MONITORIA DE RLA

Estruturas de repetição, Arrays de caracteres(strings)

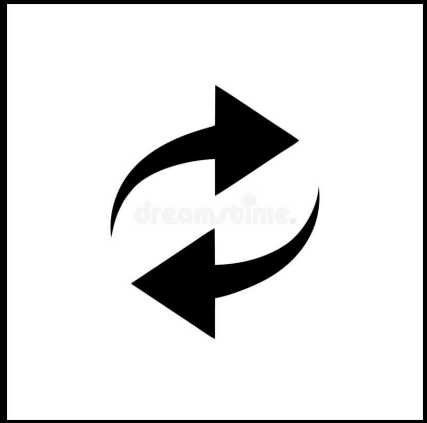
Comandos de repetição:

Na seção anterior, vimos como realizar desvios condicionais em um programa. Desse modo, criamos programas em que um bloco de comandos é executado somente se determinada condição for verdadeira. **Entretanto, há casos em que é preciso que um bloco de comandos seja executado mais de uma vez se determinada condição for verdadeira:**

Para isso, precisamos de uma estrutura de repetição que permita executar um conjunto de comandos quantas vezes forem necessárias.

Por condição entende-se qualquer expressão relacional (ou seja, que use os operadores >, <, >=, <=, == ou !=) que resulte em uma resposta do tipo verdadeiro ou falso. A condição pode ainda ser uma expressão que utiliza operadores:

- Matemáticos : +, —, *, /, %
- Relacionais: >, <, >=, <=, ==, !=
- Lógicos: &&, ||



Importante

Laço infinito Um laço infinito (ou loop infinito) é uma sequência de comandos em um programa de computador que sempre se repete, ou seja, infinitamente. Isso geralmente ocorre por algum erro de programação, quando:

- Não definimos uma condição de parada.
- A condição de parada existe, mas nunca é atingida.

Comando while :

A forma geral de um comando while é:

```
while (condição){
        sequência de comandos;
}
```

Na execução do comando while, a condição será avaliada é:

• Se a condição for considerada verdadeira (ou possuir valor diferente de zero), a sequência de comandos será executada. Ao final da sequência de comandos, o fluxo do programa é desviado novamente para o teste da condição.

• Se a condição for considerada falsa (ou possuir valor igual a zero), a sequência de comandos não será executada.

O comando while segue todas as recomendações definidas para o comando if quanto ao uso das chaves e definição da condição usada.

```
#include
#include

int main(){
        int a,b;
        printf("Digite o valor de a: ");
        scanf("%d",&a);
        printf("Digite o valor de b: ");
        scanf("%d",&b);

        while (a < b){
                a = a + 1;
                printf("%d \n",a);
        }
        system("pause");

        return 0;
}
```

Comando do-while : O comando do-while é bastante semelhante ao comando while visto anteriormente. Sua principal diferença é com relação à avaliação da condição: enquanto o comando while avalia a condição para depois executar uma sequência de comandos, o comando do-while executa uma sequência de comandos para depois testar a condição.
A forma geral de um comando do-while é:

```
do{
        sequência de comandos;
} while(condição);
```

Na execução do comando do-while, é executada esta ordem de passos:

- A sequência de comandos é executada
- A condição é avaliada:
    - º Se a condição for considerada verdadeira (ou possuir valor diferente de zero), o fluxo do programa é desviado novamente para o comando do, de modo que a sequência de comandos seja executada novamente.
    - º Se a condição for considerada falsa (ou possuir valor igual a zero), o laço termina (fim do comando do-while).

```
#include
#include

int main(){
        int i;

        do{
                printf (“Escolha uma opcao:\n”);
                printf (“(1) Opção 1\n”);
                printf (“(2) Opção 2\n”);
                printf (“(3) Opção 3\n”);
                scanf(“%d”, &i);
        } while ((i < 1) || (i > 3));

printf (“Voce escolheu a Opcao %d.\n”,i);
system(“pause”);

return 0;
}
```

Comando for : O comando for é muito similar ao comando while, visto anteriormente. Basicamente, o comando for é usado para repetir um comando ou uma sequência de comandos diversas vezes. A forma geral de um comando for é:

```
for (inicialização; condição; incremento) {
        sequência de comandos;
}
```

Na execução do comando for, é realizada esta sequência de passos:

• A cláusula inicialização é executada: nela as variáveis recebem um valor inicial para usar dentro do for.

