Argumentos da Função `main(.)`

Passando argumentos para a função `main(.)` por meio da linha de comandos do ***Sistema Operacional***.

```
int main(int arg_c, char *arg_v[])
```

```
int main(int arg_c, char **arg_v)
```

O objetivo da aula de hoje é entender os argumentos (parâmetros formais) associados a função `main`(), enfim, quais os seus significados e implicações.

Argumentos da Função `main(.)`

Há muito, muito tempo atrás, antes do advento das modernas ***interfaces gráficas*** a comunicação entre o usuário e o computador se dava por meio de um outro tipo de interface, a famigerada, temida, amada e odiada

```
> Linha de comandos
```

Significa que para rodar um aplicativo qualquer ou um comando do SO, você deveria digitar o seu nome a partir da linha de comandos e, se desejasse, também poderia especificar outros parâmetros.

Argumentos da Função `main(.)`

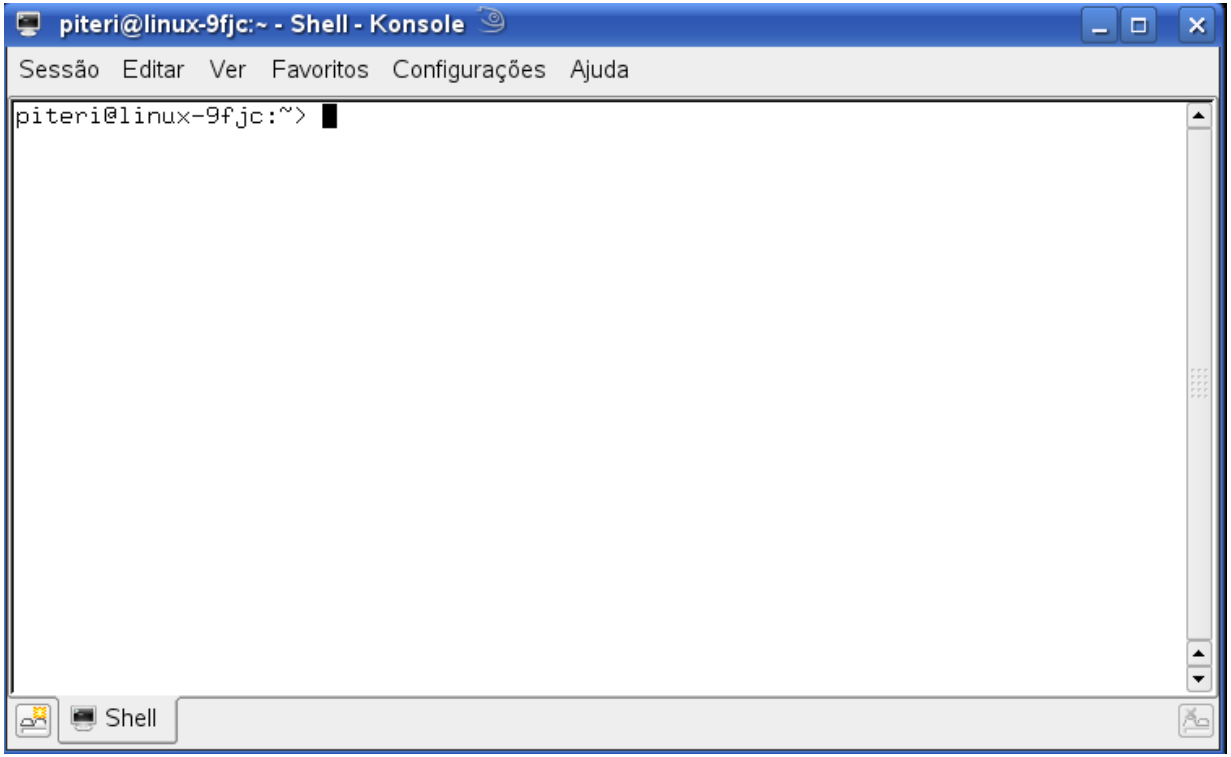
Quem é mais “velho” ou já teve a oportunidade de usar o MS-DOS, ou ainda, o sistema operacional Unix ou Linux, certamente já utilizou a “console” em que é necessário passar parâmetros por meio da linha de comandos.

