Curso de C
Entrada e Saída por Console

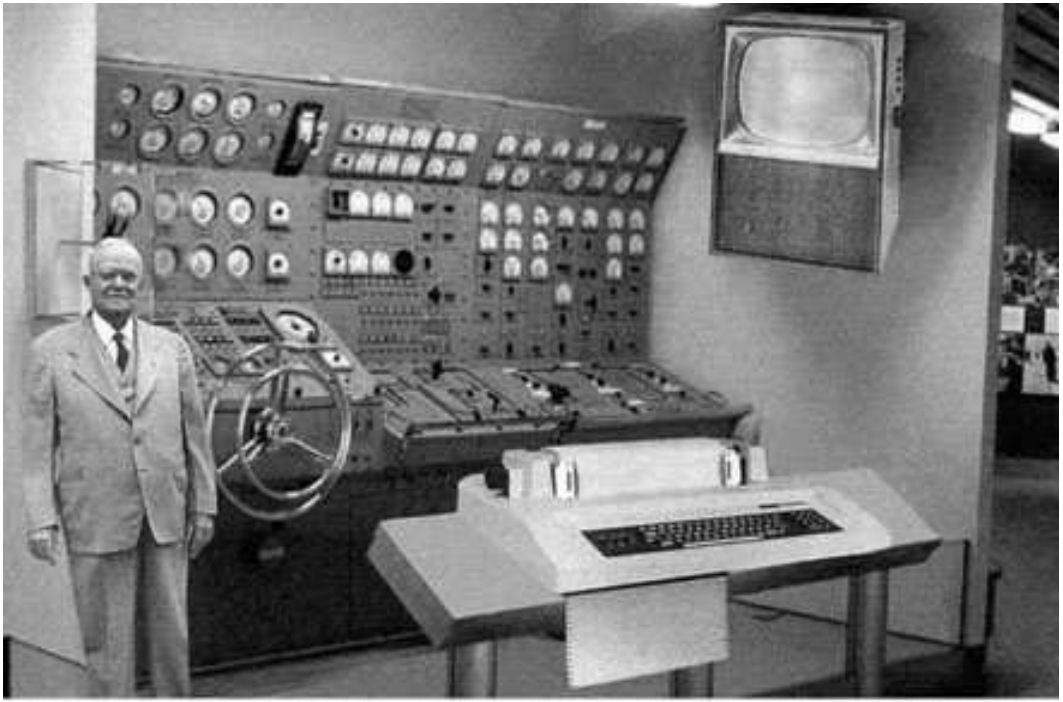
Adriano Cruz
adriano@nce.ufrj.br

15 de agosto de 2013

Section Summary

Introdução

Computador de casa de 2004 em 1950. Modelo Rand Corporation. Com a linguagem Fortran e o teletipo ele será fácil de ser usado.



Scientists from the RAND Corporation have created this model to illustrate how a "home computer" could look like in the year 2004. However the needed technology will not be economically feasible for the average home. Also the scientists readily admit that the computer will require not yet invented technology to actually work, but 50 years from now scientific progress is expected to solve these problems. With teletype interface and the Fortran language, the computer will be easy to use.

Entrada e Saída Padrão em C

1. A entrada de dados será feita pelo teclado e a saída poderá ser vista na tela do computador.
2. Em C, normalmente, três fluxos (arquivos) de dados são abertos para operações de entrada e saída:
   1. um para entrada,
   2. um para saída,
   3. um para imprimir mensagens de erro ou diagnóstico.

Entrada e Saída Padrão em C

1. Normalmente o fluxo de entrada está conectado ao teclado.
2. O fluxo de saída e o de mensagens de erro estão conectados ao monitor.
3. Estas configurações podem ser alteradas de acordo com as necessidades dos usuários (operações são chamadas de redirecionamento).
4. O fluxo de entrada é chamado de entrada padrão (*standard input*).
5. O fluxo de saída é chamado de saída padrão (*standard output*).
6. O fluxo de erros é chamado de saída padrão de erros (*standard error output*).
7. Formas abreviadas: `stdin`, `stdout` e `stderr`.

Bibliotecas

1. Linguagens de programação dispõem de bibliotecas de funções.
2. Estas biblitecas contém funções matemáticas, de processamento de imagens etc.
3. Elas permitem que usemos funções já testadas.
4. Economizam tempo de projeto.
5. Aumentam a confiabilidade dos sistemas.

