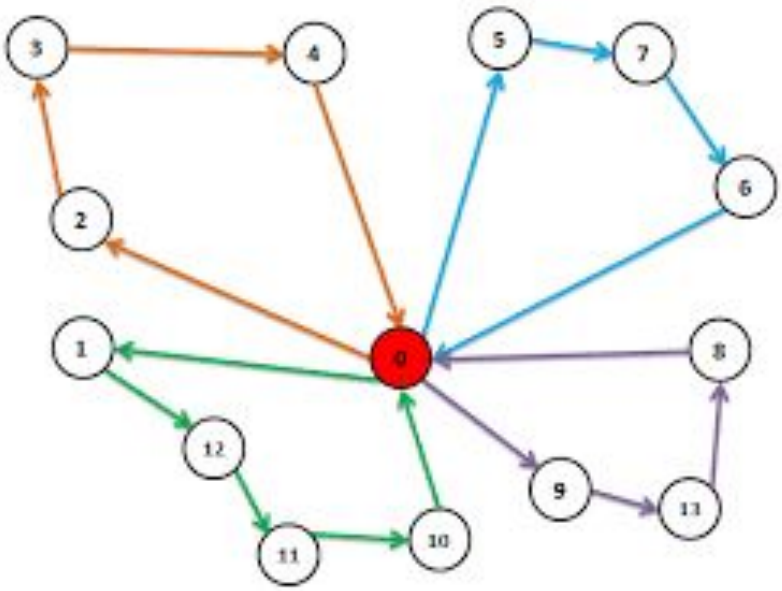
Programação

Flavius Gorgônio
flavius@dct.ufrn.br

Agenda

- Importância da modularização
- Funções e procedimentos
- Entrada e saída de funções
- Passagem de parâmetros
  - Por valor
  - Por referência
- Compilação de módulos em separado

