                else
                        printf("Acertou!\n");
                }
        printf("Mais? (S/N) ");
        scanf("%c",&feito);
                }
}
```

- Exemplo 2: Contagem com funções nos argumentos do "for".

```
main()
{
int t;
for (prompt();t=readnum();prompt())
```

```
        sqrnum(t);
}

prompt()
{
printf("Digite um número inteiro!");
}
readnum()
{
int t;
scanf("%d",&t);
return t;
}

sqrnum(num)
int num;
{
printf("%d\n",num*num);
}
```

Loops Infinitos

```
for(;;)
```

printf("Este loop rodará eternamente!\n");

A ausência de condições de inicialização, continuidade e terminação, causarão um processo contínuo e teoricamente infinito (veremos posteriormente a intrução **break**, que tem a capacidade de encerrar um processo assemelhado ao exemplificado).

Loop Vazio

```
for(i=0;i<10;i++);
```

A presença do ponto e vírgula finalizando o comando, força a execução do loop sem que seja executado qualquer outro comando.

Loop Finito

Ao contrário de outras linguagens que não permitem o término do loop a não ser quando a condição de finalização for satisfeita, a linguagem C permite que um loop seja interrompido antes de seu término normal (desestruturação) sem que exista qualquer tipo de inconveniente. O comando "break" causa a interrupção conforme pode ser visto a seguir:

```
for(;;) {
        scanf("%d",&c);
        if (c == 'A')
                break; /* interrompe o que deveria ser um anel eterno */
   }
printf("Fim do Loop!");
```

Conclusão

As características do comando "for", como pudemos notar são bem superiores as dos comandos "for" das linguagens BASIC, Clipper, Pascal e COBOL. Enquanto em algumas daquelas linguagens é recomendável que seu uso seja evitado, em C seu uso é normalmente o mais recomendado, pois substitui geralmente com vantagem seu análogo "while", como veremos a seguir.

Loop while

- *Sintaxe:*
- *while <condição> <comando>;*
- O loop se repete, enquanto a condição for verdadeira.
- Exemplo: Contagem

```
main()
{
int i;
while (i < 10) {
        printf("%d",i);
        i--;
  }
}
```

Loop do/while

Ao contrário das estruturas "for" e "while" que testam a condição no começo do loop, "do / while" sempre a testa no final, garantido a execução ao menos uma vez da estrutura. Este comando é similar (porém não idêntico) ao "repeat" da linguagem Pascal.

Sintaxe:

- *do {*
- *<comandos>;*
- *} while <condição>;*

- Exemplo: Término determinado pelo usuário.

```
main()
{
int num;
do {
        scanf("%d",&num);
} while (num < 100);
}
```

Laboratório (Aula 09L)

- 1- Retome o programa da tabuada, porém agora imprimindo todas de 1 ao 10.
- 2- Elabore programa que decida se um número informado é primo.
- 3- Imprima todos os números primos dentro de um intervalo fornecido pelo usuário.
- 4- Dada uma seqüência de números reais, calcule sua média e desvio padrão.
- 5- Dada uma seqüência de números reais, imprima a mediana da série.
- 6- Retome o exercício 2 e o resolva usando do/while e while.

Comandos Desestruturadores

Vimos anteriormente o comando "break" finalizando opções do comando "switch" e também terminando um loop "for". Apesar deste último tipo de uso não ser recomendado por alguns defensores da programação estruturada, quando usado de forma adequada (isto é aumentando a velocidade de execução sem prejudicar a compreensão do programa), não há qualquer inconveniente para esta utilização. Aproveitamos para lembrar que a linguagem C é conhecida como "a linguagem dos profissionais" não tendo a pretensão de ser compreendida por leigos como por exemplo as chamadas linguagens de comitês (BASIC e COBOL). O comando "break" é análogo ao comando "exit" da linguagem Clipper.

break

Exemplo: Loops encadeados terminados com uso do "break"

```
main()
{
int t,cont;
for (t=0;t<100;++t) {
        cont = 1;
        for (;;) {
```

```
                printf("%d",cont);
                cont++;
                if (cont == 0) break;
                }
}
```

Observe que break, quebra apenas a estrutura "for" mais interna, a externa será processada até o final normalmente.

continue

O comando "**continue**" funciona de maneira análoga ao "**break**", contudo ao invés de forçar o encerramento do loop, força nova iteração (semelhante a instrução "loop" da linguagem Clipper) saltando o código entre seu uso e a marca de término do loop.

- Exemplo: Imprimir somente os números pares entre 1 e 100.

