AULA 01

Simone Aires, UTFPR-PG

2

Revisão e Erros frequentes

Profa. Simone Aires - Prof. Saulo Queiroz

Revisão Linguagem C

- Disponível no Moodle
  - Aula 1 – Revisão
- Qualquer dúvida poderá recorrer a este material ou Google ☺
  - Infelizmente neste disciplina não temos tempo para rever tudo... Mas o material no Moodle está bem completo.

Erros frequentes

- ✓ Comparar duas expressões com = em vez de == exemplo:

```
if(i=1)
        printf("Erro!\n");
```

Correto:

```
if(i==1)
        printf("Erro!\n");
```

Erros frequentes

- Colocar ";" depois da condição de um if ou for;

```
if(i>=1);
   i=0;
```


Errado: o compilador entende que não há instrução nenhuma mesmo o if sendo verdadeiro! Isso pq ele encontra o ; que representa final de linha de instrução!

Erros frequentes

- Colocar ";" depois da condição de um if ou for;

```
if(i>=1);
   i=0;
```

Errado: o compilador entende que não há instrução nenhuma mesmo o if sendo verdadeiro! Isso pq ele encontra o ; que representa final de linha de instrução!

```
while(k>0);
{
   k--;
   printf("*");
}
```

Erros frequentes

- Colocar ";" depois da condição de um if ou for;

```
if(i>=1);
   i=0;

while(k>0);
{
   k--;
   printf("*");
}
```

Errado: o compilador entende que não há instrução nenhuma mesmo o if sendo verdadeiro! Isso pq ele encontra o ; que representa final de linha de instrução!

Erros frequentes

- Colocar "," em vez de ";" como separador das expressões do for.

```
for(i=0,i<n,i+)...;
```

- Usar "fflush(stdin)" a seguir a cada "printf“ posterior a scanf:

```
...printf("Aqui");
  fflush(stdin);...*...
```

Erros frequentes

- Colocar "," em vez de ";" como separador das expressões do for.

  for(i=0,i<n,i+)...;

- % somente para inteiros

Você limpa o buffer do stdin (padrão de entrada – teclado) antes de ler um dado. Ou seja, antes do scanf()

10

Strings: Cadeia de Caracteres, Vetor de char

Profa. Simone Aires - Prof. Saulo Queiroz

Tipo caracter (char) e cadeia (*string*)

- Vimos que variáveis do tipo char comportam no máximo 1 caracter
- Como armazenar mais de um caracter em uma variável?
  -
  -

Tipo caracter (char) e cadeia (*string*)

- Vimos que variáveis do tipo char comportam no máximo 1 caracter
- Como armazenar mais de um caracter em uma variável?
  - Em C não existe o tipo cadeia (*string*)
  - Para termos variáveis que guardam muitas letras usamos um **vetor de char**!

Strings em C: Vetores de char

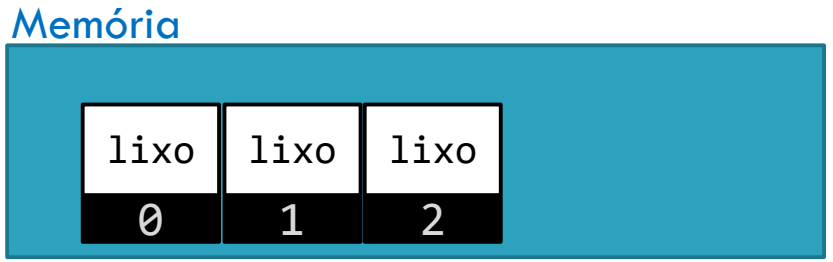
- Vetores de char tem uma peculiaridade: sua última posição deve ser finalizada com um caracter especial chamado nulo, ou, '\0'
  - É assim que sabemos que a *string* terminou
- Conclusão: se pretende armazenar N caracteres, então você precisa de um vetor com N+1 posições devido o '\0'

Organização de vetores na memória


```
main()
{




}
```

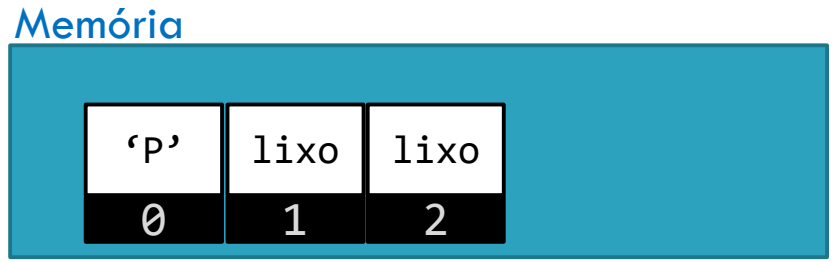
Organização de vetores na memória


```
main()
{
   char palavrinha[3];



}
```