Recordar é viver. Alguns comandos em MS-DOS

```
C:\ > format a:/s
C:\ > copy a:*.txt b:
C:\ > type d:\arquivo.txt
C:\ > del a:*.*
```

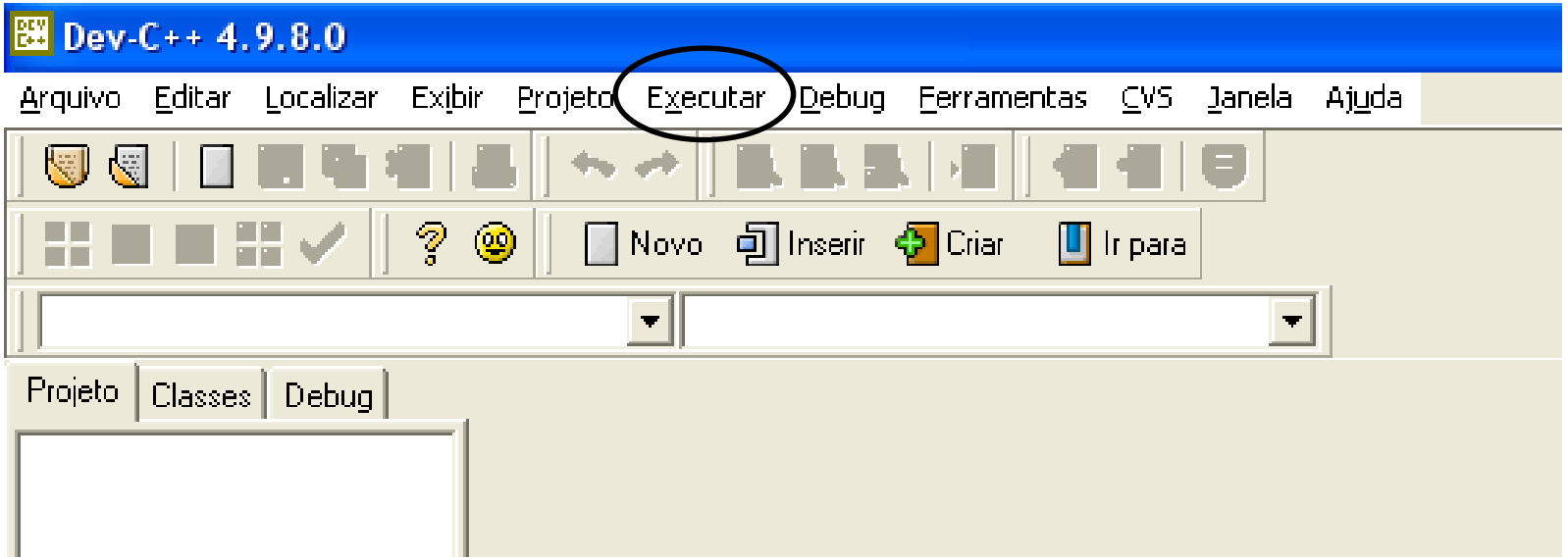
Argumentos da Função `main(.)`

A linha de comandos (console) ainda é muito comum para usuários que utilizam Unix/Linux e encontram maior produtividade (para muitas tarefas) a partir de seu uso.



Argumentos da Função `main(.)`

Até hoje, todas as vezes que executamos nossos programas, o fazemos a partir da **IDE** (**I**ntegrated **D**evelopment **E**nviroment) do ambiente de programação (Dev-C++, CodeBlocks, ...).