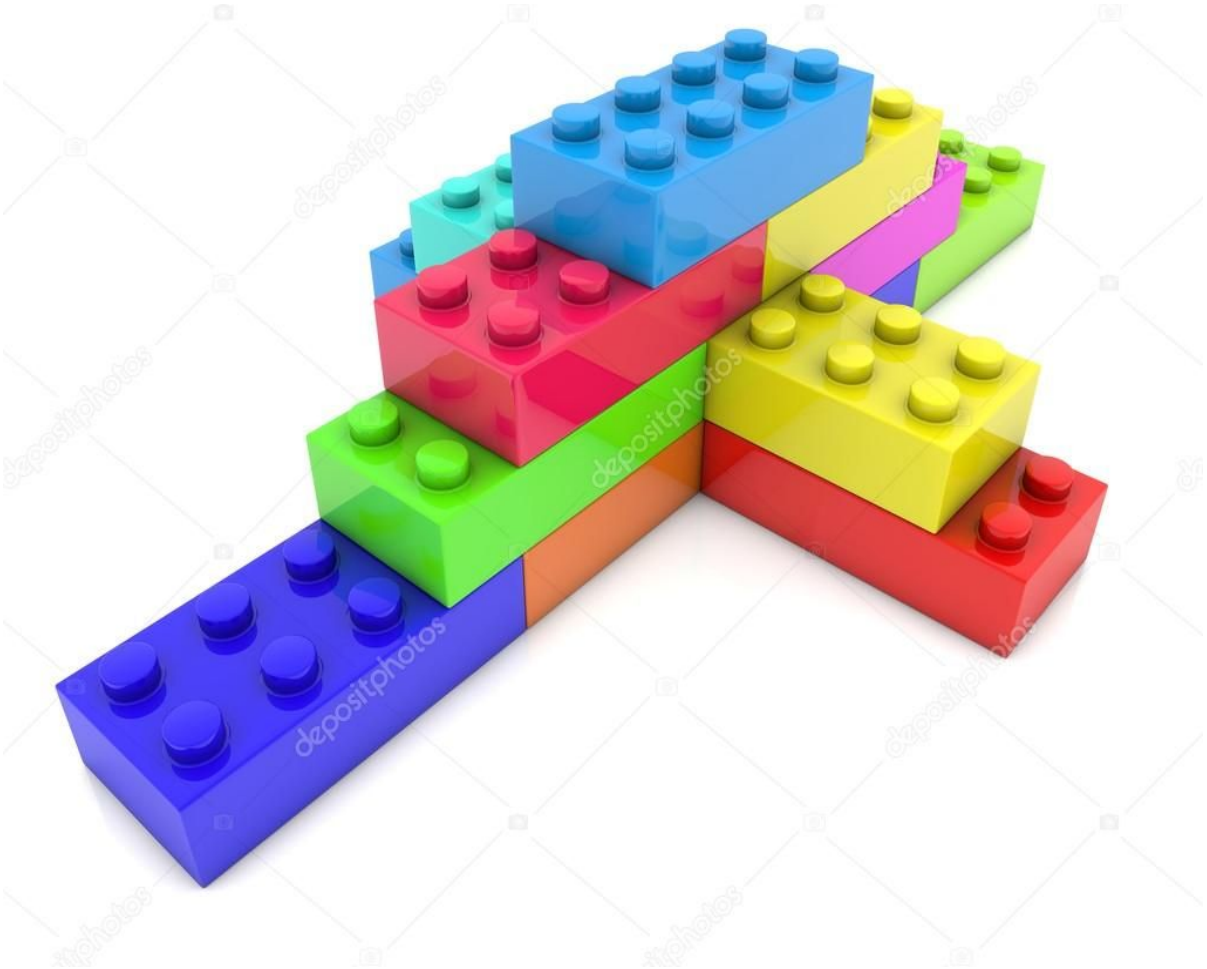
Subprogramas e modularização



- Programas de computador são (um bloco de) linhas de código, executadas sequencialmente, para atender a um determinado objetivo (ver o conceito de algoritmo)
- Por vezes, o fluxo de execução das instruções precisa ser modificado ao longo da atividade, seja por condições de entrada ou por escolhas do usuário
- Alguns trechos de código precisam ser repetidos várias vezes ao longo da execução de um programa
- Outros trechos de código são tão úteis que podem ser reaproveitados e reutilizados até mesmo em outros programas (rotinas de validação, por exemplo)

Modularização de programas

1. Por que criar subprogramas?
2. É possível programar sem dividir o programa em módulos?
3. É possível desenvolver sistemas mais complexos sem dividir o programa em módulos?
4. Como escrever módulos reutilizáveis?
5. Como potencializar a reutilização dos módulos desenvolvidos?



A função main( )

- Todo programa em C deve ter uma, e apenas uma, função **main()**
- A função **main()** é o ponto de entrada para execução do programa, é a primeira função a ser executada
- A função **main()** retorna um valor inteiro para o sistema operacional, sinalizando se o programa foi encerrado corretamente ou se houve algum erro durante a sua execução
- A função **main()** também pode receber parâmetros, que são passados quando o programa é executado

```c
#include <stdio.h>

int main(void) {
  printf("Hello World\n");
  return 0;
}

// parâmetros: aquilo que a função recebe quando é chamada
// void: nulo

// retorno: aquilo que a função devolve quando é encerrada
// int: status de encerramento do programa
```

https://repl.it/@flaviusgorgonio/AFuncaoMainc

Fluxo de execução de uma função

```c
#include <stdio.h>

// Assinatura da função
void exibeMensagem(void);

int main(void) {
 printf("Hello World\n");
 exibeMensagem();
 return 0;
}

// Corpo da função
void exibeMensagem(void) {
 printf("Olá pessoal\n");
 printf("Tudo bem com vocês?\n");
 printf("Até logo...\n");
}
```

- A execução do programa inicia pela função main(), a partir daí, as demais funções do programa são chamadas
- É necessário incluir a assinatura da função exibeMensagem() no início do código para que o compilador saiba que a função existe e quais são os parâmetros de entrada e saída da mesma
- Quando a função exibeMensagem() é chamada, a execução da função main() é suspensa
- Após a execução da função exibeMensagem() ser concluída, a função main() continua a execução de onde parou

https://replit.com/@flaviusgorgonio/estruturaDeUmaFuncaoc

Tipos de subprogramas: procedimentos e funções

Procedimentos não retornam valor

Sintaxe:

```
void nome_funcao(<parametros>) {
 // Corpo da função
}
```

Semântica:

Ao ser chamado, o subprograma recebe parâmetros (opcionais) e executa um trecho de código

Por ser void, não há retorno para a unidade chamadora

Entretanto, após a função ser encerrada, o fluxo de execução retorna à unidade chamadora

```c
#include <stdio.h>

// Assinatura do subprograma
void exibeMensagem(void);

int main(void) {
  printf("Hello World\n");
  exibeMensagem();
  return 0;
}

// Corpo do subprograma
void exibeMensagem(void) {
  printf("Olá pessoal\n");
  printf("Tudo bem com vocês?\n");
  printf("Até logo...\n");
}
```

https://replit.com/@flaviusgorgonio/estruturaDeUmaFuncaoc

Tipos de subprogramas: procedimentos e funções

Funções retornam valor

Sintaxe:

```
<tipo> nome_funcao(<parametros>) {
 // Corpo da função
 return <var>;
}
```

Semântica:

Ao ser chamada, a função recebe parâmetros (opcionais), executa um trecho de código e retorna um valor para a unidade chamadora

Após a função ser encerrada, o fluxo de execução retorna à unidade chamadora

Exemplos de funções:

```
int dado(void) {
  return 1 + rand() % 6;
}
```

```
int bissexto(int ano) {
  if ((ano % 4 == 0) && (ano % 100 != 0)) {
    return 1;
  } else if (ano % 400 == 0) {
    return 1;
  } else {
    return 0;
  }
}
```

https://replit.com/@flaviusgorgonio/DadoComFuncaoc

Parâmetros de entrada e saída

```c
#include <stdio.h>

int somaValor(int, int);

int main(void) {
 int v1, v2, soma;
 printf("Programa Soma Números\n");
 printf("Informe um valor inteiro: ");
 scanf("%d", &v1);
 printf("Informe outro valor inteiro: ");
 scanf("%d", &v2);
 soma = somaValor(v1, v2);
 printf("A soma %d + %d = %d\n", v1, v2, soma);
 return 0;
}

int somaValor(int a, int b) {
 return a + b;
}
```

- A função **somaValor()** possui dois parâmetros de entrada do tipo int, ou seja, recebe dois valores inteiros ao ser chamada
- Por sua vez, ela retorna um valor inteiro, que é devolvido para a unidade chamadora, ou seja, aquela função que chamou a função **somaValor()**
- A variável, expressão ou valor de retorno deve ser, necessariamente, do mesmo tipo de retorno da função
- Simplificando: a função **somaValor()** recebe dois valores inteiros e devolve um valor inteiro que corresponde à soma dos dois valores recebidos

https://replit.com/@flaviusgorgonio/FuncaoSomaValorc

Trocando valores de variáveis (versão errada)

A função abaixo "tenta" trocar os valores de duas variáveis:

```c
#include <stdio.h>

void trocaValor(int, int);

void trocaValor(int a, int b) {
  int aux;
  aux = a;
  a = b;
  b = aux;
  printf("\nDentro da função\n");
  printf("A = %d e B = %d\n", a, b);
}
```

```c
int main(void) {
  int a, b;
  printf("Programa Troca Valores - versão errada\n");
  printf("Informe dois valores inteiros\n");
  printf("Valor de A = ");
  scanf("%d", &a);
  printf("Valor de B = ");
  scanf("%d", &b);
  printf("\nValores originais\n");
  printf("A = %d e B = %d\n", a, b);
  printf("\nInvertendo os valores\n");
  trocaValor(a, b);
  printf("\nDe volta ao programa principal\n");
  printf("A = %d e B = %d\n", a, b);
  return 0;
}
```

https://replit.com/@flaviusgorgonio/FuncaoTrocaValoresErradac

Trocando valores de variáveis (versão correta)

