Funções

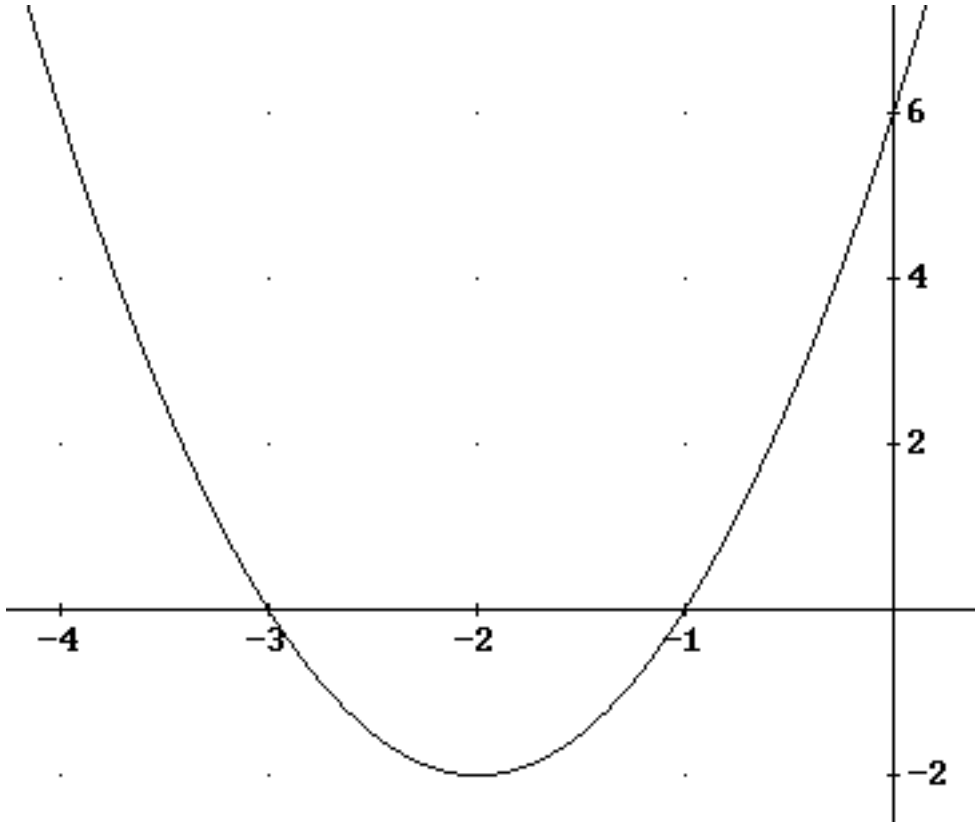
Funções

- Funções definem operações que são usadas frequentemente

- Funções, na matemática, requerem parâmetros de entrada, e definem um valor de saída