Diretiva `include`

1. Para acessar as bibliotecas que contém estas funções o programa usa a diretiva **#include** `<umnomequalquer.h>` no início do programa.
2. Exemplo **#include**`<math.h>`
3. A diretiva **#include** instrui o compilador a ler o arquivo indicado entre '<' e '>', e processá-lo como se ele fosse parte do arquivo original.
4. Se o nome estiver entre < e >, ele será procurado em um diretório específico de localização pré-definida.
5. Quando se usa aspas o arquivo é procurado de maneira definida pela implementação, normalmente o diretório atual.
6. Exemplo **#include** `"minhasfuncoes.h"`
7. Programadores usam < e > para incluir os arquivos de cabeçalho padrão e aspas para a inclusão de arquivos do próprio projeto.

Atenção!

1. O arquivo `math.h` não contém o código das funções.
2. Este arquivo inclui somente informações úteis para o compilador.
3. O compilador descobre onde estão as funções das biblioteca usando outras dicas.
4. Por exemplo, bibliotecas ficam em diretórios padrão e o compilador procura nestes diretórios.
5. No Unix um diretório usado é `/usr/lib`
6. O compilador precisa ser avisado para usar uma determinada biblioteca.
7. Por exemplo, para compilar com a biblioteca matemática usamos:
8. `gcc -o prog prog.c -lm`

Biblioteca Padrão em C

1. Em C não existem comandos de entrada e saída.
2. Operações de entrada e saída são executadas com auxílio de variáveis, macros e funções especiais.
3. Para acessar à biblioteca que contém estas ferramentas o programa deve conter a declaração **#include** `<stdio.h>` no início do programa.

Section Summary

- `printf` faz com que dados sejam escritos na saída padrão, que normalmente é a tela do computador.
- O protótipo da função é:
  int `printf(controle, arg1, arg2, ...);`
- Os argumentos `arg1, arg2, ...` são impressos de acordo com o formato indicado pela cadeia de caracteres que compõe `controle`.
- O formato é ao mesmo tempo simples e bastante flexível
- A função retorna o número de caracteres impressos, não incluindo o `nulo` em vetores de caracteres.
- Um valor negativo é retornado no caso de erro.

Exemplo de printf

```c
#include <stdio.h>
int main (void) {
    int ano = 1997;

    /* Imprime o valor do ano */
    printf("Estamos no ano %d", ano);
    return 0;
}
```

Este programa irá imprimir na tela do computador:

```
Estamos no ano 1997
```

Cadeia de controle

- A cadeia de caracteres `controle` deve vir entre ".
- `"Estamos no ano %d"`
- Ela define como serão impressos os valores dos argumentos.
- No `controle` podem existir dois tipos de informações: caracteres comuns e códigos de formatação.
- Os caracteres comuns são escritos na tela sem nenhuma modificação.
- Os códigos de formatação, aparecem precedidos por um caractere '%' e são aplicados aos argumentos na ordem em que aparecem.
- Deve haver um código de formatação para cada argumento.
- O código `%d` indica que o valor armazenado em `ano` deve ser impresso na notação inteiro decimal.
- É importante notar que o campo de `controle` aparece somente uma vez na função `printf` e sempre no início.

Códigos de conversão

Código	Comentário
%c	Caracter simples
%d	Inteiro decimal com sinal
%i	Inteiro decimal com sinal
%E	Real em notação científica com E
%e	Real em notação científica com e
%f	Real em ponto flutuante
%G	%E ou %f, o que for mais curto
%g	%g ou %f, o que for mais curto
%o	Inteiro em base octal
%s	Cadeia Caracteres
%u	Inteiro decimal sem sinal
%x	Inteiro em base hexadecimal (letras minúsculas)
%X	Inteiro em base hexadecimal (letras maiúsculas)
%p	Endereço de memória
%%	Imprime o caractere %

Modificadores

- Entre o caractere % e o código de conversão podem ser inseridos caracteres que alteram o formato.
- Estes caracteres são chamados de modificadores, largura, precisão e comprimento.
  - %[modificadores][largura][.precisão][comprimento]código
- Todos os diferentes tipos de caracteres são opcionais.
- Exemplo:

```
#include<stdio.h>
int main (void)
{
    long double x = 1.0L;
    printf("%Lf\n", x);
    return 0;
}
```

Modificadores

- – um sinal de menos especifica que o argumento deve ser justificado à esquerda. Caso nenhum sinal seja usado o argumento será ajustado à direita.
- %-6d $\longrightarrow$ 2 0 ␣ ␣ ␣ ␣
- %6d $\longrightarrow$ ␣ ␣ ␣ ␣ 2 0
- + força que o resultado seja precedido por sinal de menos ou de mais. O padrão é que somente negativos sejam precedidos por sinal de menos.
- %+6d $\longrightarrow$ ␣ ␣ ␣ + 2 0
- espaço Caso nenhum sinal vá ser escrito, um espaço é inserido antes do valor.