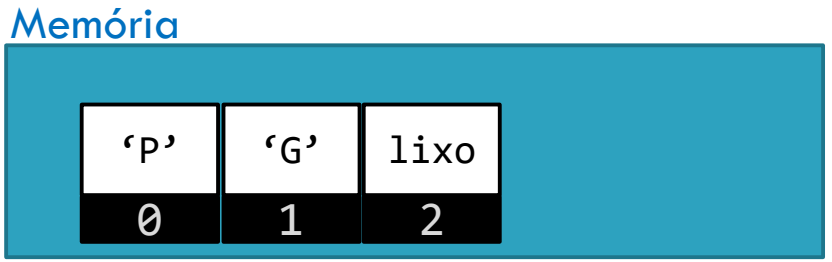
Organização de vetores na memória


```
main()
{
   char palavrinha[3];
   palavrinha[0] = 'P';


}
```



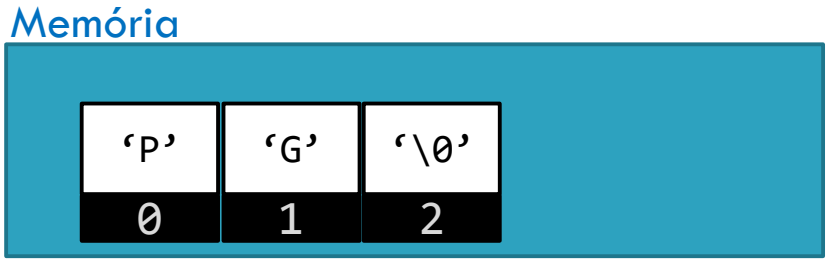
Organização de vetores na memória


```
main()
{
   char palavrinha[3];
   palavrinha[0] = 'P';
   palavrinha[1] = 'G';

}
```



Organização de vetores na memória


```
main()
{
  char palavrinha[3];
  palavrinha[0] = ‘P’;
  palavrinha[1] = ‘G’;
  palavrinha[2] = ‘\0’;
}
```



String: Inicialização

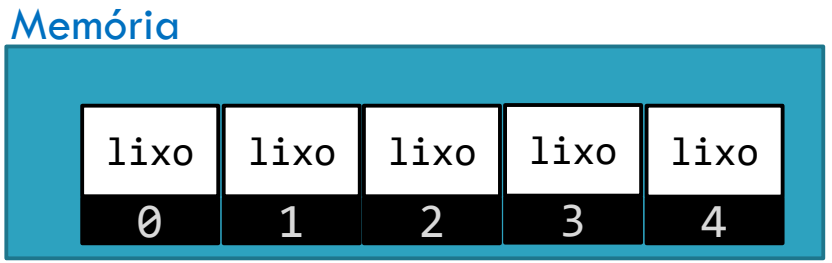
- Podemos inicializar da mesma forma que outros tipos de vetores.
  - `char nome[]={‘F’, ‘u’, ‘l’, ‘a’, ‘n’, ‘o’, ‘\0’}`
- Forma mais cômoda:
  - `char nome[7]=“Fulano”;`

Strings: Entrada e Saída padrão

- Tanto para ler como para escrever usamos o código %s para strings
  - Com esse código o scanf coloca o '\0' automaticamente no fim da variável *string*

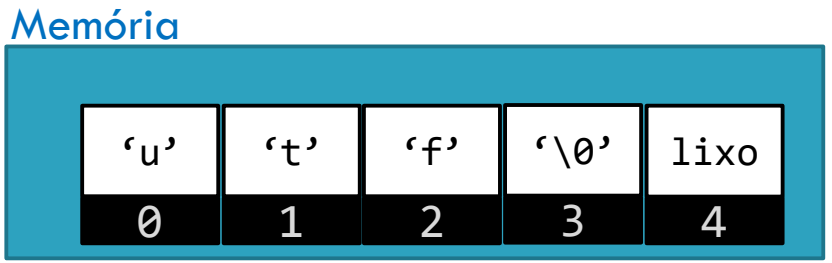
Organização de vetores na memória


```
#include <stdio.h>
main()
{
  char nome[5];
  printf("Digite seu nome");
  scanf("%s",nome);
  printf("Seu nome é %s", nome);
}
```