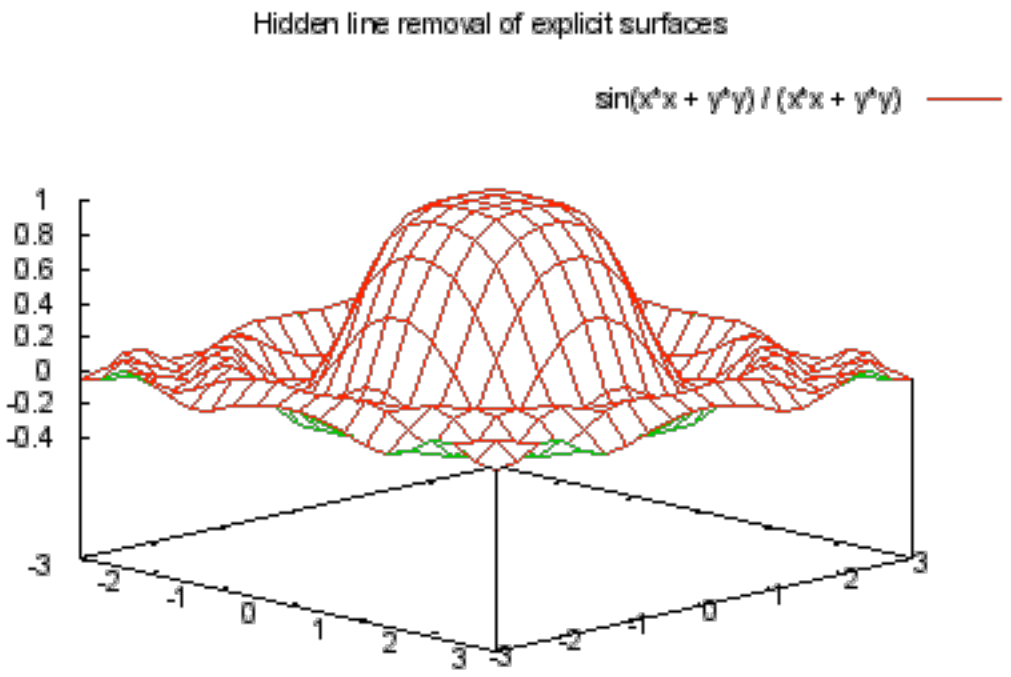
Funções - Exemplos

- Função quadrática $y = ax^2+bx+c$
  - Entrada: x
  - Saída: y



Funções - Exemplos

- Função $z = \sin(x^2+y^2)/(x^2+y^2)$
  - Entrada: x, y
  - Saída: z



Funções na programação

- Em linguagens imperativas, TODOS os programas usam funções

- No C, o programa SEMPRE começa executando a função **main**.

```
#include <stdio.h>

int main(void)
{
    puts("Olá, Mundo!");
    return 0;
}
```

Funções

- Usamos funções para evitar de escrever várias vezes o mesmo código
  - Código que será executado várias vezes em um programa, mas com valores diferentes
  - Operações comuns a um ou mais programas

Funções na programação

- Nas linguagens de programação, as funções possuem:
  - Zero ou mais parâmetros de entrada, com ou sem os seus tipos
  - **Zero** ou mais parâmetros de saída, com ou sem os seus tipos
  - Código a ser executado pela função

Funções - C

- Em C, definimos a função por:
  - Zero ou mais parâmetros de entrada, com os seus tipos
  - Um parâmetro de saída com tipo, sendo que o tipo pode ser “sem saída”
  - Código da função
- Ao chamarmos a função, devemos passar valores para TODOS os parâmetros, sem exceção (não é o caso no C++…)

Funções - C

- Declarando uma função

```
1.tipo nome(tipo par1, tipo
  par2, … )
2.{
3.    …
4.    …
5.    return VALOR;
6.}
```

Funções - C

- **tipo** define como será o retorno da função:
  - Inteiro: int
  - Caractere: char
  - Real: float, double
  - ….
- Devemos indicar o valor de saída (guardado em uma variável, ou uma constante) usando o comando **return**.
- Funções podem não retornar saída: **void**

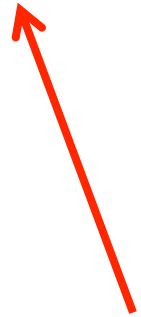
C - Exemplo

```
1.double logistica(double x) {
2.    return 1.0/
  (1.0+exp(-1.0*x));
3.}
```

Nome da função

C - Exemplo

```
1.double logistica(double x) {
2.    return 1.0/
  (1.0+exp(-1.0*x));
3.}
```



Variável de entrada

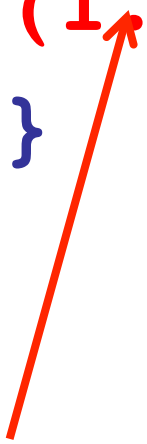
C - Exemplo

```
1.double logistica(double x) {
2.    return 1.0/
   (1.0+exp(-1.0*x));
3.}
```

