INSTITUTO FEDERAL
Paraná
Campus Paranaguá

Ministério da Educação

ENSINO MÉDIO INTEGRADO - INFORMÁTICA
Disciplina de Linguagem de Programação
Aula 11: Ponteiros e Biblioteca <string.h>

Gil Eduardo de Andrade

Conceitos Preliminares

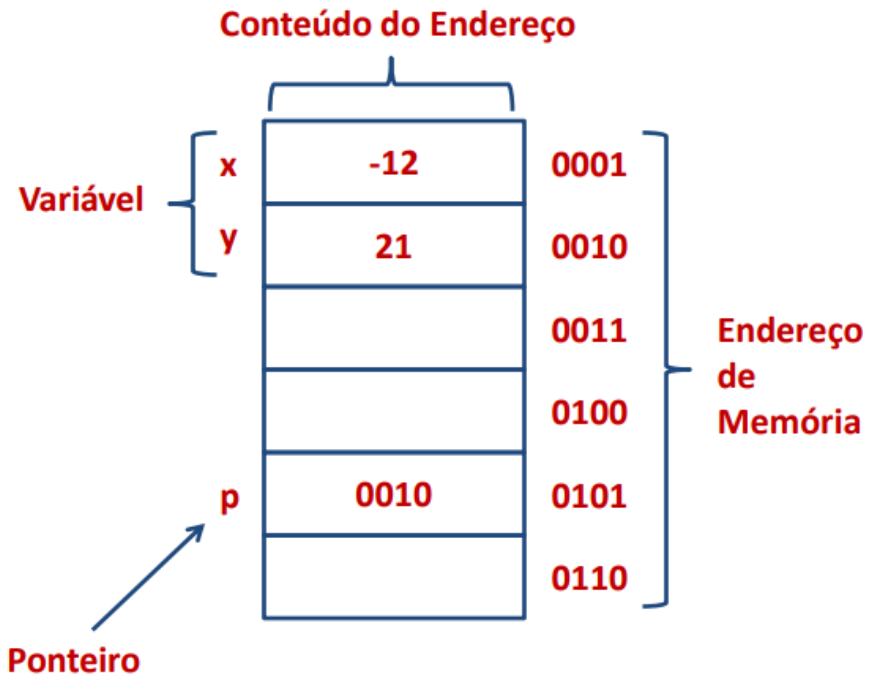
(https://docs.microsoft.com/pt-br/cpp/c-language/?view=msvc-170)

Introdução

O ponteiro é uma variável que contém um endereço de memória, normalmente relativo à localização de uma outra variável em memória. Sendo assim, chamamos uma variável de ponteiro porque ela aponta para o endereço de memória onde encontra-se outra variável.

Variáveis: Regular x Ponteiro

Uma variável regular faz referência direta para um valor armazenado em memória. Uma variável ponteiro contém o endereço de memória de outra variável que faz referência direta ou não para um valor armazenado em memória.



Observe na imagem anterior que o ponteiro **"p"** armazena o endereço de memória (**0010**) da variável **"y"**, que possui como conteúdo o valor **21**. Ou seja, ponteiro **"p"** não armazena o valor **21**, mas sim o endereço de memória da variável **"y"**.

Utilização e Benefícios

O conceito de ponteiro permite que alocações dinâmicas de memória sejam efetuadas. Através da alocação dinâmica de memória torna-se possível especificar o tamanho de um vetor ou de uma matriz durante a execução do programa. Além disso, o conceito de ponteiro permite que funções modifiquem o conteúdo de variáveis recebidas como argumento. Ademais, o conceito de ponteiro possibilita aumenta a eficiência de determinadas rotinas.

A declaração de um ponteiro é efetuada da seguinte maneira:

int *nome_ponteiro;

- **tipo:** indica qual é o tipo da variável que será apontada pelo ponteiro. Exemplo: ***int***, ***float***, ***double***;
- **nome_poteiro:** indica o nome pelo qual o ponteiro é referenciado;

Operadores

Existem dois operadores especiais para ponteiros:

- **"&"**: retorna o endereço de memória
  - Ex.: int *p = &a; *("p" recebe o endereço de memória de "a")*

- **"*"**: retorna o conteúdo (valor) da variável;
  - Ex.: int b = *p; *("b" recebe o conteúdo do endereço de memória apontado por "p")*

Alocação de Memória

A alocação dinâmica é o meio pelo qual um programa pode obter memória durante a sua execução. Esse recurso mostra-se útil porque as variáveis locais e globais não podem obter mais memória durante o tempo de execução. Sendo

assim, para obter maior quantidade de memória em tempo de execução utilizamos os conceitos de ponteiros e alocação dinâmica de memória.
A linguagem C possui diversas funções de alocação dinâmica, contudo suas principais são a função ***malloc()*** e a função ***free()***.

