Programação Estruturada

Aula 2 - printf e scanf

Yuri Malheiros (yuri@ci.ufpb.br)

Introdução

- printf é a função usada para escrever dados
- scanf é a função usada para ler dados
- Elas são duas das funções mais usadas na linguagem

printf

- Usamos printf para exibir o conteúdo de um string
- Ela suporta adicionar valores em determinados pontos do string
- Os valores exibidos podem ser constantes, variáveis ou expressões

printf

- Os dados adicionados ao string são determinados pelas **especificadores de conversão**
    - São as partes que começam com `%`
- A informação após o `%` determina como o `printf` deve exibir o valor
    - `%d` - especifica que o valor deve exibido como inteiro
    - `%f` - especifica que o valor deve exibido como um float

printf

```c
#include <stdio.h>

int main(void) {
    int i=10, j=20;
    float x=43.2892, y=5527.0;

    printf("i=%d, j=%d, x=%f, y=%f\n", i, j, x, y);

    return 0;
}
```

printf

- O compilador pode funcionar mesmo que existam alguns erros nos especificadores de conversão
    - Ele deve gerar *warnings*, mas não erros

```c
#include <stdio.h>

int main(void) {
    int i=10, j=20;
    float x=43.2892, y=5527.0;

    printf("%d %d", i);

    return 0;
}
```

printf

- O `printf` fornece um grande controle sobre a aparência da sua saída
- Vamos nos aprofundar um pouco na exibição dos floats
- Já vimos que podemos especificar a quantidade de dígitos (N) depois do . de um float usando: `%.Nf`

printf

- De forma mais geral, podemos ter conversões com o formato `%M.NX`
  - `M` e `N` são opcionais
  - `X` pode ser `f` ou `d` ou outros...
- `M` especifica o número mínimo de caracteres que deve ser exibido para o valor
- Se o valor tem menos de `M` caracteres, então ele é justificado à direita
  - Espaços são colocados do lado esquerdo para que o valor tenha `M` caracteres

printf

```c
#include <stdio.h>

int main(void) {
    int i=10;

    printf("%5d\n", i);

    return 0;
}
```

- 10 tem dois caracteres, então três espaços serão adicionados à esquerda

printf

- A utilidade dessa formatação fica mais clara quando temos mais valores:

```c
#include <stdio.h>

int main(void) {
    printf("%5d\n", 3);
    printf("%5d\n", 15);
    printf("%5d\n", 98);
    printf("%5d\n", 1500);
    printf("%5d\n", 458);
    printf("%5d\n", 12);

    return 0;
}
```

printf

- Para justificar à esquerda, colocamos um `-` antes do `M`:

```c
#include <stdio.h>

int main(void) {
    printf("%-5d\n", 3);
    printf("%-5d\n", 15);
    printf("%-5d %-5d\n", 98, 1500);
    printf("%-5d %-5d\n", 458, 12);

    return 0;
}
```

printf

- Isto também vale para os floats

```c
#include <stdio.h>

int main(void) {
    printf("%10f\n", 3.5237);
    printf("%10f\n", 153.826);
    printf("%10f\n", 98.128974);
    printf("%10.2f\n", 1500.2);
    printf("%10.3f\n", 458.999);
    printf("%10f\n", 12.25);

    return 0;
}
```

- Notem que quando `N` não é especificado, o compilador utiliza o tamanho 6 por padrão

printf

- O valor de `N` varia de acordo com `X`
- Para floats ele especifica a quantidade de dígitos após o `.`
- Para inteiros ele especifica a quantidade de dígitos que são exibidos. Se o número for menor que `N`, zeros são adicionados à esquerda

```c
#include <stdio.h>

int main(void) {
    printf("%.5d\n", 3);

    return 0;
}
```

printf

- O especificador de conversão `%e` exibe um float em notação científica
- `N` vai indicar a quantidade de dígitos após a vírgula que serão exibidos

```c
#include <stdio.h>

int main(void) {
    printf("%e\n", 0.00315);
    printf("%e\n", 123.0);
    printf("%.2e\n", 0.00315);
    printf("%.2e\n", 123.0);

    return 0;
}
```

printf

- O especificador de conversão `g` exibe um float no formato padrão ou como notação científica de acordo com o seu tamanho
- Na maioria dos casos temos a exibição padrão
- Para números muito grandes ou muito pequenos, ele deve ser exibido como notação científica
- `N` indica quantos dígitos significantes devem ser usados

printf

```c
#include <stdio.h>

int main(void) {
    printf("%g\n", 123.0);
    printf("%g\n", 0.0035);
    printf("%g\n", 19872391287391273.0);
    printf("%g\n", 0.00000000000009123);
    printf("%.3g\n", 19872391287391273.0);
    printf("%.3g\n", 0.00000000000009123);

    return 0;
}
```