Saída do tipo double

C - Exemplo

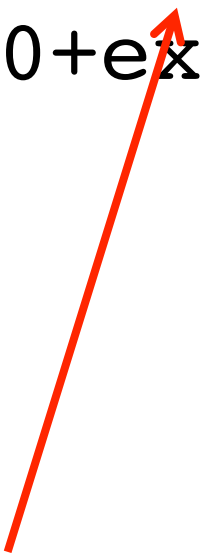
```
1.double logistica(double x) {
2.    return 1.0/
   (1.0+exp(-1.0*x));
3.}
```



Código a ser executado: marcação de início e fim de bloco usando chaves

C - Exemplo

```
1.double logistica(double x) {
2.    return 1.0/
   (1.0+exp(-1.0*x));
3.}
```



Saída: “Retorne” o valor da expressão a seguir

C – Exemplo: usando funções

```
1.double logistica(double x) {
2.    return 1.0/
  (1.0+exp(-1.0*x));
3.}
4.
5.int main() {
6. double entrada = 10.0;
7. double saida =
  logistica(entrada);
8.}
```

Variável saida recebe o valor da função

Exemplo 2

```
int sep (int v[], int p, int r) {
   int w[1000], i = p, j = r, c = v[p], k;
   for (k = p+1; k <= r; ++k)
      if (v[k] <= c) w[i++] = v[k];
      else w[j--] = v[k];
   // agora i == j
   w[i] = c;
   for (k = p; k <= r; ++k) v[k] = w[k];
   return i;
}
```

Função sep: três parâmetros, retorna um inteiro

Exemplo 3 – usando o main

- A função main é especial:
  - É a primeira a ser chamada no programa
    - Todo programa tem um!
  - Seu retorno indica se o programa executou corretamente (retorno 0) ou não (retorno != 0)
  - Seus parâmetros, quando existem, são os parâmetros passados para o programa quando foi executado

Exemplo 3 – usando o main

```
1. #include <stdio.h>
2.
3. int main(int argc, char **argv) {
4.   int i;
5.   for(i=0;i<argc;i++) {
6.     printf("%s",argv[i]);
7.   }
8.   return 0;
9. }
```

Exemplo 3 – usando o main

- O programa imprime todos os argumentos recebidos pelo main
- Útil para mudar o funcionamento do seu programa passando parâmetros

Funções na programação

- Em linguagens de programação, funções podem não ter parâmetros de entrada ou de saída: o importante é o **efeito da execução da função**

Funções sem retorno – C

- Funções sem retorno devem ter o tipo de retorno **void**.
- Exemplo: função para imprimir mensagem de boas-vindas do programa

```
1. void saudacao() {
2.   printf("Ola usuario! Digite o comando que quer
   executar, ou ? para ajuda.");
3. }
4. int main() {
5.   saudacao();
6.   …
7.   return 0;
8. }
```

Porque não retornar valor?

- Porque o importante pode ser a ação **colateral** da função, e não o seu valor de saída:
  - Impressão de uma mensagem
  - Ligar/desligar um componente do hardware
  - …

Funções: escopo de variáveis

- Variáveis podem ser acessadas somente dentro do seu escopo
- No C, o escopo é definido do momento da declaração até o fim do bloco
- No C, uma variável declarada dentro de um bloco de laço vive somente uma iteração do laço

Funções: escopo de variáveis

```
int teste(int x) {
…
}

int main() {
int y;
  for(int i=0;i<10;i++) {
      if(i < 5) {
         int a;
      } else {
        int b;
      }
  }
}
```

Funções: escopo de variáveis

```
int teste(int x) {
…
}
```

Escopo de x

```
int main() {
int y;
  for(int i=0;i<10;i++) {
      if(i < 5) {
         int a;
      } else {
        int b;
      }
  }
}
```