A função abaixo realmente troca os valores das duas variáveis, pois atua diretamente nos endereços de memória onde as variáveis estão armazenadas:

```c
#include <stdio.h>

void trocaValor(int*, int*);

void trocaValor(int *apa, int *apb) {
 int aux;
 aux = *apa;
 *apa = *apb;
 *apb = aux;
}

// &a : endereço de a
// *a : conteúdo armazenado em a
```

```c
int main(void) {
  int a, b;
  printf("Programa Troca Valores\n");
  printf("Informe dois valores inteiros\n");
  printf("Valor de A = ");
  scanf("%d", &a);
  printf("Valor de B = ");
  scanf("%d", &b);
  printf("\nValores originais\n");
  printf("A = %d e B = %d\n", a, b);
  printf("\nInvertendo os valores\n");
  trocaValor(&a, &b);
  printf("A = %d e B = %d\n", a, b);
  return 0;
}
```

https://replit.com/@flaviusgorgonio/FuncaoTrocaValoresCorretac

Alterando valores de variáveis em funções

```c
#include <stdio.h>
float calculaMedia(float, float, float);
int main(void) {
  float n1, n2, n3, media;
  printf("Programa Calcula Média\n");
  printf("Informe a Nota 1: ");
  scanf("%f", &n1);
  printf("Informe a Nota 2: ");
  scanf("%f", &n2);
  printf("Informe a Nota 3: ");
  scanf("%f", &n3);
  media = calculaMedia(n1, n2, n3);
  printf("\nAs notas do aluno foram: \n");
  printf("Nota 1: %.1f\n", n1);
  printf("Nota 2: %.1f\n", n2);
  printf("Nota 3: %.1f\n", n3);
  printf("Média: %.1f\n", media);
  return 0;
}
```

```c
float calculaMedia(float n1, float n2, float n3) {
 if ((n1 > 3.0) && (n1 <= 9.5)) {
   n1 += 0.5;
 }
 if ((n2 > 3.0) && (n2 <= 9.5)) {
   n2 += 0.5;
 }
 if ((n3 > 3.0) && (n3 <= 9.5)) {
   n3 += 0.5;
 }
 return (n1 + n2 + n3) / 3;
}
```

https://replit.com/@flaviusgorgonio/FuncaoCalculaMedia1c

Alterando valores de variáveis em funções

```c
#include <stdio.h>
float calculaMedia(float*, float*, float*);
int main(void) {
  float n1, n2, n3, media;
  printf("Programa Calcula Média\n");
  printf("Informe a Nota 1: ");
  scanf("%f", &n1);
  printf("Informe a Nota 2: ");
  scanf("%f", &n2);
  printf("Informe a Nota 3: ");
  scanf("%f", &n3);
  media = calculaMedia(&n1, &n2, &n3);
  printf("\nA média do aluno é %.1f\n", media);
  printf("\nAs notas do aluno foram: \n");
  printf("Nota 1: %.1f\n", n1);
  printf("Nota 2: %.1f\n", n2);
  printf("Nota 3: %.1f\n", n3);
  return 0;
}
```

```c
float calculaMedia(float *n1, float *n2, float *n3) {
 if ((*n1 > 3.0) && (*n1 <= 9.5)) {
   *n1 += 0.5;
 }
 if ((*n2 > 3.0) && (*n2 <= 9.5)) {
   *n2 += 0.5;
 }
 if ((*n3 > 3.0) && (*n3 <= 9.5)) {
   *n3 += 0.5;
 }
 return (*n1 + *n2 + *n3) / 3;
}
```

https://replit.com/@flaviusgorgonio/FuncaoCalculaMedia2c

Alternativas para atualizar variáveis em funções

Passagem de parâmetros por referência

```c
#include <stdio.h>

void dobraValor(float*);

int main(void) {
  float sal;
  printf("Programa Dobra Salário\n");
  printf("Informe o valor do salário: R$ ");
  scanf("%f", &sal);
  dobraValor(&sal);
  printf("Novo salário: R$ %.2f\n", sal);
  return 0;
}

void dobraValor(float* sal) {
  *sal = (*sal) * 2.0;
}
```

https://replit.com/@flaviusgorgonio/FuncaoDobraSalario1c

Atualização por variável de retorno

```c
#include <stdio.h>

float dobraValor(float);

int main(void) {
  float sal;
  printf("Programa Dobra Salário\n");
  printf("Informe o valor do salário: R$ ");
  scanf("%f", &sal);
  sal = dobraValor(sal);
  printf("Seu salário será: R$ %.2f\n", sal);
  return 0;
}

float dobraValor(float sal) {
  return (2.0 * sal);
}
```