Argumentos da Função `main(.)`

Em todos os programas anteriores que desenvolvemos, sem exceção, a função `main(.)` assumia uma das formas abaixo e os ***dados de entrada*** eram introduzidos a partir do teclado por meio de instruções de leitura usando o comando `scanf(.)`. Mais a frente, vamos fazer a leitura dos ***dados de entrada*** a partir de arquivos (dispositivos externos de armazenamento).

```
int main(void) {

... Códigos ...

return(0);
}
```

```
int main( ) {

... Códigos ...

return(0);
}
```

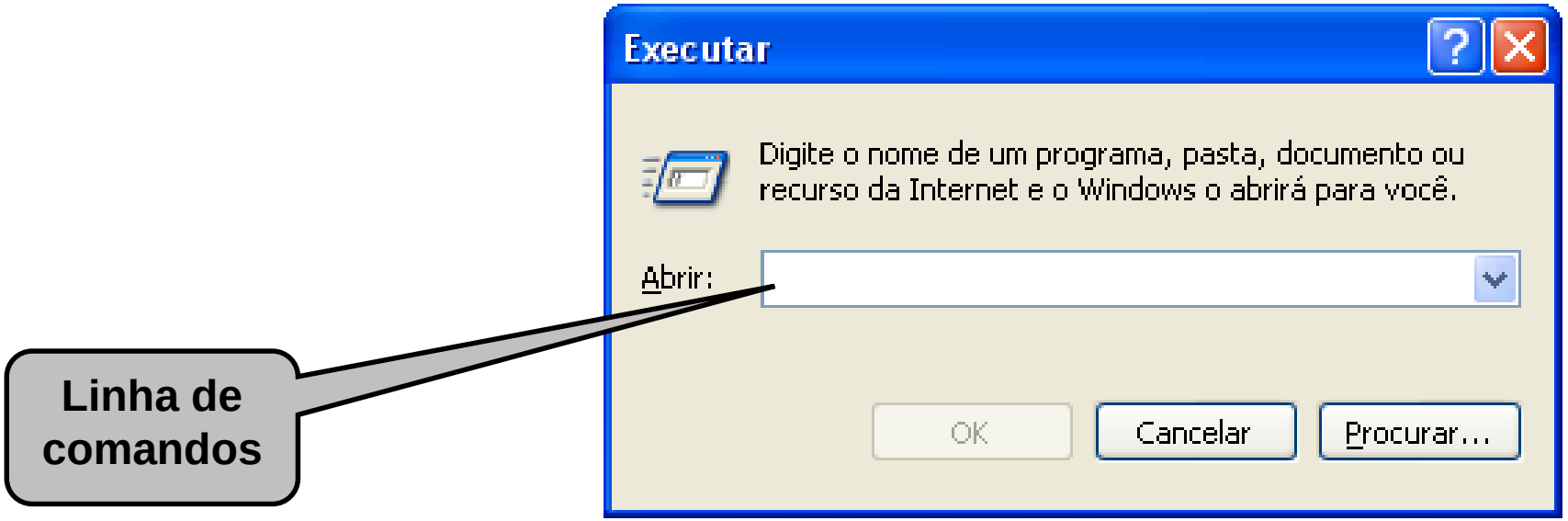
ou seja, a função `main` de nossos programas nunca teve quaisquer parâmetros.

Strings

Argumentos da Função `main(.)`

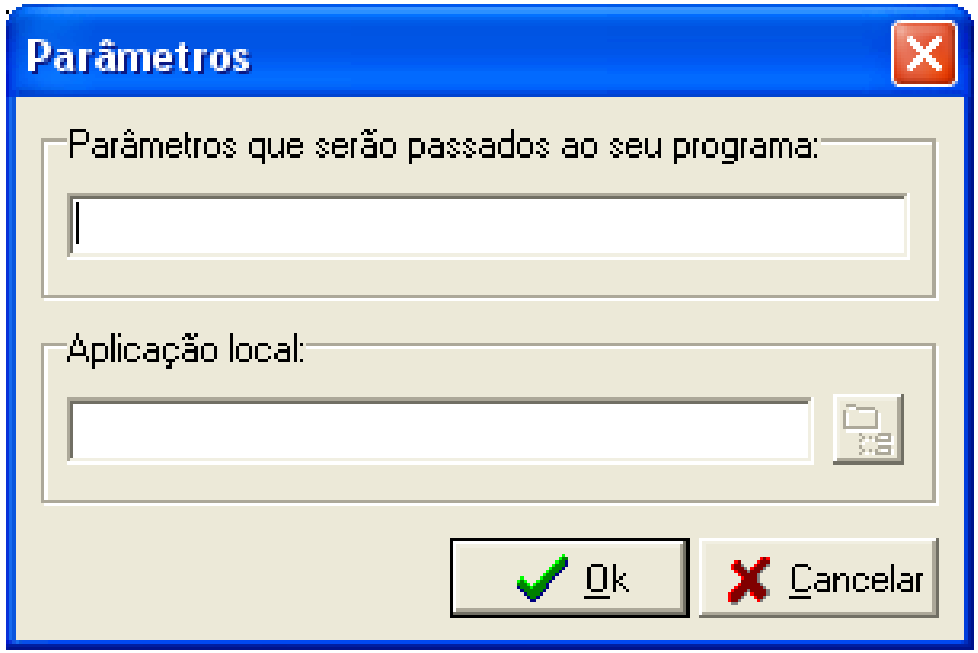
Quando o SO assume o controle de um código em ***C*** (executável), a 1ª função a ser executada é a função `main()`, assim, os argumentos introduzidos na linha de comandos são passados pelo SO como argumentos para a `main()`.