Funções: escopo de variáveis

```
int teste(int x) {
…
}

int main() {
int y;
  for(int i=0;i<10;i++) {
      if(i < 5) {
         int a;
      } else {
        int b;
      }
  }
}
```

Escopo de y

Funções: escopo de variáveis

```
int teste(int x) {
…
}

int main() {
int y;
  for(int i=0;i<10;i++) {
      if(i < 5) {
         int a;
      } else {
        int b;
      }
  }
}
```

Escopo de i

Funções: escopo de variáveis

```
int teste(int x) {
…
}

int main() {
int y;
  for(int i=0;i<10;i++) {
      if(i < 5) {
         int a;
      } else {
        int b;
      }
  }
}
```

Escopo de a

Funções: escopo de variáveis

```
int teste(int x) {
…
}

int main() {
int y;
  for(int i=0;i<10;i++) {
      if(i < 5) {
         int a;
      } else {
        int b;
      }
  }
}
```

Escopo de b

Funções e escopo

- As variáveis locais (variáveis de uma função) são armazenadas em um modelo de pilha:
  - Cada nova variável criada é adicionada ao topo da pilha
  - Ao terminar o bloco, eliminamos todas as variáveis daquele bloco da pilha

  – O mesmo vale para chamada de funções

Módulos

Módulo

- Um módulo é uma forma de organizar um programa grande

- Dividimos o programa em módulos, onde cada um deles possui um conjunto de tarefas bem específico:
  - Módulo de entrada/saída
  - Módulo de gerenciamento de memória
  - Módulo de cálculo
  - ….

Módulo

- Módulos terão uma ou mais funções, que desta forma realizam operações similares.
  - Módulo de operações matemáticas
  - Módulo de operações sobre o tempo
  - Módulo para entrada e saída

Exemplos de módulos em C

- Módulo de operações matemáticas (log, pow, sqrt, …)
- Módulo de operações sobre o tempo (gettimeofday, localtime, ….)
- Módulo para entrada e saída (printf, scanf, gets, getchar, …)

Módulos e bibliotecas

- Módulos muito úteis podem ser empacotados em **bibliotecas**, para que possam ser utilizados em outros programas

Módulos e bibliotecas - C

- Em C, carregamos módulos e bibliotecas com o comando #include

1. #include<stdio.h>
2. #include<math.h>
3. #include “meumodulo.h”

- O uso de <> ou “ ” depende da localização do módulo/ biblioteca
  - No diretório de bibliotecas do sistema: <>
  - Em outro lugar (por exemplo, no diretório onde está o meu programa) : “ “

C - Definindo um módulo

- O módulo consiste em:
  - Arquivo de cabeçalhos de funções e declaração de tipos de dados (extensão .h)
  - Arquivo com o código das funções (extensão .c)

C - Definindo um módulo

Arquivo simples.h

```
1.double media (double
  a, double b);
2.
3. double dif(double
  a, double b);
4.
```

Arquivo simples.c

```
1.#include “simples.h”
2.
3.double media(double
  a, double b) {
4.  return (a+b)/2;
5.}
6.
7.double dif(double a,
  double b) {
8.  return a – b;
9.}
```

Bibliotecas padrão do C

- Muitas funções comuns:
  - stdio.h – Entrada e saída
  - Math.h – Funções matemáticas mais complexas
  - stdlib.h – gerenciamento do programa: alocar memória, sair, …
  - sys/time.h – Gerenciar o tempo: imprimir datas, ver a hora/data atual…

Bibliotecas padrão do C

- Podemos encontrar a lista de funções em manuais, livros e em sites Web, i.e.:
  - http://www.acm.uiuc.edu/webmonkeys/book/c_guide/

Bibliotecas padrão do C – Math.h

- A math.h é especial: precisa de um parâmetro na compilação

- gcc codigo.c **-lm** -o programa

- **-l”nome”** indica que queremos que o programa “incorpore” código de um módulo externo.
- TODA biblioteca precisa do –l”nome”, EXCETO as funções padrão do C