```
main()
{
int x;
for (x=0;x<100;x++) {
        if (x%2) continue;
                printf("%d",x);
  }
}
```

Desta forma toda vez que for gerado um número ímpar "if" será executado saltando o comando "printf". "Continue" assemelha-se ao "loop" da linguagem Clipper.

goto

De maneira geral o comando "goto" é considerado o grande vilão das linguagens de programação e geralmente seu uso é desaconselhado quando não proibido por equipes de desenvolvimento e por autores de manuais de linguagens.

Devemos dizer a priori, que uma instrução apenas não teria o poder de causar tantos estragos, porém certos programadores ...

"Goto" apesar de não ser imprescindível para escrita de programas em linguagem C e de ter seu uso restrito a condições particulares, pode ajudar não só a tornar a execução de um programa mais rápida como também mais clara (!!), conforme veremos posteriormente.

"Goto" necessita de um rótulo para poder operar. O rótulo nada mais é que um identificador válido em C seguido de dois pontos. Além disso o rótulo deverá pertencer a função onde se encontra o "goto". Um uso indevido do "goto" está exemplificado a seguir:

```
x = 1;
loop1:
x++;
if (x<100)
        goto loop1;
```

Preferencialmente deveríamos ter usado uma estrutura "while" ou "for" para elaboração deste trecho de programa.

Exemplo: Uso adequado do "goto"

O esboço de programa a seguir, estruturado, é bastante confuso, conforme é fácil observar:

```
feito = 0;
        for (...) {
                for (...) {
```

```
                        while (...) {
                                if (...) {
                                                        feito = 1;
                                                        break;
             }
             .
             .
             .
                   }
                if (feito) break;
                }
        if (feito) break;
        }
if (feito)
        break;
```

Vejamos a solução desestruturada (usando Goto)

```
for (...) {
        for (...) {
                while (...) {
                        if (...)
                                        goto stop;
                           }
           .
           .
           .
         }
}

stop:
printf("Saída!");
```

Normalmente soluções não estruturadas são mais rápidas que aquelas estruturadas, e neste caso a solução também é muito mais clara, justamente pela utilização adequada do "goto".

Velocidade x Estilo x Clareza

As três versões do programa de verificação de um número primo, servem para nos mostrar as diferenças de desempenho que podem ocorrer, independente da linguagem utilizada, somente em função do algoritmo que utilizarmos. Processe estes exemplos e constate as diferenças:

Versão 1: Algoritmo Estruturado, sem conceituação matemática:

```
main()
{
  int i,ini,fim,n,j,nao;
  char ac[80];
  cls();
  printf("Digite extremo inferior: "); scanf("%d",&ini);
  printf("\nDigite extremo superior: ");scanf("%d",&fim);
  dostime(ac, 2);
  puts(ac);
  for(i=ini;i<=fim;i++) {
    nao = 1;
    for(j=2;j<i;j++)
      if(i % j == 0)
        nao = 0;
    if (nao || i == 2)
       printf("%d ",i);
  }
  printf("\n");
  dostime(ac, 2);
  puts(ac);
}
```

Versão 2: Algoritmo Estruturado com conceituação matemática.

```
main()
{
  char ac[80];
  int j,i,ini,fim,n,nao;
  double r;
  cls();
  printf("Digite extremo inferior: "); scanf("%d",&ini);
  printf("\nDigite extremo superior: ");scanf("%d",&fim);
  dostime(ac, 2);
  puts(ac);
  for(i=ini;i<=fim;i++) {
    nao = 1;
    j = 2;
    r = i;
    r = sqrt(r);
    while (j<=r) {
      if(i % j == 0)
        nao = 0;
      j++;
    }
    if (nao || i == 2)
       printf("%d ",i);
  }
  printf("\n");
  dostime(ac, 2);
  puts(ac);
}
```

Versão 3: Algoritmo com conceituação matemática, com liberdades na estruturação

```
main()
{
  char ac[80];
  int j,i,ini,fim,n,nao;
  double r;
  cls();
  printf("Digite extremo inferior: "); scanf("%d",&ini);
  printf("\nDigite extremo superior: ");scanf("%d",&fim);
  dostime(ac, 2);
  puts(ac);
  for(i=ini;i<=fim;i++) {
    nao = 1;
    if (i % 2 == 0)
       nao = 0;
    else {
       j = 3;
       r = i;
       r = sqrt(r);
       while (j<=r) {
        if (i % j == 0) {
           nao = 0;
           break;
           }
        j =j + 2;
        }
    }
    if (nao || i == 2)
       printf("%d ",i);
  }
  printf("\n");
  dostime(ac, 2);
  puts(ac);
}
```

Laboratório (Aulas 10L, 11L e 12L)

Elabore um jogo onde o humano tenta descobrir o número do computador e vice-versa. Os números devem estar entre 1 e 1023, sendo que o jogo só termina quando um (ou ambos) jogadores acertarem o número de seu oponente. O Empate ocorre quando os dois jogadores acertarem o número de seu oponentes na mesma jogada.

Solução:

```
/* Autores: Deusdeth, Luis Fernando, Ana */
int verif=0,numm,numh,ia=0,ib=1023,contm=0,conth=0,resph,palpm;
char nome[32];
main ()
{
cls ();
apresent ();
nome_hum ();
num_maq ();
while (verif == 0){
        joga_hum ();
        joga_maq ();
        rotina_verif ();
        rotina_verif1 ();
        rotina_verif2 ();
        }
rotina_venceu ();
}

apresent ()
{
puts ("|*********************************************************|");
puts ("|*             J O G O     A L O - V E J A               *|");
puts ("|************* ( A P R E S E N T A C A O ) ***************|");
puts ("|* - JOGO ENTRE A MICRO E VOCE. O MICRO IRA' ESCOLHER UM *|");
puts ("|*   NUMERO E VOCE OUTRO NUM INTERVALO ENTRE 1 E 1023.   *|");
puts ("|* - CADA UM IRA' TENTAR DESCOBRIR O NUMERO DO OPONENTE, *|");
puts ("|*   ATE' QUE UM DOS JOGADORES ADIVINHE O NUMERO DO OUTRO*|");
puts ("|* - O MICRO IRA' INFORMAR SE O SEU PALPITE FOI CORRETO, *|");
puts ("|*   BAIXO OU ALTO.                                      *|");
puts ("|* - VOCE DEVERA' FAZER O MESMO, INFORMANDO:             *|");
puts ("|*   (1) PARA UM CHUTE BAIXO;                            *|");
puts ("|*   (2) PARA UM CHUTE ALTO;                             *|");
puts ("|*   (3) PARA CERTO.                                     *|");
puts ("|*********************************************************|");
}

nome_hum ()
{
printf ("INFORME O SEU NOME: ");
gets (nome);
}

num_maq ()
{
numm=rand()/32;
}

joga_hum ()
{
printf ("%s, tente acertar o numero que eu escolhi : ",nome);
scanf  ("%d",&numh);
puts   ("o resultado sera divulgado apos a jogada do micro");
conth=conth+1;
puts ("************************************************************");
puts   (" ");
}

joga_maq ()
{
palpm=(ia+ib+1)/2;
printf ("%s, acho que voce pensou no numero %d",nome,palpm);
puts   (" ");
printf ("digite (1) baixo, (2) alto ou (3) certo : ");
scanf  ("%d",&resph);
contm=contm+1;
puts ("************************************************************");
puts   (" ");
}

rotina_verif ()
{
```

```
if (numh == numm)
        verif = verif + 1;
else
        if (resph == 3)
                verif = verif + 1;
}

rotina_verif1 ()
{
if (numh > numm){
        puts (" ");
        printf ("seu chute foi alto");
        puts (" ");
        }
else
        if (numh < numm){
                puts (" ");
                printf ("seu chute foi baixo");
                puts (" ");
   }
}

rotina_verif2 ()
{
if (resph == 1)
        ia = palpm;
else
        if (resph == 2)
                ib = palpm;
}

rotina_venceu ()
{
if (numh == numm)
        if (resph == 3)
                printf("\nOcorreu Empate! \n\n* fim do jogo.\n");
        else {
                puts  (" ");
                printf("* %s parabens, voce acertou em %d tentativas.",nome,conth);
                puts  (" ");
                puts  ("* fim do jogo.");
                puts  (" ");
         }
else
        if (resph == 3){
                puts  (" ");
                printf("* %s o micro acertou em %d tentativas.",nome,contm);
                puts  (" ");
                puts  ("* fim do jogo.");
                puts  (" ");
          }
}
```

Exercício:

Elabore programa baseado no Jogo da Velha, onde o jogador humano enfrenta a máquina. Observe que o computador nunca deverá ser derrotado, podendo eventualmente derrotar o jogador humano.

Matrizes

Correspondem a elementos do mesmo tipo, agrupados sob o mesmo nome e diferenciados entre si através de índices. Na linguagem C, todos os vetores ou matrizes consistem em posições contíguas, sendo que o endereço mais baixo corresponde ao primeiro elemento e o endereço mais alto ao último elemento.

Declaração

Deve ser feita nestes formatos:

```
char nome[20];
float preco[30];
int m[5][3]; /* bidimensional */
char c[3] = {'f','i','m'}; /* declarada e inicializada */
char d[3] = "fim"; /* idem */
int a[5] = {1,10,3,5,30} /* declarada e inicializada, numérica */
```

- Exemplo 1: Imprima 5 números na ordem oposta a que forem informados.
- Solução sem matriz

```
main()
{
int a,b,c,d,e;
scanf("%d %d %d %d %d",&a,&b,&c,&d,&e);
printf("%d %d %d %d %d",e,d,c,b,a);
}
```

- Solução com matriz

```
main()
{
int i,a[5];
for (i=1;i<=5;i++) {
        printf("Elemento %d: ",i);
        scanf("%d",a[i]);
}
puts(" ");
for (i=5;i>0;i--)
        printf("Elemento %d: ",a[i]);
}
```

- Exemplo 2: Tabuada em matrizes.