• A condição é testada: º Se a condição for considerada verdadeira (ou possuir valor diferente de zero), a sequência de comandos será executada. Ao final da sequência de comandos, o fluxo do programa é desviado para o incremento. º Se a condição for considerada falsa (ou possuir valor igual a zero), a sequência de comandos não será executada (fim do comando for).

• Incremento: Terminada a execução da sequência de comandos, ocorre a etapa de incremento das variáveis usadas no for. Ao final dessa etapa, o fluxo do programa é novamente desviado para a condição.

```
#include
#include
int main(){
        int a,b,c;
        printf(“Digite o valor de a: ”);
        scanf(“%d”,&a);
        printf(“Digite o valor de b: ”);
        scanf(“%d”,&b);

        for (c = a; c <= b; c++){
                printf(“%d \n”,c);
        }

        system(“pause”);

        return 0;
}
```

Omitindo uma cláusula do comando for???

Aninhamento de repetições:
Uma repetição aninhada é simplesmente um comando de repetição utilizado dentro do bloco de comandos de um outro comando de repetições. Basicamente, é um comando de repetição dentro de outro, semelhante ao que é feito com o comando if.
A forma geral de um comando de repetição aninhado é:

```
repetição(condição 1) {
        sequência de comandos;
        repetição(condição 2) {
                sequência de comandos;
        repetição... }
}
```

em que repetição representa um dos três possíveis comandos de repetição da linguagem C: while, for e do-while.

Comando break:
Vimos, anteriormente, que o comando break pode ser utilizado em conjunto com o comando switch. Basicamente, sua função era interromper o comando switch assim que uma das sequências de comandos da cláusula case fosse executada. Caso o comando break não existisse, a sequência de comandos do case seguinte também seria executada, e assim por diante. Na verdade, o comando break serve para quebrar a execução de um conjunto de comandos (como no caso do switch) ou interromper a execução de qualquer comando de repetição (for, while ou do-while).

O comando break faz com que a execução do programa continue na primeira linha seguinte ao laço que está sendo interrompido.

```
int a=0;
int b=10;
while (a <= b){
        a = a + 1;
        if(a == 5) break;
        printf("%d \n",a);
}
```

Comando continue:
O comando continue é muito parecido com o comando break. Tanto o break quanto o continue ignoram o restante da sequência de comandos da repetição que os sucedem. A diferença é que, enquanto o comando break termina o comando de repetição que está sendo executado, o comando break interrompe apenas aquela repetição e passa para a próxima repetição do laço, se ela existir. Por esse mesmo motivo, o comando continue só pode ser utilizado dentro de um laço, diferentemente do comando break, que pode ser usado em laços e no comando switch.

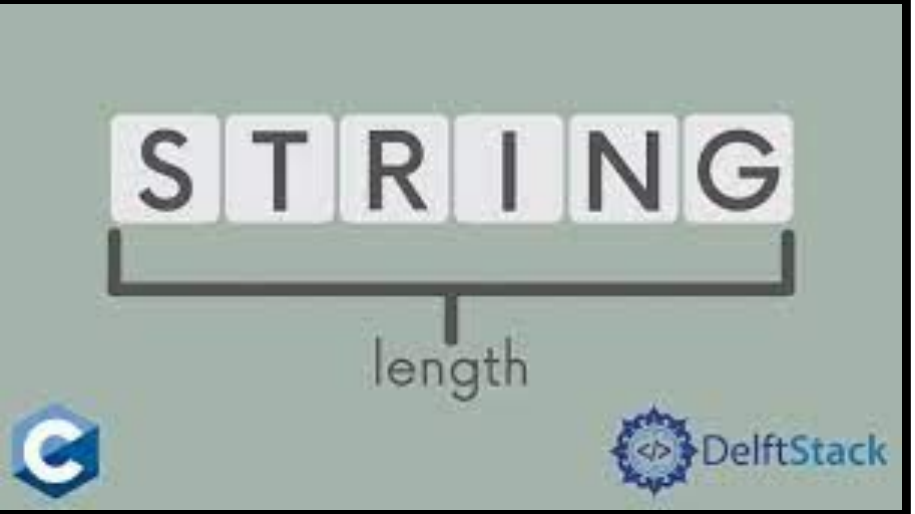
Quando o comando “continue” é executado, os comandos restantes da repetição são ignorados. O programa volta a testar a condição do laço para saber se ele deve ser executado novamente ou não.

```
while (a <= b){
        a = a + 1;
         if(a == 5) continue;
        printf(“%d \n”,a);
}
```