https://replit.com/@flaviusgorgonio/FuncaoDobraSalario2c

Validando a leitura de uma data (sem funções)

```c
#include <stdio.h>
int main(void) {
 int dia, mes, ano;
 int bissexto, maiorDia, dataValida;
 printf("Programa Data de Nascimento\n\n");
 do {
   printf("Informe sua data de nascimento\n");
   printf("Dia: ");
   scanf("%d", &dia);
   printf("Mês: ");
   scanf("%d", &mes);
   printf("Ano: ");
   scanf("%d", &ano);
   if ((ano % 4 == 0) && (ano % 100 != 0)) {
     bissexto = 1;
   } else if (ano % 400 == 0) {
     bissexto = 1;
   } else {
     bissexto = 0;
   }
   if (ano < 0 || mes < 1 || mes > 12) {
     dataValida = 0;
   } else {
     if (mes == 2) {
       if (bissexto)
         maiorDia = 29;
       else
         maiorDia = 28;
     } else if (mes == 4 || mes == 6 || mes ==
9 || mes == 11) {
         maiorDia = 30;
     } else {
       maiorDia = 31;
     }
     if (dia < 1 || dia > maiorDia) {
       dataValida = 0;
     } else {
       dataValida = 1;
     }
   }
```

https://repl.it/@flaviusgorgonio/ValidaDataSemFuncoesc

Validando a leitura de uma data (sem funções)

```
   if (!dataValida) {
     printf("A data %02d/%02d/%d não é
válida\n", dia, mes, ano);
     printf("Tente novamente!!!\n\n");
   }
 } while (!dataValida);

 printf("A data %02d/%02d/%d é válida\n", dia,
mes, ano);

 return 0;
}
```

1. O trecho de código localizado dentro da estrutura

   ```
   do {
       // cerca de 40 linhas de código
   } while (!dataValida);
   ```

   executa a leitura e validação de UMA ÚNICA data de nascimento

2. Se for necessário ler outras datas ao longo do programa, o trecho deverá ser repetido e isso gera, obviamente, repetição desnecessária de código
3. Se for necessário corrigir um bug, por exemplo, haverá maiores dificuldade de manutenção

https://repl.it/@flaviusgorgonio/ValidaDataSemFuncoesc

Validando a leitura de uma data (com funções)

1. Alternativamente, é possível usar um módulo em separado (uma função) para realizar a validação das datas
2. Uma vantagem óbvia é que reduz-se a quantidade de código repetido a cada nova leitura de uma data
3. A manutenção do programa também se torna mais simples, pois modificações são feitas em um único trecho de código
4. Além disso, funções armazenadas em bibliotecas podem ser reaproveitadas em novos programas

```
int main(void) {
 int dia, mes, ano;
 int dataValida;
 printf("Programa Data de Nascimento\n\n");
 do {
   printf("Informe sua data de nascimento\n");
   printf("Dia: ");
   scanf("%d", &dia);
   printf("Mês: ");
   scanf("%d", &mes);
   printf("Ano: ");
   scanf("%d", &ano);
   dataValida = testaData(dia, mes, ano);
   if (!dataValida) {
     printf("%02d/%02d/%d é inválida\n",dia,mes,ano);
     printf("Tente novamente!!!\n\n");
   }
 } while (!dataValida);
 printf("A data %02d/%02d/%d é válida\n",dia,mes,ano);
}
```

https://repl.it/@flaviusgorgonio/ValidaDataComFuncoesc

Validando a leitura de uma data (com funções)

```c
#include <stdio.h>

int testaData(int, int, int);
int bissexto(int);

int main(void) {
/// Função main aqui
}

int bissexto(int aa) {
 if ((aa % 4 == 0) && (aa % 100 != 0)) {
   return 1;
 } else if (aa % 400 == 0) {
   return 1;
 } else {
   return 0;
 }
}
```

```c
int testaData(int dd, int mm, int aa) {
 int maiorDia;
 if (aa < 0 || mm < 1 || mm > 12)
   return 0;
 if (mm == 2) {
   if (bissexto(aa))
     maiorDia = 29;
   else
     maiorDia = 28;
 } else if (mm == 4 || mm == 6 ||
            mm == 9 || mm == 11) {
              maiorDia = 30;
 } else
   maiorDia = 31;
 if (dd < 1 || dd > maiorDia)
   return 0;
 return 1;
}
```

