Fazer um programa que, chamando a função imprime de forma iterativa, gere a tela:

```
// gera linhas contendo seqüência de letras
#include <stdio.h>
#include <stdlib.h>
void imprime(char, int); // protótipo: apenas com tipo dos parâmetros
int main()
 {
    int num=10;
    char letra='a';
    system ("color F1");
    ??????
    system("pause >> null");
    return 0;
 }
void imprime(char c, int n) // declaração da função void
{
    int cont; // variável local de controle do laço
    for (cont = 0; cont < n; cont++)
        printf("%c",c);
}
```

Fazer um programa que, chamando a função imprime de forma iterativa, gere a tela:

```
// gera linhas contendo seqüência de letras
#include <stdio.h>
#include <stdlib.h>
void imprime(char, int); // protótipo: apenas com tipo dos parâmetros
int main()
 {
    int num;
    char letra='a';
    system ("color F1");
    for(num=1;num<=15;num++,letra++)
          imprime(letra,num);
    system("pause >> null");
    return 0;
 }
void imprime(char c, int n) // declaração da função void
{
    int cont; // variável local de controle do laço
    for (cont = 0; cont < n; cont++)
        printf("%c",c);
    printf("\n");
}
```

Fazer um programa que, chamando a função imprime de forma iterativa, gere a tela:

```
// gera linhas contendo seqüência de letras
#include <stdio.h>
#include <stdlib.h>
void imprime(char, int); // protótipo: apenas com tipo dos parâmetros
int main()
 {
    int num;
    char letra;
    system ("color F1");
    for(num=1,letra='a'; num<=15; num++,letr
          imprime(letra,num);
    system("pause >> null");
    return 0;
 }
void imprime(char c, int n) // declaração da função void
{
    int cont; // variável local de controle do laço
    for (cont = 0; cont < n; cont++)
        printf("%c",c);
    printf("\n");
}
```

Formas de passagem de valores de parâmetros

- Por valor:
  - passa somente o valor.
  - para cada parâmetro, é alocada uma área de uso local da função e o valor do argumento é copiado para esta área (valor inicial).
  - alterações deste valor, durante o processamento da função, não alteram o valor da variável enviada pelo programa que chamou a função (argumento ou parâmetro real), apenas a cópia local.

Formas de passagem de valores de parâmetros

- Por referência:
  - passa um endereço de memória, o qual é associado ao parâmetro, localmente.
  - alterações feitas no parâmetro afetam o valor da variável (é a própria variável) referida na chamada (argumento ou parâmetro real).
  - para passar parâmetros por referência precisamos usar ponteiros.

Subprogramas que trocam valores de variáveis

```
procedimento troca (x, y)
// x e y inteiros
variável temp // inteiro, local
início
  temp ← x
  x ← y
  y ← temp
fim subprograma troca
```

- pode (e deve) declarar variáveis dentro do subprograma
- são chamadas variáveis locais

```
programa trocavarios
variáveis a, b - inteiros
          v (10) - inteiros
início
  ler a, b
  executar troca (a, b)
  escrever a, b
  preencher v (10) por leitura
  executar troca ( v[1], a )
  escrever v[1]
...
```

Exemplo 1: passagem por valor

- Fazer um procedimento que receba 2 inteiros x e y e troque seus valores.

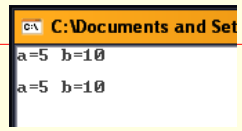
```
void troca(int x, int y)
{
    int temp;

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

```
int main()
{

    int a=5, b=10;
    printf("a=%d b=%d\n",a,b);
    troca(a,b);
    printf("a=%d b=%d\n",a,b);
    .....
}
```

O que será impresso na tela?



- Os valores não foram trocados, porque a passagem de parâmetros foi feita por valor.
- Dentro do subprograma, os valores de x e y foram trocados, mas esta mudança foi feita sobre as cópias locais dos valores e não foi propagada para fora do subprograma.

Exemplo 2: passagem por referência

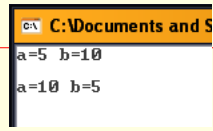
- Fazer um procedimento que receba 2 inteiros x e y e troque seus valores.

```
void troca(int *x, int *y)
{
    int temp;

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

```
int main()
{
    int a=5, b=10;
    printf("a=%d b=%d\n",a,b);
    troca(&a,&b);
    printf("a=%d b=%d\n",a,b);
    getch();
}
```

x e y agora recebem os endereços de a e b



- Os valores foram trocados, pois a passagem de parâmetros foi feita por referência.
- Dentro do subprograma, x e y receberam os endereços de memória de a e b.