Escape

- O caractere `\n` que usamos é uma sequência de escape
- Com os escapes, é possível criar strings com caracteres que podiam causar problemas para o compilador
    - Por exemplo, a quebra de linha
- Outras sequências de escape:
    - `\t` - tab
    - `\b` - backspace
    - `\"` - aspas

scanf

- Usamos o `scanf` para ler dados
- Nele, usamos um string semelhante ao do `printf`, mas que especifica como os valores devem ser lidos
- As conversões permitidas são essencialmente as mesmas do `printf`

scanf

```c
#include <stdio.h>

int main(void) {
    int i, j;
    float x, y;

    scanf("%d%d%f%f", &i, &j, &x, &y);
    printf("i=%d j=%d x=%f y=%f\n", i, j, x, y);

    return 0;
}}
```

- Omitir o `&` pode gerar um *warning*, mas não um erro de compilação
- Entretanto, sem ele, o seu programa deve apresentar erros inesperados

scanf

- Para efetuar a leitura, o `scanf` processa a informação do string da esquerda para a direita
- Para cada conversão, o `scanf` lê a entrada parando quando encontra um caractere que não pertence àquela conversão

scanf

- Por exemplo: `scanf("%d%d", &i, &j);`
- O usuário pode digitar um número: 123
- Ao digitar enter, temos uma quebra de linha (caractere \n) que não faz parte do padrão de um número inteiro
- Então, o `scanf` para de ler a primeira conversão e, nesse caso, atribui um valor à variável `i`
- Ao continuar digitando números, o `scanf` atribuirá o novo valor à variável `j`

scanf

- Não precisamos usar a quebra de linha para iniciar a leitura de outro valor
  - Um espaço funciona também
- Espaços, quebras de linha e tabs no início de um número são ignorados
  - Então, se você digitar ••••10 (• representa um espaço), o `scanf` vai funcionar normalmente

scanf

- Ignorar espaços, quebras de linha e tabs é crucial para entendermos mais a fundo o funcionamento do `scanf`
- Quando o `scanf` está lendo um número e o usuário pressiona, por exemplo, enter, o caractere da quebra de linha não é jogado fora
    - O caractere é passado para próxima leitura
    - Mas, como ele é ignorado no início de um número, então parece que ele parou a leitura e foi ignorado

scanf

- Vamos fazer um teste:

```c
#include <stdio.h>

int main(void) {
    int i, j;

    scanf("%d%d", &i, &j);
    printf("i=%d j=%d\n", i, j);

    return 0;
}
```

- Tente separar os valores numéricos com um caractere diferente, por exemplo, X
    - 10X20

scanf

- Note que o primeiro valor foi lido corretamente, mas o segundo não
  - O primeiro valor foi lido como 10
  - O segundo valor foi lido como X20
- Quando um item não é lido corretamente, o `scanf` termina sua execução e o resultado da leitura não é o esperado

scanf

- Outro teste que podemos fazer para esse programa é entrar com os seguintes caracteres:
  - 10-20
- Nesse caso, o primeiro valor é lido como 10
  - O `scanf` passa para próxima conversão, pois não existe o número `10-`
- O `-` termina a leitura da primeira conversão, mas inicia a próxima, por isso, o segundo valor é lido como `-20`

scanf

- Para um inteiro, o `scanf`:
  - Procura por um digito, um sinal de + ou de -
  - Em seguida, ele lê até encontrar um caractere que não é um dígito

scanf

- Para um float, o `scanf`:
  - Procura por um sinal de `+` ou `-` (opcional)
  - Em seguida, uma série de dígitos, podendo conter um ponto ou um expoente (ambos opcionais)

scanf

- O que acontece se colocarmos outros caracteres no string do `scanf`?

```c
#include <stdio.h>

int main(void) {
    int i, j;

    scanf("%dX%d", &i, &j);
    printf("i=%d j=%d\n", i, j);

    return 0;
}
```

- Nesse caso, o `scanf` espera que após o primeiro inteiro, o usuário digite o `X`
- Se ele o fizer, o valor é descartado e é iniciada a leitura do segundo inteiro
- Se não fizer, o `scanf` aborta sua execução

scanf

- Espaços à esquerda dos inteiros são ignorados
- Espaços à esquerda dos caracteres não são
  - 10X 20 é valido
  - 10 X20 é inválido

Exemplo

- Vamos fazer um programa que recebe uma data e converte ela pro formato americano (ano/mes/dia)

Exemplo

```c
#include <stdio.h>

int main(void) {
    int dia, mes, ano;

    scanf("%d/%d/%d", &dia, &mes, &ano);
    printf("%d/%d/%d\n", ano, mes, dia);

    return 0;
}
```