Mais recursos da linguagem C

(incluindo exemplos do Kernighan & Ritchie)

9ª série de exercícios

1. Goto e rótulos *("labels")*

O comando goto era muito usado nos primórdios das linguagens de programação.
Hoje em dia, seu uso deve ser evitado, pois torna o programa difícil de ser depurado e analisado.
Mas há casos em que é mais conveniente, como quando precisamos "pular" ou sair de um trecho de programa muito interno.

```
// checar se ao arrays A[ ] e B[ ] tem um elemento em comum
for(i=0;i<n;i++)
  for(j=0;j<m;j++)if(A[i] == B[j])goto found;
// didn't find any common element
printf("\nThe intersection between A and B is EMPTY\n");
// there is at least one common element
found: printf("\nThe intersection between A and B is NOT EMPTY\n");
```

EXERCÍCIO 1: Por que o comando *break* não funciona no exemplo acima?

EXERCÍCIO 2: Re-escreva o trecho acima sem usar o comando goto.

EXERCÍCIO 3: Escreva um trecho de programa em C para colocar a intersecção de A[ ] com B[ ] no array C[ ].

EXERCÍCIO 4: Escreva um trecho de programa em C para colocar a união de A[ ] com B[ ] no array C[ ].

2. Recursividade

Uma função no C pode chamar a si mesma. Isso é chamado de "recursividade", e é muito útil em certos casos. Deve ser usada apenas após uma análise muito rigorosa, que quase sempre é bastante complicada de ser feita por um programador inexperiente.

A função abaixo calcula o fatorial de um número inteiro N:

```
int fact(int N){return (N==1) ? 1 : N*fact(N-1); }
```

Esta calcula o termo N da série de Fibonacci (1 1 2 3 5 8 13 21 ...) :

```
int f(int N){
   if( (N==2) || (N==1) )return 1;
   else return f(N-1)+f(N-2);
}
```

3. Ponteiros

Um ponteiro *(pointer)* armazena um endereço de memória.
Se **p** é um apontador, **p = &c** vai fazer com que **p** aponte para o endereço da variável **c**.
***p** acessa o conteúdo de quem **p** está apontando.
z = *p coloca o conteúdo de quem **p** aponta na variável **z**.

O trecho abaixo exemplifica o uso dos operadores & e * com apontadores:

```
int x = 1, y = 2, z[10];
int *ip;            // p é um apontador para um int

ip = &x;            // ip agora aponta para x
y = *ip;            // y agora é 1
*ip = 0;            // x agora é 0
ip = &z[0]          // ip agora aponta para z[0]
```

O exemplo abaixo ilustra o uso de apontadores nos argumentos de funções:

```
void swap_ERRADA(int x, int y){
   int temp;

   temp = x;
   x = y;
   y = temp;
}

void swap(int *x, int *y){
   int temp;

   temp = *x;
   *x = *y;
   *y = temp;
}

int main(){
    int a = 2, b = 3;
    ...
    swap_ERRADA(a,b); // do not interchange the contents of a and b
    swap(&a,&b); // ok
    .....
}
```

EXERCÍCIO 5: Escreva uma função void sete(int *x) que substitui x pelo maior múltiplo de sete que seja menor ou igual a x.

4. Arrays e operações aritméticas com ponteiros

Se p e q apontam para elementos de um mesmo array,

p++ faz com que p aponte para o próximo elemento
p +=i faz com que o ponteiro avance i elementos
p < q é verdadeiro se p aponta para um elemento com índice menor do que o que q aponta
p -q aponta para o elemento que está q atrás de p

Se A é declarado como um array, por exemplo, int A[10], então A é um ponteiro que aponta para A[0].
*(A+1) é o conteúdo de A[1]
*(A+i) é o conteúdo de A[i]
p = A+3 vai apontar para A[3]

O exemplo abaixo compara o uso de arrays na notação de ponteiros com a notação comum.

Na prática, os ponteiros são usados quando é preciso alocar memória para os arrays em tempo de execução, ou para programação que envolve operações mais sofisticadas ou detalhadas com a memória ou o com o hardware.

```
#include <stdio.h>

int main ()
{
   int A[3][4] = { {1, -1, 3, 2}, {0, 2, -3, 1}, {-1, 2, 0, 4} };
   int B[4]={-1, 0, 4, 2}, C[4];
   int i, j, k, t, *maior, M;

   // imprimir a matriz A - na notacao de apontadores
   printf("\nMatriz A:\n");
   for(i=0;i<=2;i++){
      for(j=0;j<=3;j++)printf("  %2d", *(*(A+i)+j));
      // NOTE que o conteudo de (A+i) eh um endereco !!!
      // (A+i) aponta para a linha i de A
      printf("\n");
   }
   // imprimir a matriz A - na notacao comum
   printf("\nMatriz A:\n");
   for(i=0;i<=2;i++){
      for(j=0;j<=3;j++)printf("  %2d", A[i][j]);
      printf("\n");
   }
```

```
   // imprimir o maior elemento de B - na notacao de apontadores
   maior = B;
   for(i=1;i<=3;i++)if (*(B+i) > *maior)maior = (B+i);
   printf("\nMaior elemento de B : %d\n", *maior);

   // imprimir o maior elemento de B - na notacao comum
   M = B[0];
   for(i=1;i<=3;i++)if (B[i] > M)M = B[i];
   printf("\nMaior elemento de B : %d\n", M);

   // multiplicando A.B - na notacao de apontadores
   for(i=0;i<=2;i++){ t = 0;
                for(k=0;k<=3;k++)t += *(*(A+i)+k)*(*(B+k));
                *(C+i) = t;
               }
   printf("\nMatriz A.B:\n");
   for(j=0;j<=2;j++)printf("  %2d\n", *(C+j));
   printf("\n");

   // multiplicando A.B - na notacao comum
   for(i=0;i<=2;i++){ t = 0;
                for(k=0;k<=3;k++)t += A[i][k]*B[k];
                C[i] = t;
               }
   printf("\nMatriz A.B:\n");
   for(j=0;j<=2;j++)printf("  %2d\n", C[j]);
   printf("\n");

   system("pause");     return(0);
}
```