Como funcionam os ponteiros - tipo

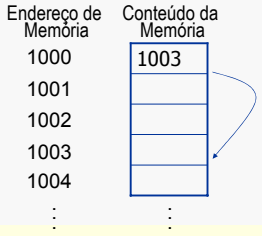
INT guarda inteiros
FLOAT guarda número de ponto flutuante
CHAR guarda caractere

PONTEIROS guardam endereços de memória

- O endereço é a posição de uma outra variável na memória
- Se uma variável **a** contém o endereço de uma variável **b**, dizemos que **a** aponta para **b**

O que são ponteiros?

- Ponteiros são variáveis que guardam endereços de memória.



Declaração de Ponteiros

Sintaxe:

<tipo> *<nome_da_variavel ponteiro>;

- operador especial, para indicar que a variável armazena um endereço de memória.
- qualquer tipo válido em C

Exemplos:

```
float *f; // f é um ponteiro para variáveis do tipo float
int *p;  // p é um ponteiro para variáveis do tipo inteiro
```


Operadores de Ponteiros

& operador unário, que devolve o endereço de memória de seu operando

Exemplo:

```
int count, q, *m;
m = &count; //coloca em m o endereço de memória da variável count
```

- Lê-se: "m recebe o endereço de count"
- Se a variável count está armazenada na posição de memória 2000, o valor de m será 2000.

Operadores de Ponteiros

***** operador unário que devolve o valor da variável localizada no endereço que o segue

Exemplo:

```
int count, q, *m;
q = *m; //coloca em q o valor contido no endereço de memória apontado por m
```

- Lê-se: "q recebe o valor que está no endereço m"
- Se a variável m aponta para um endereço que contém o valor 100, o valor de q será 100.

Exemplo 1

```
#include <stdio.h>
#include <stdlib.h>
int main()
{
   int count, q, *m;

   count = 100;
   m = &count;    // recebe endereço de count
   q = *m;        // recebe valor apontado por m
   printf("count = %d\n", count);
   printf("m = %p\n", m);   // %p para imprimir ponteiro (endereço apontado)
   printf("q = %d\n", q);
   printf("m aponta para %d\n\n", *m); // conteúdo apontado por m
   system("pause");
   return 0;
}
```

```
C:\Cora\Disciplinas\INF
count = 100
m = 0022FF74
q = 100
m aponta para 100

Pressione qualquer tecla para conti
```


Exemplo 2 – Onde está o problema?

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

int main()
{
   float x;
   int *p;

   x = 100;
   p = &x;
   printf("x=%f\p=%p\n\n", x,p);
   system("pause");
}
```

Erro de compilação: tipos incompatíveis
p é um ponteiro para inteiros, logo não pode apontar para uma variável do tipo float.

Atribuições com ponteiros

Como qualquer variável, um ponteiro pode ser usado no lado direito de um comando de atribuição para passar seu valor para um outro ponteiro.

Exemplo:

```
int x = 200, *p1, *p2;
p1 = &x;  // p1 aponta para x
p2 = p1;  // p2 recebe p1 e também passa a apontar para x
printf("x - %d, p1=%p e p2=%p\n", x, p1, p2);    // endereço de x
printf("x - %d, p1=%d e p2=%d\n",x,*p1, *p2);  // conteúdo de x
```

```
x - 200, p1=0022FF60 e p2=0022FF60
x - 200, p1=200 e p2=200
```

Exemplo 3 – O que será impresso na tela?

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

int main()
{
      int a, *x, *y;
      a = 10;
      x = &a;
      y = x;
      printf("a = %d, *x = %d, *y = %d \n",a,*x, *y);
      printf("a = %d, x = %p, y = %p\n\n", a, x, y);
      *x = 20;
      printf("a = %d, *x = %d, *y = %d \n",a, *x, *y);
      printf("a = %d, x = %p, y = %p\n\n", a, x, y);
      system("PAUSE>>null");
      return 0;
}
```

```
C:\Cora\Disciplinas\INF
a = 10, *x = 10, *y = 10
a = 10, x = 0022FF60, y = 0022FF60

a = 20, *x = 20, *y = 20
a = 20, x = 0022FF60, y = 0022FF60
```