```
main()
{
int a[10], i, t = 3;
for (i=0;i<11;i++)
        a[i] = i * t;
for (i=0;i<11;i++)
        printf("%d * %d = %d",i,t,a[i]);
}
```

Exercício: Retome o Exercício da Aula de Laboratório 8, exercício 3 e o resolva com o uso de matrizes.

Iaboratório (Aula 11L)

Jogo da Velha: Continuação.

Ordenação

Exemplo: Dada uma série de números inteiros, ordene-a de forma crescente.

- Algoritmo:

```
leia n
para i de 1 até n
        leia a[i]
para i de 1 até n-1 faça
        para j de i+1 até n faça
                se a[i] > a[j] então
                        m = a[i]
                        a[i] = a[j]
                        a[j] = m
para i de 1 até n
        imprima a[i]
```

Em C teríamos:

```
main()
{
int x,y,z,w,a[100];
do
        {
        traca_linha()
        puts("Programa para ordenar 'n' numeros digitados.");
        traca_linha();
        printf("Quantos Numeros (0=fim) -> ");
        x = leintf();
        if (x==0) break;
                traca_linha()
        for (y = 0; y < x; y++) {
                printf("a[%2d]= ",y);
                a[y] = leintf();
                }
        traca_lina();
        for (y=0;y<(x-2),y++)
                for (z=y+1;z<(x-1);z++)
                        if (a[y] > a[z]) {
                                w = a[y];
                                a[y] = a[z];
                                a[z] = w;
                        }
        for (y=0;y<(x-1);y++)
                printf("%d\n",a[y]);
    } while(1);
}

traca_linha()
{
int x;
for (x=1;x != 80; x++)
        putchar('=');
putchar('\n');
}

leintf()
{
char s[20];
gets(s);
return(atoi(s));
}
```

Iaboratório (Aula 12L)

Jogo da Velha: Continuação.

Ponteiros - Apresentação

Introdução

Ponteiros são endereços, isto é, são variáveis que contém um endereço de memória. Se uma variável contém o endereço de outra, então a primeira (o ponteiro) aponta para a segunda.

"x" o "ponteiro" aponta para o "inteiro" a

Operadores

&- (E comercial) que fornece o endereço de determinada variável. Não confundir com o operador lógico de operações de baixo nível, de mesmo símbolo. Atribui o endereço de uma variável para um ponteiro.

*- (Asterístico) que acessa o conteúdo de uma variável, cujo endereço é o valor do ponteiro. Não confundir com o operador aritmético de multiplicação de mesmo símbolo. Devolve o valor endereçado pelo ponteiro.

Exemplos:

```
main()
{
int *pont, cont, valor;
cont = 100;
pont = &cont;
val = *pont;
printf("%d",val);   /* 100 */
}
```

```
main()
{
char a,b,*x;
b = 'c';
p = &a;
*p = b;
printf("%c",a);   /* c */
}
```

```
main()
{
int x,y,*px,*py;
x = 100;
px = &x;   /* px tem o endereco de x */
py = px;   /* py tem o endereco de x */
y = *py;   /* y vale 100, pois recebe o conteúdo de x */
           /* , através do  ponteiro py */
printf("%d %d",x,y);
}
```

Laboratório (Aula 13L)

- 1- Executar Programa de Ordenação de uma Série de Números.
    - 2- Dado um nome invertá-o.
    - 3- Calcule a média de "n" números, posteriormente imprimindo-os.
    - 4- Dada uma série com "n" números, imprima a média dos n-1 maiores termos, usando matrizes obrigatoriamente.
- 5- Simule a execução do programa abaixo:

```
#include "stdio.h"
main()
{
int i,k,*pi,*pk;
char a;
i = 2; k = 0;
puts("Qual sera o valor de k? ");
pk = &k;
pi = &i;
*pk = i;
printf("para *pk = i, temos k= %d\n",k);
k = *pi;
printf("para k = *pi, temos k= %d\n",k);
scanf("%c",&a);
}
```

- 6- Simule a execução do programa abaixo:

```
main()
{
int x,y,*px,*py;
printf("Digite um valor: ");
scanf("%d",&x);
px = &x;
y = *px;
```

```
printf("digitou= %d e y= %d\n",x,y);
*px = 8;
printf("valor mudou para %d\n",x);
}
```

- 7- Simule a execução do programa a seguir:

```
main()
{
char a,b,*p;
b = 'c';
p = &a;
*p = b;
printf("%c",a);
}
```

Ponteiros - Conceitos de Endereços

Aritmética de Ponteiros

São válidas as operações de soma e subtração, sendo que seu resultado depende do tipo de variável apontada pelo ponteiro.

Supondo que

```
int *p, x;
char *q, a;
```

se

```
q = &a;
p = &x;
```

e ainda que

a-endereço 100

x-endereços 101/102

então q++ --> q "apontará" para o endereço 101

p++ --> p "apontará" para o endereço 103

Este conceito é particularmente importante no que se refere a matrizes pois como se sabe, matriz nada mais é que um conjunto de variáveis do mesmo tipo, dispostas seqüencialmente em memória.

Por conterem endereços, ponteiros permitem apenas as operações de soma e subtração. Supondo que:

```
int i,*pi;
char c,*pc;
float f,*pf;
```

Supondo ainda que pi, pc e pf apontem para i, c e f que estariam com os seguintes endereços: 2340, 2345 e 2350.

Se

pc = pc + 1, então pc valerá 2346, pois variáveis caracteres possuem apenas 1 byte.

pi = pi + 1, então pi valerá 2342 (!), pois variáveis inteiras ocupam 2 bytes.

pf = pf + 5, então pf valerá 2370 (!), pois variáveis pt. flutuante ocupam quatro bytes.

Exemplo

```
int i,*pi;
char c,*pc;
float f,*pf;
```

Supondo ainda que pi, pc e pf apontem para i, c e f que estariam com os seguintes endereços: 2340, 2345 e 2350.

Se

pc = pc + 1, então pc valerá 2346, pois variáveis caracteres possuem apenas 1 byte.

pi = pi + 1, então pi valerá 2342 (!), pois variáveis inteiras ocupam 2 bytes.

pf = pf + 5, então pf valerá 2370 (!), pois variáveis pt. flutuante ocupam quatro bytes.

- Exemplo: Atribuições indiretas

```
main()
{
int x,y,*px,*py;
x = 100;
px = &x;   /* px tem o endereco de x */
py = px;   /* py tem o endereco de x */
y = *py;   /* y vale 100, pois recebe o conteúdo de x, através do         ponteiro py */
printf("%d %d",x,y);
}
```

Comparação de Ponteiros

Por se tratar de endereços os ponteiros podem ser comparados.

Exemplo:

```
if (p > q) puts("p aponta para endereço maior que q");
```

Ponteiros - Prática - Conceitos Básicos (Aula 14 L)

Processar exemplos vistos em teoria.

Ponteiros Conceitos Avançados

Strings

Consideremos o trecho de código abaixo:

```
char linha[80],*p;*p1;
p = &linha[0];
```

```
p1 = linha;                     /* p1 e p possuem o mesmo endereço, i.é, */
if (p==p1)
puts("iguais!");        /* apontam para o 1º elemento da matriz! */
```

Exemplo: Inverter os elementos de uma string, usando ponteiros.

Solução: Pela maneira convencional teríamos:

```
main()
{
char str[80];
int i;
printf("Digite uma palavra: "); gets(str);
for (i=strlen(str)-1;i>=0;i--)
printf("%c",str[i]);
}
```

- com ponteiros teríamos:

```
main()
{
char str[80],*p;
int i;
printf("Digite uma palavra: "); gets(str);
p = str;
for(i = strlen(str) - 1;i>=0;i--)
printf("%c",*(p+i));
}
```

- Exemplo: Uso de Ponteiros e String em mesma variável

```
char *p = "Imprimindo Algo\n";
main()
{
int i;
printf("%s\n - por ponteiro...\n\n",p);
for(i=0;p[i];i++)
printf("%c",p[i]);
puts("\n - por caracteres em ponteiro ");
}
```

Velocidade

O acesso aos elementos da matriz é mais rápido quando feito através de endereços do que seria caso fosse executado pela maneira convencional, porém expressões que envolvam cálculos para se obter um elemento específico da matriz devem ser feitas preferencialmente pelo método convencional, pois expressões complexas envolvendo ponteiros são processadas em velocidade semelhante àquelas feitas pelo modo convencional, que apresenta a vantagem de ser mais facilmente compreendido.

Matrizes e Ponteiros

Em C você pode indexar um ponteiro como se este fosse uma matriz!

Exemplo:

```
main()
{
int i,*p,a[5];
for (i=0;i<=4;i++) {
        printf("Digite o %d elemento",);
        scanf("%d",&a[i]);
         }
p = a;
for (i=0;i<=4;i++)
```

```
        printf("%d %d %d\n",a[i],p[i],*(p+i));
}
```

Ponteiros - Prática - Conceitos Avançados (Aula 15 L)

Processar Exemplos vistos em teoria

Ponteiros - Pilhas

Uma pilha é uma lista de variáveis do mesmo tipo (semelhantes a uma matriz, ou mesmo uma matriz), onde utilizamos o conceito de que "o primeiro que entra é o último a sair". Imaginemos um bloco de folhas. Normalmente utilizaremos primeiro a última folha do bloco ("a de cima"), enquanto que a primeira folha colocada ("a de baixo") será a última a ser utilizada. Nos exemplos a seguir serão utilizadas duas funções: push() e pop(). Usaremos push () para inserir elementos na pilha e pop() para sacá-los.

Exemplo: Faça com que seja criada uma pilha com no máximo 50 elementos, onde números positivos empilhem elementos na pilha, 0 tire o último elemento da pilha e -1 cause o encerramento do programa:

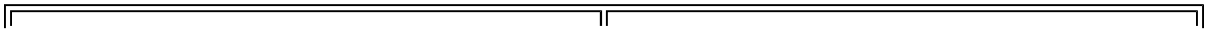
```
int pilha[50],*p1,*to;
main()
{
int valor;
p1 = pilha;
to = p1;
printf("Numero --> Pilha, 0 recupera e -1 finaliza \n");
do {
        scanf("%d",&valor);
        if (valor != 0)
                push(valor);
        else {
                valor = pop();
                printf("%d\n",valor);
      }
        } while (valor != -1);
}

