parâmetros são as variáveis, **Dias** e **Anos**, das quais devem ser lidos os valores do inteiro e do **float**, respectivamente.

AUTO AVALIAÇÃO

1 - Veja como você está. O que faz o seguinte programa?

```
#include <stdio.h>
int main()
{
  int x;
  scanf("%d",&x);
  printf("%d",x);
  return(0);
}                    }
```

2 - Compile e execute os programas desta página

Introdução às Funções

Uma função é um bloco de código de programa que pode ser usado diversas vezes em sua execução. O uso de funções permite que o programa fique mais legível, mais bem estruturado. Um programa em C consiste, no fundo, de várias funções colocadas juntas.

Abaixo o tipo mais simples de função:

```
#include <stdio.h>
int mensagem ()  /* Funcao simples: so imprime Ola! */
{
 printf ("Ola! ");
 return(0);
}
int main ()
{
 mensagem();
 printf ("Eu estou vivo!\n");
 return(0);
}
```

Este programa terá o mesmo resultado que o primeiro exemplo da seção anterior. O que ele faz é definir uma função **mensagem()** que coloca uma string na tela e retorna 0. Esta função é chamada a partir de **main()** , que, como já vimos, também é uma função. A diferença fundamental entre main e as demais funções do problema é que main é uma função especial, cujo diferencial é o fato de ser a primeira função a ser executada em um programa.

- Argumentos

Argumentos são as entradas que a função recebe. É através dos argumentos que passamos *parâmetros* para a função. Já vimos funções com argumentos. As funções **printf()** e **scanf()** são funções que recebem argumentos. Vamos ver um outro exemplo simples de função com argumentos:

```
#include <stdio.h>
int square (int x)  /* Calcula o quadrado de x */
{
  printf ("O quadrado e %d",(x*x));
 return(0);
}
int main ()
{
   int num;
   printf ("Entre com um numero: ");
   scanf ("%d",&num);
   printf ("\n\n");
   square(num);
    return(0);
}
```

Na definição de **square()** dizemos que a função receberá um argumento inteiro **x**. Quando fazemos a chamada à função, o inteiro **num** é passado como argumento. Há alguns pontos a observar. Em primeiro lugar temos de satisfazer aos requisitos da função quanto ao tipo e à quantidade de argumentos quando a chamamos. Apesar de existirem algumas conversões de tipo, que o C faz automaticamente, é importante ficar atento. Em segundo lugar, não é importante o nome da variável que se passa como argumento, ou seja, a variável **num**, ao ser passada como argumento para **square()** é copiada para a variável **x**. Dentro de **square()** trabalha-se apenas com **x**. Se mudarmos o valor de **x** dentro de **square()** o valor de **num** na função **main()** permanece inalterado.

Vamos dar um exemplo de função de mais de uma variável. Repare que, neste caso, os argumentos são separados por vírgula e que deve-se explicitar o tipo de cada um dos argumentos, um a um. Note, também, que os argumentos passados para a função não necessitam ser todos variáveis porque mesmo sendo constantes serão copiados para a variável de entrada da função.

```
#include <stdio.h>

 int mult (float a, float b,float c)   /* Multiplica 3
numeros */
{
    printf ("%f",a*b*c);
   return(0);
}

int main ()
{
   float x,y;
   x=23.5;
   y=12.9;
   mult (x,y,3.87);
    return(0);
}
```

- Retornando valores

Muitas vezes é necessário fazer com que uma função retorne um valor. As funções que vimos até aqui estavam retornando o número 0. Podemos especificar um tipo de retorno indicando-o antes do nome da função. Mas para dizer ao C *o que* vamos retornar precisamos da palavra reservada **return**. Sabendo disto fica fácil fazer uma função para multiplicar dois inteiros e que retorna o resultado da multiplicação. Veja:

```
#include <stdio.h>
int prod (int x,int y)
{
   return (x*y);
}

int main ()
{
    int saida;
    saida=prod (12,7);
    printf ("A saida e: %d\n",saida);
    return(0);
}
```

Veja que, como prod retorna o valor de 12 multiplicado por 7, este valor pode ser usado em uma expressão qualquer. No programa fizemos a atribuição deste resultado à variável saida, que posteriormente foi impressa usando o printf. Uma observação adicional: se não especificarmos o tipo de retorno de uma função, o compilador C automaticamente suporá que este tipo é inteiro. Porém, não é uma boa prática não se especificar o valor de retorno e, neste curso, este valor será sempre especificado.

Com relação à função main, o retorno sempre será inteiro. Normalmente faremos a função main retornar um zero quando ela é executada sem qualquer tipo de erro.

Mais um exemplo de função, que agora recebe dois floats e também retorna um float::

```
#include <stdio.h>
float prod (float x,float y)
{
     return (x*y);
}

int main ()
{
     float saida;
     saida=prod (45.2,0.0067);
     printf ("A saida e: %f\n",saida);
     return(0);
}
```

- Forma geral

Apresentamos aqui a forma geral de uma função:

tipo_de_retorno nome_da_função (lista_de_argumentos)
{
* código_da_função*
}

AUTO AVALIAÇÃO

Veja como você está. Escreva uma função que some dois inteiros e retorne o valor da soma.