- ***malloc()***: aloca a quantidade de memória necessária;
- ***free()***: libera a quantidade de memória alocada;

Função *malloc()*

```
void* malloc(size_t * numero_bytes);
```

- **void***: retorna um ponteiro para qualquer tipo, ou seja, pode ser utilizada para qualquer tipo de ponteiro (int *; char *);
- **size_t**: definido na biblioteca **<stdlib.h>**, uma espécie de ponteiro sem sinal;
- **numero_bytes**: total de bytes que se deseja alocar;

Exemplo: `int* p = malloc(10 * sizeof(int));`

No exemplo anterior é alocado um espaço de memória para 10 valores inteiros. A função **sizeof(int)** retorna o tamanho de um inteiro na memória, como esse tamanho é multiplicado por 10, permite alocar espaço para 10 valores inteiros.
Observe que cada tipo possui um tamanho específico, por exemplo, um **double** utiliza um espaço de memória maior que um **int**;

Realocação de Memória

A realocação dinâmica é o meio pelo qual um programa pode obter memória mais de uma vez para um mesmo ponteiro, durante a sua execução. Na linguagem C, a função de realocação dinâmica, que permite efetuar tal processo é a **realloc()**.

Exemplo 1: `char* p = realloc(NULL, 12 * sizeof(char));`
Exemplo 2: `char* p = realloc(p, 12 * sizeof(char));`

Realoca o tamanho do ponteiro **"p"** para 12, o **sizeof(char)** garante que seja alocado 12 vezes o tamanho de um caractere em memória.

Observe que diferentemente da função ***malloc()*** a função ***realloc()*** possui um parâmetro a mais (**void***) que indica se o conteúdo atual de **"p"** deve ser mantido após a realocação ou não.

No primeiro exemplo é efetuada a realocação de memória para o ponteiro **"p"**, **SEM MANUTENÇÃO** do seu conteúdo;

No segundo exemplo é efetuada a realocação de memória para o ponteiro **"p"**, **COM MANUTENÇÃO** do seu conteúdo;

Biblioteca <string.h>

A biblioteca <string.h> disponibiliza funções para manipulação de Strings (conjunto de caracteres) em C. Dentre as funções disponibilizadas temos como recurso a possibilidade de obter o tamanho de uma string, efetuar comparações, cópias e concatenações (juntar).

Função: ***int strlen(char* str)***
Calcula o comprimento (quantidade de caracteres) para uma determinada string.

- **int:** valor inteiro de ***retorno*** que indica o total de caracteres calculado para string **str**.
- **char***: string, passada a função, da qual deseja-se descobrir o comprimento;

Função: ***int strcmp(char* str1, char* str2)***
Efetua a comparação entre duas strings.

- **int**: valor inteiro de ***retorno*** que indica qual foi o resultado da comparação entre ***str1*** e ***str2***. Para retorno:
  - **= 0**: as strings ***str1*** e ***str2*** são **iguais**;
  - **> 0**: a string ***str1*** é lexicograficamente ***maior que*** a string ***str2***;
  - **< 0**: a string ***str1*** é lexicograficamente ***menor que*** a string ***str2***;
- **char***: strings passadas a função para serem comparadas;

Função: ***strcpy(char* destino, char* origem)***

Copia uma string de ***origem*** para outra string de ***destino***

- **char* destino**: string que receberá o conjunto de caracteres a ser copiado da string de origem.
- **char* origem**: string que contém os caracteres que serão copiados para a string de destino.

Função: ***strcat(char* destino, char* origem)***

Concatena (junta) uma string de ***origem*** com uma string de ***destino***

- **char* destino**: string que juntará ao seu conteúdo o conjunto de caracteres copiados da string de origem;
- **char* origem**: string que contém os caracteres que serão concatenados a string de destino.