Quando for necessário executar seu código a partir da **linha de comandos**, alguns argumentos podem ser exigidos e você pode fazer isso a partir da opção abaixo:



Argumentos da Função `main(.)`

No ambiente Dev-Cpp, uma outra alternativa é passar os argumentos para a função `main()` através da opção de menu **Executar→Parametros**, que abre a janela abaixo, onde podem ser colocados os valores de entrada (dados esperados) para que seu programa execute.



Argumentos da Função `main(.)`

Para o SO poder fazer a passagem de parâmetros para a função `main()`, precisamos modificar a declaração de `main()`, que agora passa a ser:

```
int main(int arg_c, char *arg_v[])
```

Onde, `argc` e `argv` são os parâmetros formais da função `main`, que recebem os valores passados pelo SO. O par de colchetes vazio (`[]`) em `arg_v`, indica que o tamanho desse vetor é indeterminado.

`arg_c (argument count)`: Valor inteiro

armazena o número de argumentos que foram passados pela linha de comandos. O nome do programa é o primeiro dos parâmetros. Logo, `arg_c` ≥ 1.

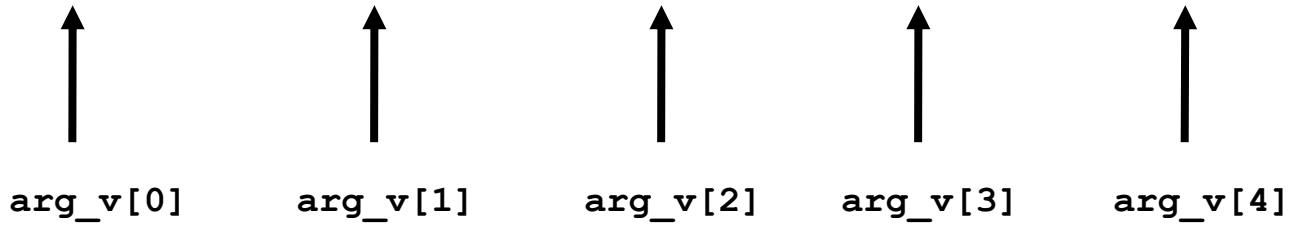
Argumentos da Função `main(.)`

`arg_v (argument vector)`:

Array/vetor/arranjo de ponteiros para **char**, ou seja, cada componente aponta para uma *string* que está associada a um dos argumentos. Se houver valores numéricos, eles devem ser convertidos pelo programa fazendo-se uso das funções de conversão. O número de ponteiros é igual ao valor de `arg_c`.