Introdução Básica às Entradas e Saídas

- Caracteres

Os caracteres são um tipo de dado: o **char**. O C trata os caracteres ('a', 'b', 'x', etc ...) como sendo variáveis de um *byte* (8 *bit*s). Um *bit* é a menor unidade de armazenamento de informações em um computador. Os inteiros (**int**s) têm um número maior de *bytes*. Dependendo da implementação do compilador, eles podem ter 2 *bytes* (16 *bits*) ou 4 *bytes* (32 *bits*). Isto será melhor explicado na aula 3. Na linguagem C, também podemos usar um **char** para armazenar valores numéricos inteiros, além de usá-lo para armazenar caracteres de texto. Para indicar um caractere de texto usamos apóstrofes. Veja um exemplo de programa que usa caracteres:

```
#include <stdio.h>
int main ()
{
        char Ch;
        Ch='D';
        printf ("%c",Ch);
        return(0);
}
```

No programa acima, **%c** indica que **printf()** deve colocar um caractere na tela. Como vimos anteriormente, um **char** também é usado para armazenar um número inteiro. Este número é conhecido como o código ASCII correspondente ao caractere. Veja o programa abaixo:

```
#include <stdio.h>
int main ()
{
  char Ch;
  Ch='D';
  printf ("%d",Ch);  /* Imprime o caracter como inteiro */
  return(0);
}
```

Este programa vai imprimir o número 68 na tela, que é o código ASCII correspondente ao caractere 'D' (d maiúsculo).

Muitas vezes queremos ler um caractere fornecido pelo usuário. Para isto as funções mais usadas, quando se está trabalhando em ambiente DOS ou Windows, são **getch()** e **getche()**. Ambas retornam o caractere pressionado. **getche()** imprime o caractere na tela antes de retorná-lo e **getch()** apenas retorna o caractere pressionado sem imprimí-lo na tela. Ambas as funções podem ser encontradas no arquivo de cabeçalho **conio.h**. Geralmente estas funções **não estão disponíveis em ambiente Unix** (compiladores cc e gcc), pois não fazem parte do padrão ANSI. Podem ser substituídas pela função scanf(), porém sem as mesmas funcionalidades. Eis um exemplo que usa a função **getch(),** e seu correspondente em ambiente Unix:

```
#include <stdio.h>
#include <conio.h>
/* Este programa usa conio.h . Se você não tiver a conio, ele
não funcionará no Unix */
int main ()
{
      char Ch;
      Ch=getch();
      printf ("Voce pressionou a tecla %c",Ch);
      return(0);
}
```

Equivalente ANSI-C para o ambiente Unix do programa acima, sem usar getch():

```
#include <stdio.h>
int main ()
{
      char Ch;
      scanf("%c", &Ch);
      printf ("Voce pressionou a tecla %c",Ch);
      return(0);
}
```

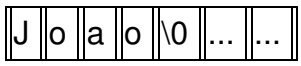
A principal diferença da versão que utiliza getch() para a versão que não utiliza getch() é que no primeiro caso o usuário simplesmente aperta a tecla e o sistema lê diretamente a tecla pressionada. No segundo caso, é necessário apertar também a tecla <ENTER>. **Lembre-se que, se você quiser manter a portabilidade de seus programas, não deve utilizar as funções getch e getche, pois estas não fazem parte do padrão ANSI C !!!**

- Strings

No C uma string é um vetor de caracteres terminado com um caractere nulo. O caracter nulo é um caractere com valor inteiro igual a zero (código ASCII igual a 0). O terminador nulo também pode ser escrito usando a convenção de barra invertida do C como sendo '\0'. Embora o assunto vetores seja discutido posteriormente, veremos aqui os fundamentos necessários para que possamos utilizar as strings. Para declarar uma string, podemos usar o seguinte formato geral:

char nome_da_string[tamanho];

Isto declara um vetor de caracteres (uma string) com número de posições igual a *tamanho*. Note que, como temos que reservar um caractere para ser o terminador nulo, temos que declarar o comprimento da string como sendo, no mínimo, um caractere maior que a maior string que pretendemos armazenar. Vamos supor que declaremos uma string de 7 posições e coloquemos a palavra João nela. Teremos:



No caso acima, as duas células não usadas têm valores indeterminados. Isto acontece porque o C *não* inicializa variáveis, cabendo ao programador esta tarefa. Portanto as únicas células que são inicializadas são as que contêm os caracteres 'J', 'o', 'a', 'o' e '\0' .

Se quisermos ler uma string fornecida pelo usuário podemos usar a função **gets()**. Um exemplo do uso desta função é apresentado abaixo. A função **gets()** coloca o terminador nulo na string, quando você aperta a tecla "Enter".

```
#include <stdio.h>
int main ()
{
    char string[100];
    printf ("Digite uma string: ");
    gets (string);
    printf ("\n\nVoce digitou %s",string);
    return(0);
}
```

Neste programa, o tamanho máximo da string que você pode entrar é uma string de 99 caracteres. Se você entrar com uma string de comprimento maior, o programa irá aceitar, mas os resultados podem ser desastrosos. Veremos porque posteriormente.