Organização de vetores na memória


```
#include <stdio.h>
main()
{
  char nome[5];
  printf("Digite seu nome");
  scanf("%s",nome);//string dada: utf
  printf("Seu nome é %s", nome);
}
```



Strings: Peculiaridades

- Por que usamos & para vetores de inteiros e não usamos para vetor e char (*strings*)?
  -
  -
    -
    -
  -
    -
    -

Strings: Peculiaridades

- Por que usamos & para vetores de inteiros e não usamos para vetor e char (*strings*)?
  - O identificador do vetor é sempre um endereço de memória (exatamente o que o scanf precisa)
  - Para vetores de números, geralmente o usuário fornece um dado por posição.
    - Ex.: vetor de notas da turma: primeiro a nota 1 depois a nota 2, não ambas de vez.
    - `scanf("%f", &nota[0]); scanf("%f", &nota[1]);`
  -
    -
    -

Strings: Peculiaridades

- Por que usamos & para vetores de inteiros e não usamos para vetor e `char` (*strings*)?
  - O identificador do vetor é sempre um endereço de memória (exatamente o que o `scanf` precisa)
  - Para vetores de números, geralmente o usuário fornece um dado por posição.
    - Ex.: vetor de notas da turma: primeiro a nota 1 depois a nota 2, não ambas de vez.
    - `scanf("%f", &nota[0]); scanf("%f", &nota[1]);`
  - Para vetores de `char`, informamos todas as posições de uma vez!
    - `scanf("%s", nomeAluno);`
    - É desnecessário (mas não errado) fazer: `scanf("%s",&nomeAluno);`

Strings e scanf

- O scanf apresenta algumas limitações para leitura de *strings*.Ex.: Tente *strings* com espaços em branco
- Alternativas para ler string. Ex `char nome[100];`
  - `scanf("%[^\n]", nome);`
  - `fgets(nome, 100, stdin);`
    - `stdin` representa dados vindos da entrada padrão
    - Pra *strings*, prefira o `fgets`. Pesquise por que.

Aritmética com endereços

- ✓ char sobrenome[]
- ✓ sobrenome é o endereço de uma variável do tipo char
- ✓ sobrenome + 1? Isso vai depender do tipo da variável...
  - ⊙ 1 byte → char
  - ⊙ 4 bytes → int/float
- ✓ se M é o nome de uma matriz e i é uma variável do tipo int, então
  - ⊙ M + i é igual a M[i]

Exemplo

```
/* Mostra o uso da aritmética com endereços */
int main()
{
        char salute[]="Ola ";
        char nome[80];
        printf("Digite o seu nome: ");
        fgets(nome,80,stdin);
        printf("%s \n", nome + 2);
        printf("%s %s\n", salute, nome + 5);
        return 0;
}
```

Tratamento de Strings

- #include <string.h>

- Principais funções para manipulação de strings:
  - strcmp (s1, s2): comparação de strings
  - strlen(s1): devolve o tamanho da string
  - strcpy(para, de): copia string
  - strcat(str1,str2): concatena duas strings
  - strupr(str): coloca *str* em letras maiúsculas
  - strlwr(str): coloca *str* em letras minúsculas

Para saber o tamanho de uma string

- `int size = strlen(str);`
  - Retorna um valor inteiro com o número de caracteres da string (desprezando o "\0").

```
main(){
    char re[80];
   int size;
   printf ("Digite a palavra: ");
    scanf ("%s", &re); //  fgets(re,80,stdin);
   size=strlen(re);
   printf ("Esta palavra tem %d caracteres.\n", size);
}
```

Exemplo 2

```
int main()
{   int len, i;
    char nome[80];
    printf("Digite o seu nome: ");
    fgets(nome,80,stdin);
    len=strlen(nome);
    for(i=0; i<len + 4; i++){
        printf("Endereço = %x\tChar  = %c\t Inteiro = %d\n", (nome+i), nome[i], (int)nome[i]);

}
return 0; }
```