Exemplo: Ao executarmos `programa` a partir da linha de comandos, como ilustrado abaixo:

```
C:programa primeiro segundo terceiro quarto
```

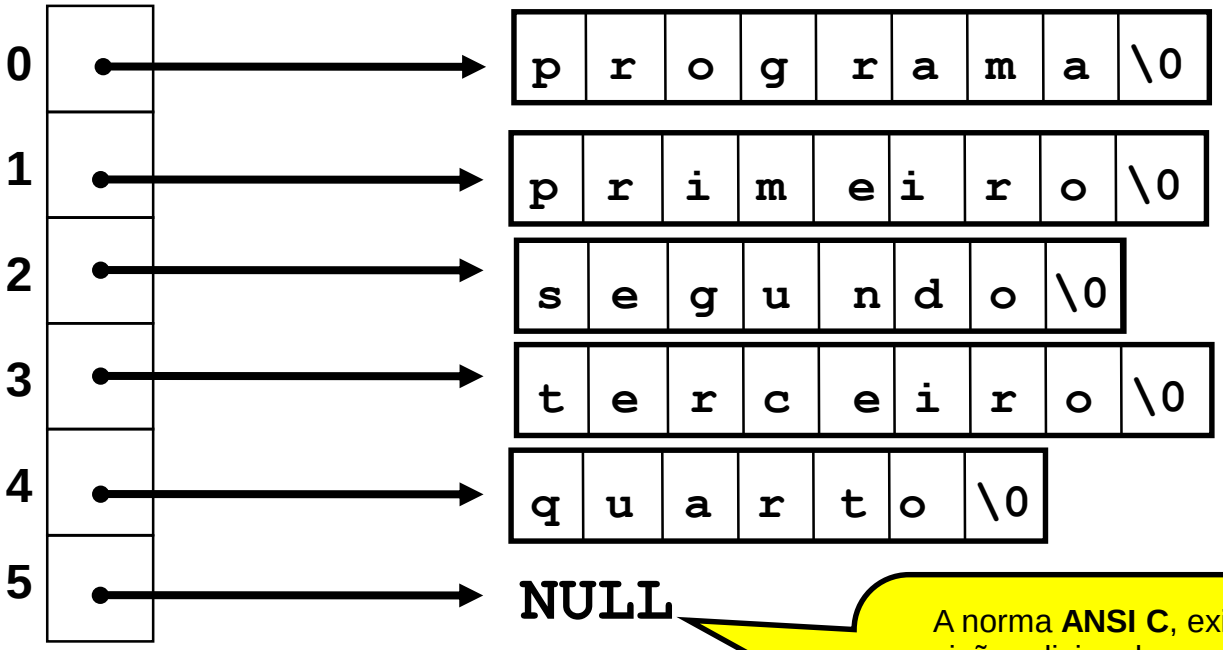


Strings

Argumentos da Função `main(.)`

```
C:programa primeiro segundo terceiro quarto
```

`arg_c = 5` `arg_v`



A norma **ANSI C**, exige uma posição adicional ao vetor `arg_v` de modo que o elemento associado a essa posição aponte para **NULL** e possamos também detectar o fim através do uso de ponteiros.

Observação: Para ser mais rigoroso os caracteres deveriam estar entre apóstrofes (' ').

Argumentos da Função `main(.)`

Uma outra alternativa é usar a forma abaixo:

```
int main(int arg_c, char **arg_v)
```

Observações: `**arg_v`: Lembre-se que o nome de um vetor aponta para o endereço (primeiro byte) do primeiro elemento/componente.