push(i)
int i;
{
p1++;
if (p1 == (to + 50)) {
        puts("Estouro da Pilha (superior) ");
        exit(1);
         }
*p1 = i;
}

pop()
{
if ((p1) == to) {
        puts("Estouro da Pilha (inferior) ");
        exit(1);
 }
p1--;
return *(p1+1);
}
```

Imaginemos agora uma pilha onde não soubéssemos antecipadamente o total de elementos de uma pilha, neste caso precisaríamos de instruções que permitissem a manipulação da área de memória livre entre o programa e área de trabalho utilizada pelo programa conforme mostra o esquema abaixo:

Memória do Sistema



Usada para Variáveis Locais e endereços de Rotinas	Do "fim" para o inicio
Memória Livre para Alocação	
Variáveis Globais	Do "inicio" para o fim
Programa	

As funções malloc() e free() permitem que utilizemos a área de memória livre para criar nossas variáveis, por exemplo matrizes com limites a serem definidos em tempo de execução.

Exemplo:

```
char x,y,*px;
main()
{
px = malloc(1000);
if (!px) {
        puts("Memória Insuficiente!");
        exit(1);
         }
x = 'a';
push(x);
x = 'b';
push(x);
y = pop();
printf("%c\n",y);
y = pop();
printf("%c",y);
}

push(i)
char i;
{
px++;
*px=i;
}

pop()
{
px--;
return *(px+1);
}
```

Exemplo: Aloque uma certa quantidade de memória e posteriormente escreva letras na área alocada, recupere estas letras e depois libere a memória previamente alocada.

```
main()
{
char c,*p,*q;
int i=0,j,r;
cls();
q = malloc(1000);
p = q;
puts("Digite Letras ... para parar digite <enter>");
for(;;) {
        printf("Digite a %d letra: ",i+1);
        scanf("%c",&c);
        if (c == ' ')
                        break;
        *p = c;
        c = ' ';
        p++;
        i++
        }
p = q;
printf("\n\nLetras Digitadas:\n");
for(j=1;j<=i;j++) {
        printf("%d Letra = %c\n",j,*p);
        p++;
        }
r=free(q);
```

```
if (r==0)
        puts("\nLiberada Memória Utilizada!");
else
        puts("\nFalha na Liberação da Memória Alocada!");
}
```

Exemplo: Retome o exercício anterior, porém ao invés de armazernar letras, armazene números.

```
main()
{
char c,*q,*p;
int i = 0, j, r, m, n;
cls();
q = malloc(1000);
p = q;
puts("Digite Numeros ...  para parar digite <enter> ");
for(;;) {
        printf("Digite o %do. Numero: ", i+1);
        scanf("%d",&m);
        if (m==0)
                break;
        *p = m;
        m = 0;
        p=p+2;
        i++;
}
p = q;
printf("\n\nLetras Digitadas: \n");
for(j=1;j<=i;j++) {
        printf("%d Letra = %d\n",j,*p);
        p=p+2;
}
r=free(q);
if (r==0)
        puts("\nLiberada Memoria Usada!");
else
        puts("\nFalha na Liberação de Memória Reservada!");
}
```

Nota: A característica típica da linguagem C, que facilita a mistura de tipos, permite o endereçamento de variáveis *através de sua posição na memória,* ou seja, podemos usar um ponteiro de caracteres para apontar um número inteiro e recuperá-lo sem qualquer tipo de inconveniente.

Devemos notar entretanto que esta característica pode acarretar grandes problemas, e caso seja utilizada de forma inadequada, pode gerar problemas se acessarmos (ou principalmente se escrevermos) alguma área de memória importante.

Outro ponto importante, é percebermos que *não temos números, caracteres ou qualquer outra coisa aramazenadas na memória além de bits*, portanto tudo o que necessitamos saber é *onde* está uma informação procurada e *qual* é seu formato e simplesmente acessá-la (ou substituí-la) de acordo com nossa necessidade.

É desnecessário dizer, que estas são características dos assemblers próprios de cada equipamento, portanto a Linguagem C nos desobriga a aprender o Assembler de cada microprocessador, porém não elimina a necessidade de conhecermos sua arquitetura.

Matriz de Ponteiros e Indexação Múltipla

São válidas expressões como as que seguem abaixo:

```
int *x[10];
x[2] = &valor;
 .
 .
 .
printf("%d",*x[2]);
```

ou

```
main()
{
int x,*p,**q;
x = 10;
p = &x;
q = &p;
printf("%d",**q); /* 10 */
}
```

Ponteiros - Prática - Conceitos Avançados (Aula 16L)

Processar Exemplos de Pilha.

Ponteiros Conceitos Complementares

Ponteiros como Strings

Em C podemos usar o fato do ponteiro nulo (0 binário) ser o terminador de

strings combinado a possibilidade de representação de matriz como ponteiros,

conforme mostra o exemplo a seguir:

```
char *p = "Frase a demonstrar \n";
main()
{
int i;
printf("%s",p);
for (i=0;p[i];i++)
        printf("%c",p[i]);
}
```

Problemas a serem evitados com ponteiros

Os erros vistos a seguir podem passar não ser notados durante a fase de desenvolvimento do sistema e mesmo durante algum tempo em sua fase operacional e ser detectado apenas esporadicamente, tornando muito difícil sua localização pelo programador, pois só serão observados, no caso de serem apontados endereços vitais para o sistema.

Atribuição de posição de memória desconhecida:

```
main()
{
int x,*p;
x = 10;
*p = x;
printf("%d",*p);  /* valor desconhecido */
}
```

Observe que 10 é atribuído a uma posição de memória desconhecida, pois o endereço de "p" não é conhecido. Caso "p" estiver apontando para algum endereço vital, o sistema será paralisado, causando suspeita de mal funcionamento (hardware) ou presença de vírus (software). Na verdade, esta ocorrência é o chamado "bug" (pequeno erro no programa), pois somente algumas vezes causará erro. Observe serem grandes as possibilidades deste programa funcionar perfeitamente, pois o ponteiro provavelmente jogará o valor num local não usado. Porém quanto maior for o programa, mais provável será a possibilidade de encontrarmos um erro conforme descrito anteriormente.

Atribuição de valor para o ponteiro:

```
main()
{
int x,*p;
x = 10;
p = x;
printf("%d",*p);  /* valor desconhecido */
}
```

Observe que não será impresso 10, mas um valor desconhecido qualquer, pois 10 será atribuído ao ponteiro p, que supostamente contém um endereço e não um valor. Se tivéssemos "p = &x;" aí o programa funcionaria de forma correta.

Passagem de Variáveis ou Valores através Funções

Quando desejamos que uma função altere o valor de uma variável da função que a chamou, passamos para a função chamada o endereço desta variável (passagem de parâmetro por referência). Quando somente precisamos do valor da variável e não pretendemos alterá-lo na rotina (passagem de parâmetro por valor), passamos diretamente a variável, conforme visto a seguir:

- Exemplo 1: Passagem por Valor (a mantém seu valor)

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

soma(z)
int z;
{
int x=5;
x = 2 * x + z;
z = 0;
return(x);
}
```

- Exemplo 2: Passagem por Referência (a muda seu valor)

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

soma(z)
int *z;
{
int x=5;
x = 2 * x + *z;
*z = 0;
return(x);
}
```

Exemplo 3- Uso de ponteiros em funções.

```
main()
{
int a,b;
a = 100;
```

```
b = 20;
swapg(&a,&b);
printf("Maior = %d ",a);
printf("Menor = %d ",b);
}

swapg(c,d)
int *c,*d;
{
int t;
if (*c <= *d)
        return;   /*   nao troca   */
t  = *c;
*c = *d;
*d = t;
}
```

Laboratório (Aula 17L)

1- Troque o valor de 2 variáveis usando ponteiros, sem usar funções. Compare com o exemplo visto na aula teórica.

2- Crie um programa de contagem com loop usando ponteiros.

3- Crie um programa com 2 variáveis, sendo que a primeira será do tipo caracter e a segunda do tipo inteiro. Atribua valores da primeira para a segunda e vice-versa através de ponteiros.

Ponteiros x Matrizes e Entradas e Saídas

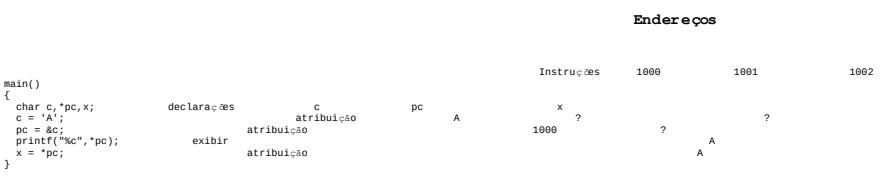
Exemplo: Retomando o exemplo da tabuada, resolvendo ao "estilo C de programar"

Estilo Pascal

```
main()
{
int a[10],i;
for(i=0;i<10;i++)
        a[i] = i*3;
for(i=0;i<10;i++)
        printf("%d x 3 = %d \n",i,a[i]);
}
```

Apesar da solução apresentada acima ser correta logicamente, peca por desconsiderar a razão principal de um programa ser escrito em C, a velocidade. Intui-se facilmente que se endereçarmos uma variável por seu endereço, com certeza seremos mais velozes do que se a acessarmos pela tabela de variáveis (como feito no exemplo acima).

Antes de resolvermos novamente este problema, voltemos aos ponteiros neste pequeno programa, que servirá para entendermos a estratégia comumente usada pelos programadores C.



Tabuada ao Estilo C