Codificação – Linguagem de Programação C

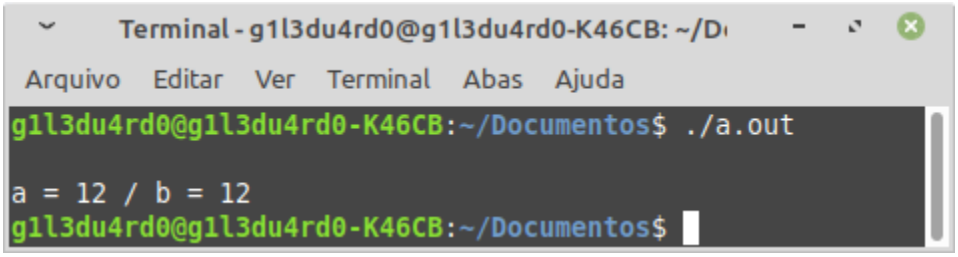
Declarando Ponteiros - Utilizando operadores

```
#include <stdio.h>

#include <stdio_ext.h>

#include <stdlib.h>

int main() {

   // declara duas variáveis inteiras

   int a, b;

   // declara um ponteiro inteiro

   int *p;

   // atribui o valor 12 a variável "a"
```

```
    a = 12;

    // aponta o ponteiro "p" para variável "a"

    // "p" recebe o endereço de memória da variável "a"

    p = &a;

    // atribui a "b" o conteúdo da variável apontada por "p"

    // nesse caso o conteúdo da variável "a"

    b = *p;

    // apresenta os valores armazenados em "a" e "b"

    printf("\na = %i / b = %i", a, b);

    printf("\n");

    return 0;

}
```

Declarando Ponteiros - Utilizando Operadores.

(Arquivo-fonte: 12 - Ponteiros/ponteiro_operadores.c)



Resultado da execução do código anterior

Declarando Ponteiros - Utilizando operadores

```
#include <stdio.h>

#include <stdio_ext.h>

#include <stdlib.h>

int main() {
```

```
    // declara uma variável inteira

    int a;

    // declara um ponteiro inteiro

    int *p;

    // atribui o valor 12 a variável "a"

    a = 12;

    // aponta o ponteiro "p" para variável "a"

    // "p" recebe o endereço de memória da variável "a"

    p = &a;

    // apresenta o conteúdo de "p" (endereço de memória apontado por "p")

    printf("\nConteúdo de p = %p", p);

    // apresenta o conteúdo do endereço de memória apontado por "p"

    printf("\nConteúdo do endereço apontado por p = %i", *p);

    printf("\n");

    return 0;

}
```

Declarando Ponteiros - Utilizando Operadores.

(Arquivo-fonte: 12 - Ponteiros/ponteiro_operadores_mais.c)

```
Terminal - g1l3du4rd0@g1l3du4rd0-K46CB: ~/D
Arquivo  Editar  Ver  Terminal  Abas  Ajuda
g1l3du4rd0@g1l3du4rd0-K46CB:~/Documentos$ ./a.out

Conteúdo de p = 0x7ffc74b3492c
Conteúdo do endereço apontado por p = 12
g1l3du4rd0@g1l3du4rd0-K46CB:~/Documentos$
```

Resultado da execução do código anterior

Alocação de Memória - Função *malloc()*

```
#include <stdio.h>

#include <stdio_ext.h>

#include <stdlib.h>

int main() {

   int a, total;

   // declara um ponteiro inteiro

   int *p;

   printf("Digite o total de números: ");

   scanf("%i", &total);

   // aloca a quantidade de memória de acordo

   // como o total de números especificado

   p = malloc(total * sizeof(int));

   // percorre a área de memória alocada e apontada por "p"

   // armazenando valores inteiros nela

   for(a=0; a<total; a++) {

       p[a] = 10 + a;

       printf("%i ", p[a]);

   }

   printf("\n");

   return 0;

}
```

Alocação de Memória - Função *malloc()*.

(Arquivo-fonte: 12 - Ponteiros/alocacao_malloc.c)

```
Terminal - g1l3du4rd0@g1l3du4rd0-K46CB: ~/D
Arquivo  Editar  Ver  Terminal  Abas  Ajuda
g1l3du4rd0@g1l3du4rd0-K46CB:~/Documentos$ ./a.out
Digite o total de números: 5
10 11 12 13 14
g1l3du4rd0@g1l3du4rd0-K46CB:~/Documentos$
```