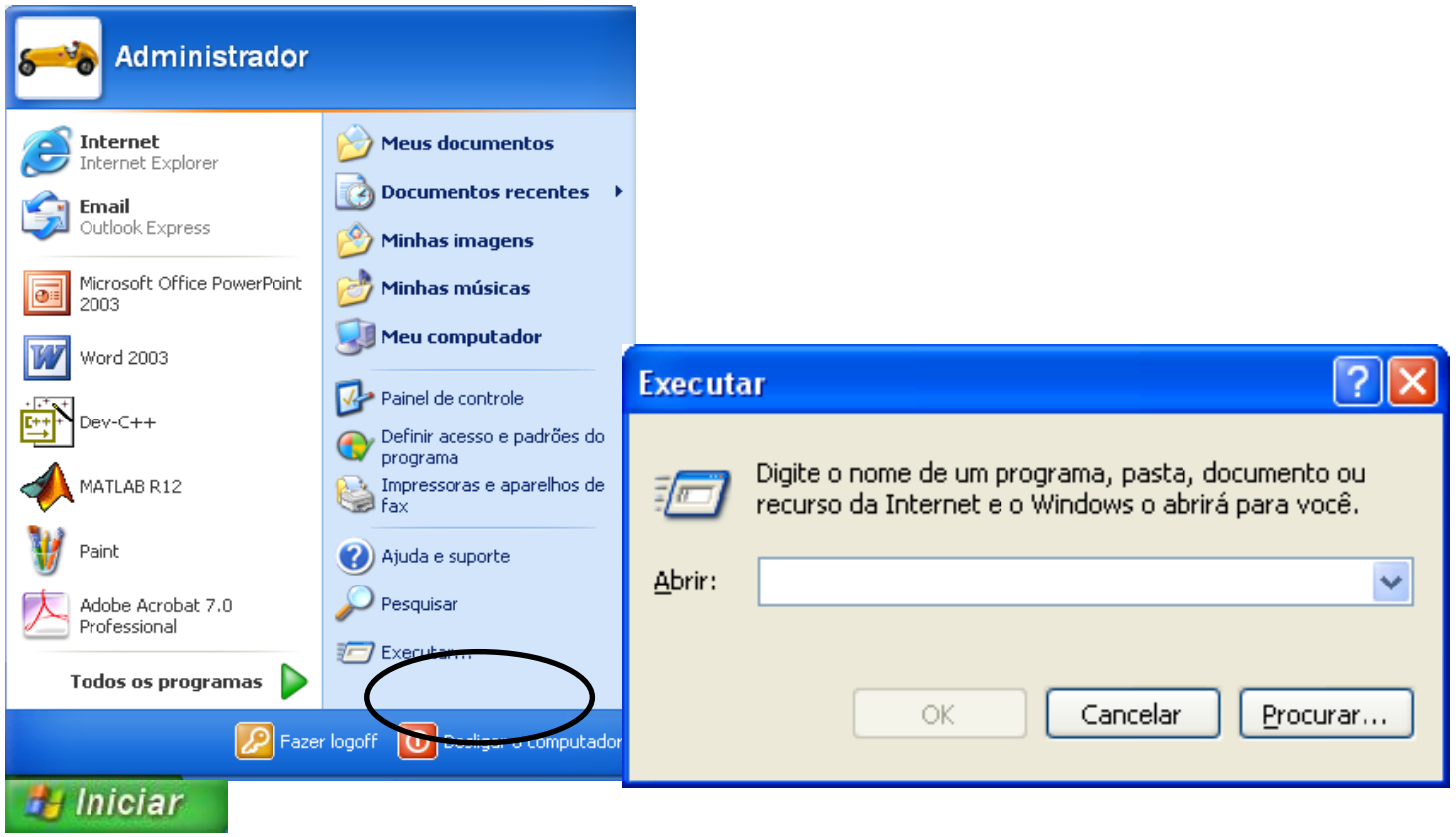
Argumentos da Função `main(.)`

Agora, ao escrevermos nossos programas podemos verificar se o número de parâmetros introduzidos pela linha de comandos está correto. Assim, podemos acrescentar algumas instruções a mais no interior da função `main(.)`, de modo a fazer essa verificação. Por exemplo, supondo que desejamos rodar um programa executável de nome **Fibonacci** e devemos passar o i-ésimo termo a ser calculado.

```
int main(int arg_c, char *arg_v[]) {

    if (arg_c != 2) {
      printf("Erro na entrada de Dados!!!\n");
      printf("Obedeça a sintaxe: < Fibonacci n1 > \n");
      printf("Tente novamente!!!");
      exit(1);          //término incondicional do programa
    }

..... // escrita de seu código aqui como sempre foi feito

return(0);
}
```

Argumentos da Função `main(.)`

- Sempre é possível criar um executável a partir do IDE e rodá-lo pela linha de comando do SO, no caso do Windows 7, …, por meio da opção **Iniciar->Executar.**



Argumentos da Função `main(.)`

Exemplo 01: Elabore um programa em C para ler argumentos da linha de comandos, imprimir o número de argumentos e os respectivos conteúdos associados a cada um deles.

```
#include <stdio.h>
int main(int arg_c, char *arg_v[]) {
int i=0;

  printf("Número de argumentos em argv[] = %d \n",arg_c);
  for (; i<arg_c;i++)
    printf("Argumento(%d) = %s ",i, arg_v[i]);

  .....

return(0);
}
```

Executar código exemplo:→
Lcomando1

Argumentos da Função `main(.)`

Terceiro Parâmetro da função `main`

Muitos compiladores aceitam que a função `main` receba um terceiro parâmetro, que também é um vetor em que cada componente aponta para uma *string* (análogo a `arg_v`) e contém as variáveis de ambiente do SO.