```
main()
{
int a[10],i,*p;
p = &a;
for(i=1;i<10;i++)
     *(ponteiro+i) = i*3;
for(i=0;i<10;i++)
       printf("%d x 3 = %d \n",i,a[i]);
}
```

Entradas e Saídas

Além das funções "printf" e "scanf", muito poderosas existem outras instruções de entrada e saída, porém de menores recursos e ocupando menos espaço em memória.

Instruções	Descrição
getchar()	lê um caracter do teclado aguardando <Enter>
getche()	lê um caracter do teclado e prossegue
getch()	lê um caracter sem eco na tela e prossegue
putchar()	escreve um caracter na tela
gets()	lê uma string do teclado
puts()	escreve uma string na tela

Todas as instruções vistas anteriormente tem como argumento uma variável do tipo necessário (caracter ou seqüência de caracteres).

Exemplo 1: Escrever "A" de diversas formas.

```
main()
{
int i;
char c;
i = 65;
c = 'A';
putchar(65);
putchar('A');
putchar(i);
putchar('\n');
}
```

A linguagem C é pródiga em "confundir" dispositivos, como podemos constatar no exemplo a seguir.

Exemplo 2 Teclado é Arquivo!

```
main()
{
char s[30];
int c;
while ((c=getchar()) != EOF)
        putchar(c);
get(s);
puts(c);
}
```

Revisando o comando "scanf", agora estamos em condições de justificar o porquê devemos endereçar as variáveis com &. O sinal % na frente dos especificadores de entrada serve para informar o tipo de dado a ser lido. Um caracter não branco na string faz com que "scanf" leia e desconsidere um caracter coincidente. Por exemplo, a string de controle "%d,%d" faz com que "scanf" primeiro leia um inteiro, depois leia e desconsidere um vírgula e, finalmente, leia outro inteiro. Se não houver ',', "scanf" será encerrada.

O comando "scanf" deve receber valores passados por seus endereços. Devemos lembrar que C trabalha desta forma para chamadas por referência, permitindo que uma função altere o conteúdo de um argumento, como em "scanf("%d",&cont);".

Para leitura de strings (matrizes de caracteres), o nome da matriz sem índice informa o "endereço do primeiro elemento" da matriz, ou seja um ponteiro, portanto não devemos usar &, como em "scanf("%s",matriz);".

Os programadores BASIC devem ter em mente que soluções como as separações por vírgula no comando "INPUT" não funcionaram adequadamente em instruções do tipo "scanf("%d %d",&r,&c);".

No exemplo a seguir o 't' (se digitado!) será descartado, 10 ficará em x e 20 em y como em "scanf("%st%s",&x,&y);".

Outro cuidado a ser tomado pode ser constatado na expressão a seguir "scanf("%s ",dado);", somente após você digitar um caracter após um caracter branco. Isto ocorre pois "%s " instruiu "scanf()" a ler e desconsiderar espaços.

Os usuários BASIC (e alguns do Clipper) tem o seguinte costume:

```
INPUT"Digite um Número: ";A%
```

Analogamente pensa-se (de forma errada) que podemos fazer:

```
scanf("Digite um número %d ",&a);
```

- algo totalmente sem sentido em C.
- Teríamos a seguinte estrutura equivalente:

```
puts("Digite um número: ");
scanf("%d",&a);
```

Principais Códigos de Formatação de scanf e printf

Código	Significado	Observações
%c	lê/escreve um único caracter	
%d	lê/escreve inteiro decimal	
%i	lê/escreve inteiro decimal	
%e	lê/escreve número com ponto flutuante	Notação Científica
%f	lê/escreve número com ponto flutuante	
%h	lê/escreve um inteiro curto	
%o	lê/escreve um número octal	
%s	lê/escreve uma string	
%x	lê/escreve um número hexadecimal	
%p	lê/escreve um ponteiro	
%n	receber um valor inteiro igual ao número de caracteres lidos até o momento	
número	especificará o total de caracteres a serem lidos para um campo qualquer	
%g	Usa %e ou %f, aquele que for menor	Somente printf
%%	Imprime %	Somente printf
%u	Decimal sem Sinal	Somente printf
h	Modificador short	
l	Modificador long	

Exemplificando:

%ld- especifica um inteiro longo.

Os modificadores "acompanham" o outro código de formato, modificando sua apresentação.

Ponteiros x Matrizes e Entradas e Saídas (Aula 18L)

1- Processar Exemplos vistos em teoria.

2- Retome exercícios anteriores, utilizando as instruções de entrada e saída getchar(), getche(), getch(), putchar(), gets() e puts().