Modificadores

- # usado com `o`, `x` ou `X` precede o valor com `0`, `0x` ou `0X` respectivamente para valores diferentes de zero.
- # usado com `e`, `E` e `f`, força que a saída contenha um ponto decimal mesmo que não haja parte fracionária. Por padrão, se não há parte fracionária o ponto decimal não é escrito.
- # usado com `g` ou `G` o resultado é o mesmo que com `e` ou `E`, mas os zeros finais não são retirados.
- 0 completa o campo, pela esquerda, com zeros (0) ao invés de espaços, sempre que a opção para completar seja especificada (ver especificador de largura do campo).
- `%6d` $\longrightarrow$ 000020

Largura

- `largura`: Caso seja usado um número inteiro, este especifica o tamanho mínimo do campo onde o argumento será impresso.
- Os espaços livres serão completados com espaços em branco.
- Se o argumento precisar de mais espaço que o especificado ele será escrito normalmente e o tamanho mínimo é ignorado.

Precisão

- `.precisão` Este número tem diferentes significados dependendo do código usado.
  - caracteres: no caso de impressão de cadeia de caracteres (`s`), este número especifica o número máximo de caracteres de uma cadeia de caracteres a serem impressos.
  - ponto flutuante: no caso de formato (`e, E, f`) é o número de dígitos a serem impressos a direita do ponto, ou seja o número de casas decimais.
  - Para o formato `g` ou `G` é o número máximo dígitos significativos.
  - inteiros: nos formatos inteiros (`d, i, o, u, x, X`) a precisão especifica o número máximo de dígitos a serem impressos. Se o número de caracteres a serem impressos é menor que este o resultado é completado com brancos. O valor não é truncado

Comprimento

- comprimento Uma letra neste campo modifica os formatos da seguinte maneira:
  - `l` Aplicado aos formatos de tipo `d`, `i`, `o`, `u`, `x` e `X` indicando que o dado é do tipo `long int` e não `int`.
  - `h` Modifica o dado, nos formatos `d`, `i`, `o`, `u`, `x` e `X` para tipo `short int`.
  - `L` Nos formatos `e`, `E`, `f`, `g` e `G` o argumento é modificado para `long double`.

Justificação da impressão

```
#include <stdio.h>
int main( void)
{
    int ano = 1997;

    printf("Justificado direita Ano = %8d\n", ano);
    printf("Justificado esquerda Ano = %-8d\n", ano)↩
        ;

    return 0;
 }
```

Precisão

```c
#include <stdio.h>
int main()
{
    float r = 1.0/3.0;
    char s[] = "Alo Mundo";

    printf("O resultado e = %9.3f\n", r);
    printf("%9.3s\n", s);

    return 0;
 }
```

Section Summary

scanf

- A função `scanf` é utilizada para entrada de dados a partir do teclado.
- Protótipo: `scanf(controle, arg1, arg2, ...);`
- Uma diferença fundamental que existe entre `scanf` e `printf` está nos argumentos que vêm depois do `controle`.
- No caso de `scanf` os argumentos são os endereços das variáveis que irão receber os valores lidos.
- Em `printf` são as próprias variáveis.
- A indicação que estamos referenciando um endereço e não a variável se faz pelo operador `&`.
- O comando `scanf("%d %d", &a, &b);` espera que dois valores inteiros sejam digitados no teclado.

Exemplo `scanf`

- Em `scanf` procure usar somente códigos de conversão similares aos usados em `printf`.

```
int i;
float x;
printf("Entre com um inteiro e um real\n");
scanf("%d %f", &i, &x);
```

- Se precisar escrever mensagens use uma outra função.
- No exemplo, `printf`.
- Não use vírgulas para separar os códigos de formatação.

Controle

- Se for realmente necessário, no scanf podem ser usados:
- **Caracteres branco:** `scanf` lê e ignora todos os caracteres `branco` e/ou `<enter>` e/ou `tab` que aparecerem antes de qualquer caractere diferente destes.
- **Caracteres comuns:** (exceto %) devem casar com o próximo caractere diferente de branco da entrada.
- Isto significa que qualquer caractere que não for igual a branco e/ou `<enter>` e/ou `tab` ou parte de um especificador de formato faz com que a função leia o próximo caractere da entrada (`stdin`) e se for igual a este ele é descartado.
- Caso os caracteres sejam diferentes a função falha e retorna deixando os caracteres seguintes não lidos.
- A função `scanf` retorna a quantidade de dados lidos corretamente.