https://repl.it/@flaviusgorgonio/ValidaDataComFuncoesc

Criação de bibliotecas

- A ideia de reutilização de código pressupõe a criação de código-fonte que possam ser usado novamente
- Alguns exemplos dessa prática inclui bibliotecas de validação de dados
- Até mesmo módulos completos podem ser reaproveitados em outros sistemas
- Essa é uma prática bastante comum no paradigma da Programação Orientada a Objetos (POO), através do conceito de herança
- Entretanto, o paradigma da Programação Imperativa também pode obter vantagens com a sua utilização

Implementação das funções (funcoes.c)

```c
/* Implementação da função Fatorial */
/* Calcula o fatorial de um valor int */
int fat(int n)
{
  int i, f;

  f = 1;
  for (i = 1; i <= n; i++)
    f *= i;
  return f;
}

/* Implementação da função Arranjo */
/* Calcula o arranjo de n valores tomados k a k
*/
int arr(int n, int k)
{
  return (fat(n) / fat(n - k));
}
```

```c
/* Implementação da função Combinação */
/* Calcula a combinação de n valores tomados k
a k */
int comb(int n, int k)
{
  return (fat(n) / ((fat(k) * fat(n - k))));
}
```

Implementação das assinaturas (funcoes.h)

```c
/* Funcoes para Analise Combinatoria */

/* Funcao Fatorial */
/* Calcula o fatorial de um valor int */
int fat(int);

/* Funcao Arranjo */
/* Calcula o arranjo de n valores tomados k a k
*/
int arr(int, int);

/* Funcao Combinacao */
/* Calcula a combinacao de n valores tomados k
a k */
int comb(int, int);
```

Implementação do programa principal (main.c)

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

int main(void)
{
  int n, k, combnk, arrnk, fatn;

  printf("Calcula n!, A(n,k) e C(n,k)\n\n");
  printf("Informe o valor de n: ");
  scanf("%d", &n);
  printf("Informe o valor de k: ");
  scanf("%d", &k);
  fatn = fat(n);
  combnk = comb(n, k);
  arrnk = arr(n, k);
  printf("%d! = %d\n", n, fatn);
  printf("A(%d,%d) = %d\n", n, k, arrnk);
  printf("C(%d,%d) = %d\n", n, k, combnk);
  return 0;
}
```