Resultado da execução do código anterior

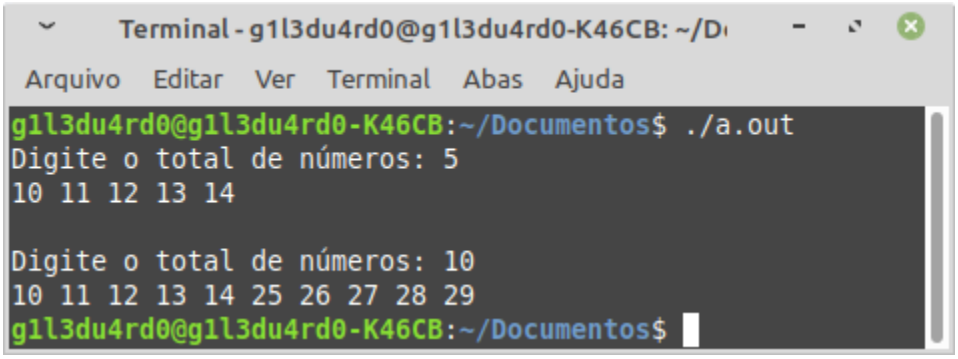
Realocação de Memória - Função *realloc()*

```c
#include <stdio.h>

#include <stdio_ext.h>

#include <stdlib.h>

int main() {

   int a, total;

   // declara um ponteiro inteiro

   int *p;

   printf("Digite o total de números: ");

   scanf("%i", &total);

   // aloca a quantidade de memória de acordo

   // como o total de números especificado

   p = malloc(total * sizeof(int));

   // percorre a área de memória alocada e apontada por "p"

   // armazenando valores inteiros nela

   for(a=0; a<total; a++) {

       p[a] = 10 + a;

       printf("%i ", p[a]);

   }

   printf("\n\nDigite o total de números: ");

   scanf("%i", &total);

   // realoca a quantidade de memória de acordo
```

```
    // como o total de números especificado, mantendo
    // o conteúdo já armazenado anteriormente
    p = realloc(p, total * sizeof(int));
    // percorre a área de memória realocada e apontada por "p"
    // armazenando novos valores inteiros nela
    for(a; a<total; a++) {
        p[a] = 20 + a;
    }
    // Exibe todo o conteúdo armazenado na memória realocada
    // e apontada por "p"
    for(a=0; a<total; a++) {
        printf("%i ", p[a]);
    }
    printf("\n");
    return 0;
}
```

Realocação de Memória - Função *realloc()*.

(Arquivo-fonte: 12 - Ponteiros/realocacao_realloc.c)



Resultado da execução do código anterior

Biblioteca <string.h> - Função *strlen()*

```
#include <stdio.h>

#include <stdio_ext.h>

#include <stdlib.h>

#include <string.h>

int main() {

   int total;

   // declara um ponteiro char e aloca memória

   char *str = malloc(50 * sizeof(char));

   printf("Digite uma string: ");

   gets(str);

   // utiliza a função strlen() para obter a quantidade

   // de letras da string "str"

   total = strlen(str);

   // apresenta o total de letras obtido pela função

   // strlen()

   printf("Total de letras: %i", total);

   printf("\n");

   return 0;

}
```

Obtendo o Tamanho de uma String - Função *strlen()*.

(Arquivo-fonte: 12 - Ponteiros/string_strlen.c)

```
Terminal - g1l3du4rd0@g1l3du4rd0-K46CB: ~/D
Arquivo  Editar  Ver  Terminal  Abas  Ajuda
g1l3du4rd0@g1l3du4rd0-K46CB:~/Documentos$ ./a.out
Digite uma string: Gil Eduardo de Andrade
Total de letras: 22
g1l3du4rd0@g1l3du4rd0-K46CB:~/Documentos$
```

Resultado da execução do código anterior

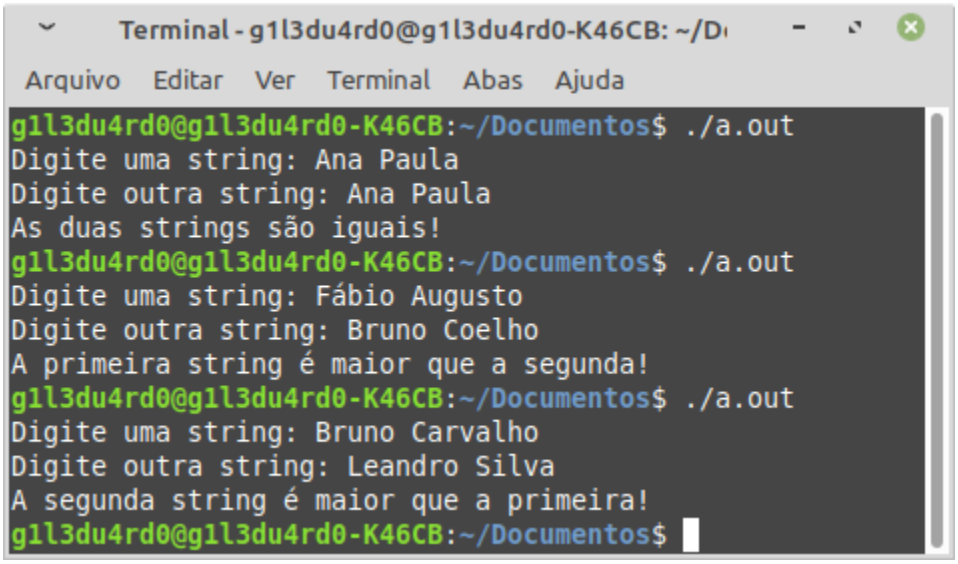
Biblioteca <string.h> - Função *strcmp()*

```
#include <stdio.h>

#include <stdio_ext.h>

#include <stdlib.h>

#include <string.h>

int main() {

   int ret;

   // declara dois ponteiros char e aloca memória

   char *str1 = malloc(50 * sizeof(char));

   char *str2 = malloc(50 * sizeof(char));

   printf("Digite uma string: ");

   gets(str1);

   printf("Digite outra string: ");

   gets(str2);

   // utiliza a função strcmp() para comparar as duas

   // strings digitadas

   ret = strcmp(str1, str2);

   if(ret == 0) {

       printf("As duas strings são iguais!");

   }

   else if(ret > 0) {

       printf("A primeira string é maior que a segunda!");
```

```
    }
    else if(ret < 0){
        printf("A segunda string é maior que a primeira!");
    }
    printf("\n");
    return 0;
}
```