De forma contrária ao `arg_v`, esse terceiro parâmetro não possui uma variável inteira contadora associado ao número de variáveis do ambiente, mas sabemos que após a última posição válida, esse vetor aponta para **NULL**.

Argumentos da Função `main(.)`

Terceiro Parâmetro da função `main`

Todas as variáveis de ambiente possui um nome simbólico (maioria em letras maiúsculas), seguido do símbolo de igual e de seu valor. Abaixo podemos ver alguns exemplos dessas variáveis.

```
INCLUDE, LIB, HOMEDRIVE, COMPUTERNAME, LONGSERVER,
HOMEPATH, PATHEXT, NUMBER_OF_PROCESSORS, Path, TEMP,
TMP,PROCESSOR_ARCHITECTURE, ...
```

O ambiente Dev-C++ implementa esse recurso. Logo, execute o programa do próximo *slide* e verifique quais são os valores de algumas variáveis de ambiente de sua máquina. O protótipo da função `main` agora assume a forma:

```
int main(int arg_c, char *arg_v[], char *arg_vv[]);
```

Strings

Argumentos da Função `main(.)`

```
#include <stdio.h>

int main(int arg_c, char *arg_v[], char *arg_vv[]) {
int i=0;

  printf("Número de argumentos em argv[] = %d \n",arg_c);

  for (; i<arg_c;i++)
    printf("Argumento(%d) = %s \n\n",i, *(arg_v + i) );

  for ( ; *arg_vv != NULL; arg_vv++)  // Variaveis de ambiente
    printf(" %s \n\n", *(arg_vv) );



system("Pause");
return(0);
}
```

Executar código exemplo:→
Lcomando2

Exercícios

01) Elabore um programa em ***C*** para concatenar/juntar/*merge* várias strings numa única *string* através da linha de comandos. No exemplo abaixo, `str1` é a string resultante da concatenação das strings `str2`, `str3`, `str4` e `str5`, enquanto `concatenacao` é o nome do executável. Aqui, o número de parâmetros é igual a 6.

```
C:\caminho\concatenacao str1 str2 str3 str4 str5
```

02) Elabore um programa em ***C*** que some todos os parâmetros passados (números inteiros) na linha de comandos, com exceção do nome do programa.

Solução Exercício 02

Versão 1: →

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

int main(int arg_c, char *arg_v[]) {
int i=1,                // não somar o primeiro parâmetro
 soma = 0;

 printf("Número de argumentos em argv[] = %d \n",arg_c);
 for (; i<arg_c; i++) soma += atoi( arg_v[i] );

printf("\nForam passados %d valores a serem somados", --arg_c);
printf("\n\n Soma Obtida = %d ",soma);

system("Pause");
return(0);
}
```

Solução Exercício 02

Versão 2: →

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

int main(int arg_c, char **arg_v) {
int  soma = 0;

 printf("Número de argumentos em argv[] = %d \n",arg_c);
 arg_v++;   // não somar o primeiro parâmetro
 for (; *arg_v != NULL; arg_v++) soma += atoi( *(arg_v) );

printf("\nForam passados %d valores a serem somados", --arg_c);
printf("\n\n Soma Obtida = %d ",soma);

system("Pause");
return(0);
}
```

Argumentos da Função `main(.)`

Observações:

- Os nomes `arg_c` e `arg_v` associados aos parâmetros formais da função `main` são utilizados tradicionalmente. Obviamente que qualquer nome pode ser usado. Mas lembre-se, muitas vezes é bom manter a ***tradição***;