Exemplo scanf

```
#include<stdio.h>
int main (void)
{
    int dia, mes, ano;
    int lidos;

    printf("Entre com uma data na formato dd/mm/aaaa↩
       \n");
    lidos = scanf("%d/%d/%d", &dia, &mes, &ano);
    printf("Li %d dados\n", lidos);
    printf("Li a data %02d --- %02d --- %4d\n",
                                dia, mes, ano);
    return 0;
}
```

Exemplo scanf

```
Entre com uma data na formato dd/mm/aaaa
1/12/2000
Li 3 dados
Li a data 01 --- 12 --- 2000
```

Lendo caracteres

```
#include  <stdio.h>
int main () {
    char c;
    int num1, num2;

    printf("Entre com um caractere qualquer.\n");
    scanf("%c", &c);
    printf("ASCII do caractere %c vale %d.\n", c, c)↩
        ;
    printf("Entre dois inteiros.\n");
    scanf("%d %d", &num1, &num2);
    printf("A soma deles vale %d.\n", num1+num2);
    return 0;
}
```

Section Summary

Lendo Caracteres

- Para ler e escrever caracteres do teclado as funções de entrada e saída mais simples são `getchar` e `putchar`.
- Elas estão na biblioteca `stdio.h`.
- Seus protótipos são os seguintes: `int getchar(void);` e `int putchar(int c);`

Exemplo

```c
#include <stdio.h>
int main (void)
{
    char c;

    printf("Entre com um algarismo entre 0 e 9.\n");
    c = getchar();

    printf("O caractere lido foi o ");
    putchar(c);

    return 0;
}
```

Atenção

- Observar que, normalmente, quando algum dado é fornecido pelo teclado termina-se a digitação com a tecla <enter>.
- No entanto, o <enter> é um caractere também, e isto pode causar problemas.
- Exemplo de problema: ler um inteiro e depois um caractere.
- Digita-se um inteiro e depois o <enter>.
- O <enter> não é lido e fica em um *buffer*.
- A próxima leitura de caracter irá ler o caractere <enter>.

Atenção - programa

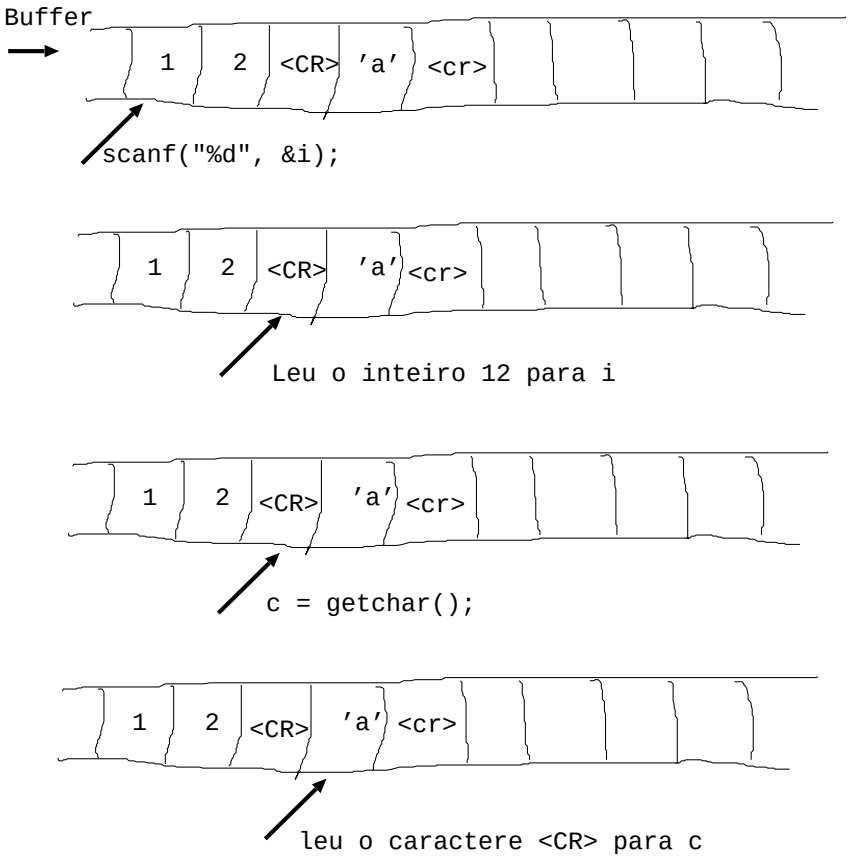
```c
#include <stdio.h>
int main (void)
{
    char c;
    int i;

    printf("Entre com um numero inteiro.\n");
    scanf("%d", &i);

    printf("Agora um caractere.\n");
    c = getchar();

    printf("Numero lido %d\n", i);
    printf("Caractere lido %c\n", c);
    return 0;
}
```

O Buffer


Buffer
1
2
<CR>
'a'
<cr>
scanf("%d", &i);
1
2
<CR>
'a'
<cr>
Leu o inteiro 12 para i
1
2
<CR>
'a'
<cr>
c = getchar();
1
2
<CR>
'a'
<cr>
leu o caractere <CR> para c

Section Summary

The End