Comparando duas Strings - Função *strcmp()*.

(Arquivo-fonte: 12 - Ponteiros/string_strcmp.c)



Resultado da execução do código anterior

Biblioteca <string.h> - Função *strcpy()*

```
#include <stdio.h>
#include <stdio_ext.h>
#include <stdlib.h>
#include <string.h>
```

```c
int main() {

   // declara dois ponteiros char e aloca memória

   char *origem = malloc(50 * sizeof(char));

   char *destino = malloc(50 * sizeof(char))

   printf("Digite uma string: ");

   gets(origem);

   // utiliza a função strcpy() para copiar o conteúdo da

   // variável "origem" para variável "destino"

   strcpy(destino, origem);

   // apresenta o conteúdo copiado para variável "destino"

   // através do uso da função strcpy()

   printf("Conteúdo copiado: %s", destino);

   printf("\n");

   return 0;

}
```

Copiando uma String - Função *strcpy()*.

(Arquivo-fonte: 12 - Ponteiros/string_strcpy.c)

```
Terminal - g1l3du4rd0@g1l3du4rd0-K46CB: ~/D
Arquivo  Editar  Ver  Terminal  Abas  Ajuda
g1l3du4rd0@g1l3du4rd0-K46CB:~/Documentos$ ./a.out
Digite uma string: Instituto Federal
Conteúdo copiado: Instituto Federal
g1l3du4rd0@g1l3du4rd0-K46CB:~/Documentos$
```

Resultado da execução do código anterior

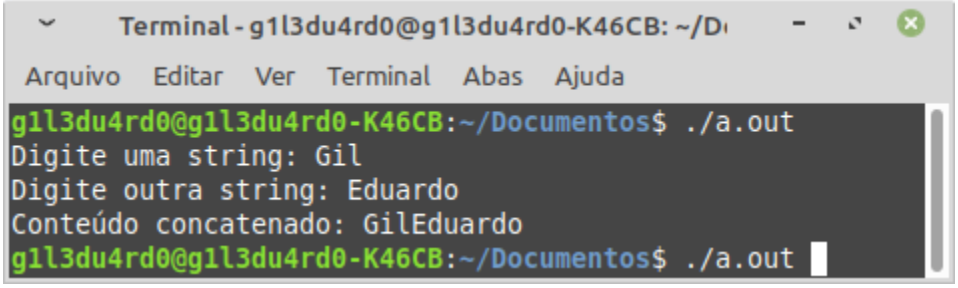
Biblioteca <string.h> - Função *strcat()*

```c
#include <stdio.h>
```

```c
#include <stdio_ext.h>

#include <stdlib.h>

#include <string.h>

int main() {

   // declara dois ponteiros char e aloca memória

   char *origem = malloc(50 * sizeof(char));

   char *destino = malloc(50 * sizeof(char));

   printf("Digite uma string: ");

   gets(destino);

   printf("Digite outra string: ");

   gets(origem);

   // utiliza a função strcat() para concatenar/juntar o conteúdo da

   // variável "destino" com o conteúdo da variável "origem"

   strcat(destino, origem);

   // apresenta o conteúdo concatenado dentro da variável "destino"

   // através do uso da função strcat()

   printf("Conteúdo concatenado: %s", destino);

   printf("\n");

   return 0;

}
```

Concatenando (juntando) duas Strings - Função *strcat()*.

(Arquivo-fonte: 12 - Ponteiros/string_strcat.c)



Resultado da execução do código anterior