- O número de parâmetros suportados por `argv` são dependentes do SO. Uma boa consulta ao manual do sistema ajuda a resolver o problema. Por exemplo, o MS-DOS (o primeiro SO desenvolvido pela Microsoft e que é o bisavô do Windows) permitia no máximo 128 caracteres por linha;

- Os parâmetros na linha de comandos devem vir separados uns dos outros, por um espaço, ou, por um caractere de tabulação;

Argumentos da Função `main(.)`

Observações:

- Um outro uso bastante comum da passagem de parâmetros pela linha de comandos é o de *nome de arquivos*. Quando isso ocorrer, é natural que os arquivos associados ao programa executável estejam no mesmo diretório (manipulação de arquivos);

- O uso de argumentos na linha de comandos facilita a distribuição de seu código executável. Além disso, algumas informações básicas/adicionais de seu programa podem ser colocadas no início da execução do código para realçar características que você (programador) ache importante;

- Um outro uso de argumentos na linha de comandos é a criação de arquivos batch (*batchs files*);

TRABALHO DE PROGRAMAÇÃO – (2016)

Suponha a existência de dois arquivos de números inteiros (ordenados) com tamanho m e n, respectivamente. Escreva um programa que lê ambos os arquivos e seja capaz de gerar um terceiro arquivo, de modo que este último também esteja ordenado e seja relativo a uma das operações abaixo:

- intersecção entre os dois conjuntos (**1**);
- união entre os dois conjuntos (**2**);
- diferença A-B (**3**);
- diferença B-A (**3**);

Lembrem-se, parâmetros do tipo streams (arquivos) só podem ser passados por endereço.

Para esse trabalho eu não quero a elaboração de uma interface. Todas as operações possíveis deverão ser realizadas através da linha de comandos. Observem que, nesse caso a linha de comandos poderá ter diferentes números de parâmetros. Vamos imaginar que seu executável tenha nome `conjunto` e que os arquivos sejam `arq1` e `arq2`, respectivamente. Então:

TRABALHO DE PROGRAMAÇÃO

Possíveis comandos a partir da **Linha de Comandos**:

conjunto t arq1 n	Cria um arquivo texto com nome arq1 com n elementos ordenados crescentemente;
conjunto b arq1 n	Cria um arquivo binário com nome arq1 com n elementos ordenados crescentemente;
conjunto t 1 arq1 arq2 arq3	Cria o arquivo texto com nome arq3 que é resultado da intersecção dos elementos dos arquivos textos arq1 e arq2;
conjunto b 1 arq1 arq2 arq3	Cria o arquivo binário com nome arq3 que é resultado da intersecção dos elementos dos arquivos binários arq1 e arq2;

TRABALHO DE PROGRAMAÇÃO

Possíveis comandos a partir da **Linha de Comandos**:

conjunto t 2 arq1 arq2 arq3	Cria o arquivo texto com nome arq3 que é resultado da união dos elementos dos arquivos textos arq1 e arq2;
conjunto b 2 arq1 arq2 arq3	Cria o arquivo binário com nome arq3 que é resultado da união dos elementos dos arquivos binários arq1 e arq2;
conjunto t 3 arq1 arq2 arq3	Cria o arquivo texto com nome arq3 que é resultado da diferença dos elementos do arquivo texto arq1, menos os elementos do arquivo texto arq2;
conjunto b 3 arq2 arq1 arq3	Cria o arquivo binário com nome arq3 que é resultado da diferença dos elementos do arquivo binário arq2, menos os elementos do binário arq1;

TRABALHO DE PROGRAMAÇÃO

Possíveis comandos a partir da **Linha de Comandos**:

conjunto t arq1	Lista na tela os elementos pertencentes ao arquivo do tipo texto arq1;
conjunto b arq1	Lista na tela os elementos pertencentes ao arquivo do tipo binário arq1;
conjunto	Lista na tela uma breve descrição das funcionalidades do programa e a sintaxe de seu uso.

Observe que o número de parâmetros permitidos pode ser 1, 3, 4 ou 6.

A parte do trabalho relativa a manipulação de arquivos do tipo binário vale 60%, o restante corresponde a manipulação de arquivos texto.

Prazo final de entrega: 17/01/2018 até às 24h