Comando goto:
O comando goto é um salto condicional para um local especificado por uma palavra-chave no código. A forma geral de um comando goto é: destino: goto destino; Nessa sintaxe, o comando goto (do inglês go to, literalmente, “vá para”), muda o fluxo do programa para um local previamente especificado pela expressão destino, em que destino é uma palavra definida pelo programador. Esse local pode ser à frente ou atrás no programa, mas deve ser dentro da mesma função.

```
int i = 0;
início:
if(i < 5){
        printf(“Numero %d\n”,i);
         i++;
         goto inicio;
}
```

Arrays de caracteres(strings)



Definição e declaração de uma string:
String é o nome que usamos para definir uma sequência de caracteres adjacentes na memória do computador. Essa sequência de caracteres, que pode ser uma palavra ou frase, é armazenada na memória do computador na forma de um array do tipo char.

Por ser a string um array de caracteres, sua declaração segue as mesmas regras da declaração de um array convencional:
`char str[6];`

Essa declaração cria na memória do computador uma string (array de caracteres) de nome str e tamanho igual a 6.
No entanto, apesar de ser um array, devemos ficar atentos para o fato de que as strings têm no elemento seguinte a última letra da palavra/ frase armazenada, um caractere "\0".
O caractere "\0" indica o fim da sequência de caracteres.

Inicializando uma string: Uma string pode ser lida do teclado ou já ser definida com um valor inicial. Para sua inicialização, pode-se usar o mesmo princípio definido na inicialização de vetores e matrizes:

```
char str [10] = { 'J', 'o', 'a', 'o', '\0' };
```

Percebe-se que essa forma de inicialização não é muito prática. Por isso, a inicialização de strings também pode ser feita por meio de aspas duplas:

```
char str [10] = "Joao";
```

Essa forma de inicialização possui a vantagem de já inserir o caractere '\0' no final da string.

Acessando um elemento da string: Outro ponto importante na manipulação de strings é que, por se tratar de um array, cada caractere pode ser acessado individualmente por indexação como em qualquer outro vetor ou matriz:

```
char str[6] = "Teste";
str[0] = 'L';
```

Trabalhando com strings: O primeiro cuidado que temos que tomar ao trabalhar com strings é na operação de atribuição.

```
char str1[20] = "Hello World";
char str2[20];
 str1 = str2; //ERRADO!
```

Isso ocorre porque uma string é um array, e a linguagem C não suporta a atribuição de um array para outro. Para atribuir o conteúdo de uma string a outra, o correto é copiar a string, elemento por elemento, para a outra string

```
for (i = 0; str1[i]!='\0'; i++) {
        str2[i] = str1[i];
}
```

Usando a função scanf()
Existem várias maneiras de se fazer a leitura de uma sequência de caracteres do teclado. Uma delas é utilizando a já conhecida função scanf() com o formato de dados "%s":

```
char str[20];
scanf("%s",str);
```

Quando usamos a função scanf() para ler uma string, o símbolo de & antes do nome da variável não é utilizado. Os colchetes também não são utilizados, pois queremos ler a string toda e não apenas uma letra.

Infelizmente, para muitos casos, a função scanf() não é a melhor opção para se ler uma string do teclado. A função scanf() lê apenas strings digitadas sem espaços, ou seja, palavras.
No caso de ter sido digitada uma frase (uma sequência de caracteres contendo espaços), apenas os caracteres digitados antes do primeiro espaço encontrado serão armazenados na string se a sua leitura for feita com a função scanf().

Usando a função gets() Uma alternativa mais eficiente para a leitura de uma string é a função gets(), a qual faz a leitura do teclado considerando todos os caracteres digitados (incluindo os espaços) até encontrar uma tecla enter:

```
char str[20];
gets(str);
```

Usando a função fgets(): Basicamente, para ler uma string do teclado utilizamos a função gets(). No entanto, existe outra função que, utilizada de forma adequada, também permite a leitura de strings do teclado. Essa função é a fgets(), cujo protótipo é:

```
char *fgets (char *str, int tamanho, FILE *fp);
```

A função fgets() recebe três parâmetros de entrada:
- str: a string a ser lida.
- tamanho: o limite máximo de caracteres a serem lidos.
- fp: a variável que está associada ao arquivo de onde a string será lida.(stdin)

Limpando o buffer do teclado: Às vezes, podem ocorrer erros durante a leitura de caracteres ou strings do teclado. Para resolver esses pequenos erros, podemos limpar o buffer do teclado (entrada padrão) usando a função setbuf(stdin, NULL) antes de realizar a leitura de caracteres ou strings (Figura 7.2).