```
=== Procedimento para compilação em separado ===

>> gcc -c funcoes.c

É gerado o arquivo funcoes.o

>> gcc -c main.c

É gerado o arquivo main.o

>> gcc -o programa funcoes.o main.o

É gerado o arquivo executável programa

===== Outra alternativa =====

>> gcc -c funcoes.c

>> gcc -o programa funcoes.o prog.c
```

Atividade prática

1. Crie uma biblioteca de funções
   a. Você pode escolher um domínio específico, alguns exemplos:
      i. Funções de validação de tempo: validar data, hora, calcular diferença de datas, etc
      ii. Validação de dados pessoais: nome, e-mail, CPF, RG, matrícula, etc
      iii. Validações comerciais: agência bancária, conta corrente, código de barras, etc
2. Crie um programa que importa e utiliza a sua biblioteca
   a. Coloque o arquivo com as funções da sua biblioteca (por exemplo: mylibrary.c) na mesma pasta do programa principal
   b. Crie um arquivo apenas com as assinaturas (por exemplo: mylibrary.h) e coloque-o na mesma pasta
   c. Use um #include "mylibrary.h" no programa principal para incluir a sua biblioteca
   d. Utilize as instruções do slide anterior para efetuar a compilação em separado
3. Analise as funções criadas da sua biblioteca
   a. As funções são genéricas o suficiente para serem utilizadas em outros programas?
   b. As funções estão documentadas e comentadas o suficiente para serem utilizadas por outros programadores em outros projetos?
   c. Compartilhe a sua biblioteca (arquivos mylibrary.c e mylibrary.h) com outros colegas

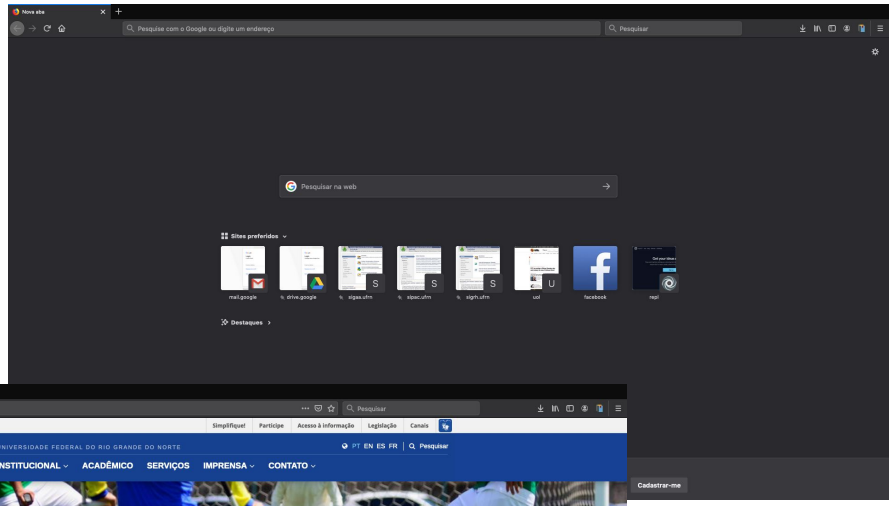
Passagem de parâmetros para o programa

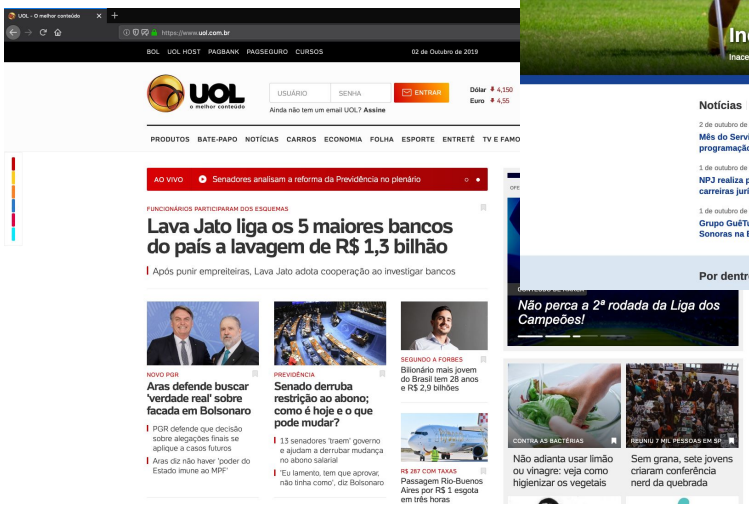
Experimente executar os seguintes comandos a partir do prompt de comandos do seu sistema operacional:

```
ufrn~$ firefox

ufrn~$ firefox www.ufrn.br

ufrn~$ firefox www.uol.com.br
```





Parâmetros para a função main()

```c
#include <stdio.h>

int main(int argc, char** argv) {
  printf("Hello Parâmetros\n");
  printf("Eu recebi %d palavras como parâmetros\n", argc);
  for (int i = 0; i < argc; i++) {
    printf("Parâmetro %d: %s\n", i, argv[i]);
  }
  return 0;
}
```

argc recebe a quantidade de parâmetros, incluindo o nome do programa

argv é um vetor de apontadores para um conjunto de palavras (parâmetros)

O programa deve tratar os parâmetros adequadamente

Bibliografia Recomendada

Celes, W. *et al*. Introdução a Estruturas de Dados com Técnicas de Programação em C, 2ª ed. Rio de Janeiro: Elsevier, 2016.

Sebesta, R. W., Conceitos de Linguagens de Programação, 9ª ed. Porto Alegre: Bookman, 2016.

Varejão, F., Linguagens de Programação: Conceitos e Técnicas, Rio de Janeiro: Campus, 2004.