Para concatenar duas strings

- ✓ strcat(str1,str2);
  - Esta função é utilizada para concatenar (unir / juntar) duas strings. str2 será adicionada no final de str1

```
int main()
{  char salute[100]="Ola ";
   char nome[80];
   printf("Digite o seu nome: ");
   fgets(nome,80,stdin);
   strcat(salute, nome);
   printf("%s\n", salute);
   return 0;          }
```

Para comparar duas strings

- strcmp (s1, s2);
- Compara duas strings:
  - <0 str1 é menor que str2 (ordem alfabética)
  - 0 str1 é igual a str2 (diferencia maiúsculas de minúsculas)
  - >0 str1 é maior que str2

Para comparar duas strings – Exemplo1

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
main( ){
    char re[80];
    printf ("Digite a senha: ");
    scanf ("%s", &re);
    if (strcmp(re,"laranja")==0) {
        printf ("Senha correta\n");
    }
    else printf ("Senha invalida\n");
 }
```

Para comparar duas strings – Exemplo2

```
int main()
{    char resposta[]="BRANCO";
     char resp[40];
     printf("Qual é a cor do cavalo branco de Napoleão?");
     fgets(resp,40,stdin);
     while (strcmp(resp,resposta) != 0){
            printf("Resposta errada. Tente de novo.\n");
            gets(resp);}
   printf("Correto!\n");
   return 0;          }
```

Comparar duas strings – Exemplo3

```
int main()
{
        printf("%d\n", strcmp("A", "A"));
        printf("%d\n", strcmp("A", "B"));
        printf("%d\n", strcmp("B", "A"));
        printf("%d\n", strcmp("C", "A"));
        printf("%d\n", strcmp("casas", "casa"));
        return 0;
}
```

Copiar o conteúdo de uma string para outra

- strcpy(str1, str2);
  - copia o conteúdo de str2 para str1 (str1 não pode ser menor que str2)

```
main()
{
  char str[80];
  strcpy (str, "Alo");
  printf ("%s\n", str);
}
```

Exercício

- Implemente um programa que possui as função:
  - Comparar duas *strings* em C
  - Concaternar duas *strings* em C
  - Calcular o tamanho de uma *string* em C
- Cada função deve receber uma strings de entrada e fornecer um valor de saída.
- Leia as entradas e imprima o resultado na tela.
  - Defina as suposições necessárias a cada código em forma de comentário

Matriz de *strings*

- Podemos usar matrizes para armazenar `l` nomes (linhas) cada um com no máximo `c` caracteres (colunas)
- Exemplo: 40 alunos cujos nomes tem, no máximo 100 caracteres.
  - `char alunos[40][100];`
- Como fazer a leitura?

Matriz de *strings*: Leitura (trecho)


```
// na posição 0 da matriz há um vetor de string
  fgets(alunos[0], 100, stdin);

// na posição 1 da matriz há um vetor de string
 fgets(alunos[1], 100, stdin);
...
// na posição 39 da matriz há um vetor de string
 fgets(alunos[39], 100, stdin);
```

Matriz de *strings*: Leitura (trecho)


```
// na posição 0 da matriz há um vetor de string
  fgets(alunos[0], 100, stdin);

// na posição 1 da matriz há um vetor de string
 fgets(alunos[1], 100, stdin);
...
// na posição 39 da matriz há um vetor de string
 fgets(alunos[39], 100, stdin);
```

Matriz de *strings*: Leitura (trecho)


```
int i;
for (i=0; i<40; i++)
{
  printf("Informe o nome do aluno %i\n", i+1);
  fgets(alunos[i], 100, stdin);
  //scanf("%[^\n]", alunos[i]);
 }
```

Matriz de *strings*: Impressão (trecho)


```
int i;
for (i=0; i<40; i++)
 printf("Nome %i é %s\n", i, nome[i]);
```

Exercícios

- Faça um algoritmo para ajudar a dar nomes às criancinhas recém-nascidas da Polônia. Leia a quantidade M de crianças e, para cada criança, leia a quantidade N_i de caracteres que o nome dela deve ter. Force as seguintes condições $M>0$ e $3 \leq N_i \leq 10$ para todo $0 \leq i < 40$. Após isso, gere aleatoriamente as N_i letras do nome da i-ésima criança. A primeira letra deve ser maiúscula e as demais minúsculas. Imprima o nome de cada criança!

Dicas para o exercício

- Gerar números aleatórios em um dado intervalo:
  - `rand() % (maximo + 1 - minimo) + minimo`

Pós-aula

- Lista de exercícios – Moodle.