Basicamente, a função setbuf() preenche um buffer (primeiro parâmetro) com determinado valor (segundo parâmetro). No exemplo anterior, o buffer da entrada padrão (stdin), ou seja, o teclado, é preenchido com o valor vazio (NULL). Na linguagem C, a palavra NULL é uma constante-padrão que significa um valor nulo. Um buffer preenchido com NULL é considerado limpo/vazio.

Exemplo: limpando o buffer do teclado

	Leitura de caracteres	Leitura de strings
01	`char ch;`	`char str[10];`
02	`setbuf(stdin, NULL);`	`setbuf(stdin, NULL);`
03	`scanf("%c", &ch);`	`gets(str);`

Escrevendo uma string na tela: Usando a função fputs():
Existe uma outra função que, se utilizada de forma adequada, também permite a escrita de strings na tela. Essa função é a fputs(), cujo protótipo é:

```
int fputs (char *str,FILE *fp);
```

A função fputs() recebe dois parâmetros de entrada:

- str: a string (array de caracteres) a ser escrita na tela.

- fp: a variável que está associada ao arquivo onde a string será escrita(stdout).

PROJETO

Prática: https://replit.com/@GuilhermeAlme18/Aprendendo-C

Projeto: https://github.com/Guilfullstack/Task_manager.git

Implementação

Hora de implementar:

Com base no que foi ensinado, vamos incluir no projeto:

- Animação de entrada e carregamento;
- Estrutura de repetição para os menus com condições que sigam uma lógica;
- Aperfeiçoar a verificação de login.
- Aperfeiçoar o menu principal.

Strings parte 2

- **Funções para manipulação de strings:**
  A biblioteca-padrão da linguagem C possui funções especialmente desenvolvidas para a manipulação de strings na biblioteca . A seguir são apresentadas algumas das mais utilizadas.

Tamanho de uma string:
Para obter o tamanho de uma string, usa-se a função strlen():

```
char str[15] = "teste";
printf("%d",strlen(str));
```

Nesse caso, a função retornará 5, que é o número de caracteres na palavra "teste", e não 15, que é o tamanho do array de caracteres.

A função strlen() retorna o número de caracteres que existem antes do caractere '\0', e não o tamanho do array onde a string está armazenada.

Copiando uma string: Vimos que uma string é um array e que a linguagem C não suporta a atribuição de um array para outro. Nesse sentido, a única maneira de atribuir o conteúdo de uma string a outra é a cópia, elemento por elemento, de uma string para outra. A linguagem C possui uma função que realiza essa tarefa para nós: a função strcpy():

```
strcpy(char *destino, char *origem)
```

Para evitar estouro de buffer, o tamanho do array destino deve ser longo o suficiente para conter a sequência de caracteres contida na origem.

Concatenando strings:
A operação de concatenação é outra tarefa bastante comum ao se trabalhar com strings. Basicamente, essa operação consiste em copiar uma string para o final de outra string. Na linguagem C, para fazer a concatenação de duas strings, usa-se a função strcat():
strcat(char *destino, char *origem)
Basicamente, a função strcat() copia a sequência de caracteres contida em origem para o final da string destino. O primeiro caractere da string contida em origem é colocado no lugar do caractere "\0" da string destino

```
char str1[15] = "bom ";
char str2[15] = "dia";
strcat(str1,str2);
printf("%s",str1);
system("pause");
```

Comparando duas strings:
Da mesma maneira como o operador de atribuição não funciona para strings, o mesmo ocorre com operadores relacionais usados para comparar duas strings.
Desse modo, para saber se duas strings são iguais usa-se a função strcmp():

```
int strcmp(char *str1, char *str2)
```

A função strcmp() compara posição a posição as duas strings (str1 e str2) e retorna um valor inteiro igual a zero no caso de as duas strings forem iguais. Um valor de retorno diferente de zero significa que as strings são diferentes