Como as strings são vetores de caracteres, para se acessar um determinado caracter de uma string, basta "indexarmos", ou seja, usarmos um índice para acessarmos o caracter desejado dentro da string. Suponha uma string chamada *str*. Podemos acessar a **segunda** letra de *str* da seguinte forma:

```
str[1] = 'a';
```

Por quê se está acessando a segunda letra e não a primeira? Na linguagem C, o índice ***começa em zero.*** Assim, a primeira letra da string sempre estará na posição 0. A segunda letra sempre estará na posição 1 e assim sucessivamente. Segue um exemplo que imprimirá a segunda letra da string "Joao", apresentada acima. Em seguida, ele mudará esta letra e apresentará a string no final.

```
#include <stdio.h>
int main()
{
    char str[10] = "Joao";
    printf("\n\nString: %s", str);
    printf("\nSegunda letra: %c", str[1]);
    str[1] = 'U';
    printf("\nAgora a segunda letra eh: %c", str[1]);
    printf("\n\nString resultante: %s", str);
    return(0);
}
```

Nesta string, o terminador nulo está na posição 4. Das posições 0 a 4, sabemos que temos caracteres válidos, e portanto podemos escrevê-los. Note a forma como inicializamos a string **str** com os caracteres 'J' 'o' 'a' 'o' e '\0' simplesmente declarando char str[10] = "Joao". Veremos, posteriormente que "Joao" (uma cadeia de caracteres entre aspas) é o que chamamos de string constante, isto é, uma cadeia de caracteres que está pré-carregada com valores que não podem ser modificados. Já a string str é uma string variável, pois podemos modificar o que nela está armazenado, como de fato fizemos.

No programa acima, **%s** indica que **printf()** deve colocar uma string na tela. Vamos agora fazer uma abordagem inicial às duas funções que já temos usado para fazer a entrada e saída.

- printf

A função **printf()** tem a seguinte forma geral:

printf (string_de_controle,lista_de_argumentos);

Teremos, na string de controle, uma descrição de tudo que a função vai colocar na tela. A string de controle mostra não apenas os caracteres que devem ser colocados na tela, mas também quais as variáveis e suas respectivas posições. Isto é feito usando-se os códigos de controle, que usam a notação **%**. Na string de controle indicamos quais, de qual tipo e em que posição estão as variáveis a serem apresentadas. É muito importante que, para cada código de controle, tenhamos um argumento na lista de argumentos. Apresentamos agora alguns dos códigos **%**:

Código	Significado
%d	Inteiro
%f	Float
%c	Caractere
%s	String
%%	Coloca na tela um %

Vamos ver alguns exemplos de **printf()** e o que eles exibem:

```
printf ("Teste %% %%") -> "Teste % %"
printf ("%f",40.345) -> "40.345"
printf ("Um caractere %c e um inteiro %d",'D',120) -> "Um
caractere D e um inteiro 120"
printf ("%s e um exemplo","Este") -> "Este e um exemplo"
printf ("%s%d%%","Juros de ",10) -> "Juros de 10%"
```

Maiores detalhes sobre a função **printf()** (incluindo outros códigos de controle) serão vistos posteriormente, mas podem ser consultados de antemão pelos interessados.

- scanf

O formato geral da função **scanf()** é:

scanf (string-de-controle,lista-de-argumentos);

Usando a função **scanf()** podemos pedir dados ao usuário. Um exemplo de uso, pode ser visto acima. Mais uma vez, devemos ficar atentos a fim de colocar o mesmo número de argumentos que o de códigos de controle na string de controle. Outra coisa importante é lembrarmos de colocar o **&** antes das variáveis da lista de argumentos. É impossível justificar isto agora, mas veremos depois a razão para este procedimento. Maiores detalhes sobre a função **scanf()** serão vistos posteriormente, mas podem ser consultados de antemão pelos interessados.

AUTO AVALIAÇÃO

Veja como você está:

a) Escreva um programa que leia um caracter digitado pelo usuário, imprima o caracter digitado e o código ASCII correspondente a este caracter.

b) Escreva um programa que leia duas strings e as coloque na tela. Imprima também a segunda letra de cada string.

Introdução a Alguns Comandos de Controle de Fluxo

Os comandos de controle de fluxo são aqueles que permitem ao programador alterar a sequência de execução do programa. Vamos dar uma breve introdução a dois comandos de controle de fluxo. Outros comandos serão estudados posteriormente.

- if

O comando **if** representa uma tomada de decisão do tipo "SE isto ENTÃO aquilo". A sua forma geral é:

if (condição) declaração;

A condição do comando **if** é uma expressão que será avaliada. Se o resultado for zero a declaração não será executada. Se o resultado for qualquer