EXERCÍCIO 6: Escreva uma função double media(double *A, int N) que retorna a média aritmética dos elementos do array A (de tamanho N). Use apenas a notação de ponteiros.

5. Estruturas

O C permite a declaração de dados estruturados, que contem vários componentes de diversos tipos.

Nos casos em que o uso de estruturas é de muita utilidade, deve-se considerar seriamente a linguagem C++. O C++ oferece as ferramentas adequadas para criar, manter e reusar estruturas avançadas de dados.

EXEMPLO em C: Uma dado estruturado para representar um ponto (x,y) no plano

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

// template
struct point{
  double x;
  double y;
};

// makepoint : constrói um ponto a partir de suas coordenadas
struct point makepoint(double x, double y){
   struct point temp;
   temp.x = x;
   temp.y = y;
   return temp;
}

// addpoint: soma dois pontos
// struct parameters are called by value, like any others
struct point addpoint(struct point p1, struct point p2){
    p1.x += p2.x;
    p1.y += p2.y;
    return p1;
}

// distance : distance from the origin
double distance(struct point p){ return sqrt(p.x*p.x+p.y*p.y); }

int main(){
// declaração de variáveis (instâncias)
struct point P, p1= {2.5, 3.4}, p2, p3, p4;
double d;
```

```
p2.x = 1.2; p2.y = -2.5;
p3 = makepoint(4.5, 7.3);

printf("\nEntre com as coordenadas (x,y) do ponto P:\n");
scanf("%lf %lf", &P.x, &P.y);
printf("\n    P(%g,%g)\n", P.x, P.y);
d = distance(P);
printf("\n P is %g from the origin\n", d);

p2 = addpoint(P, p1);
printf("\n(%g,%g) + (%g,%g) = (%g,%g)\n", P.x, P.y, p1.x, p1.y, p2.x, p2.y);

system("pause");
return(0);
}
```

EXERCÍCIO 7: Escreva um programinha para lidar com vetores no plano na forma polar (r, θ). r é o módulo do vetor e θ o ângulo com o eixo x no sentido anti-horário. O programa deve somar dois vetores (r_1, θ_1) e (r_2, θ_2), imprimindo o módulo e o ângulo da soma. O ângulo deve ser informado no intervalo $(-180^o, 180^o)$.

6. FILE IO

EXEMPLO: O programa abaixo lê um numero por linha do arquivo notas.txt do disco, e imprime a média no arquivo resultado.txt.

```
#include <stdio.h>

int main(){
  FILE  *DATAfile, *RESULTfile;
  char DATAfilename[80], st[80];
  char RESULTfilename[] = "resultado.txt";
  int i, N;
  double s, x, media;

  printf("\nQual o nome do arquivo de dados?\n");
  i=0;
  while((DATAfilename[i] = fgetc(stdin)) != '\n')i++;
  DATAfilename[i] = '\0';
  printf("\n    %s\n", DATAfilename);

  /* OPEN DATA FILE */
  // checando se o arquivo existe mesmo
  if ((DATAfile = fopen(DATAfilename, "rt")) == NULL)
  {
    printf("\n*** O arquivo %s nao existe!!!\n", DATAfilename);
    system("pause");
    return 1;
  }
  // ler as notas, uma por linha, e somar todas
  i = 1; s= 0.0;
  while(fscanf(DATAfile,"%lf", &x) != EOF){printf("\n x = %g", x); s += x; i++;}
  N = --i;

 // calcular a media
 media = s/N;
 // verificar se o arquivo de saida ja existe...
  if((RESULTfile = fopen(RESULTfilename, "rt")) != NULL)
 {
   printf("\n*** O arquivo %s ja existe... \n", RESULTfilename);
   system("pause");
   return 2;
 }
 // reabrir o arquivo de saida para escrever
 RESULTfile = fopen(RESULTfilename, "wt");
 // escrever o resultado
 fprintf(RESULTfile, "Media de %d notas = %g", N, media);

 system("pause");
 return(0);
}
```


Quando se deseja escrever um grande numero de dados em um arquivo, pode ser interessante usar o IO binário, onde os números são escritos na forma de máquina, sem conversão alguma. Consulte um manual de C para isso.