ROTEIRO

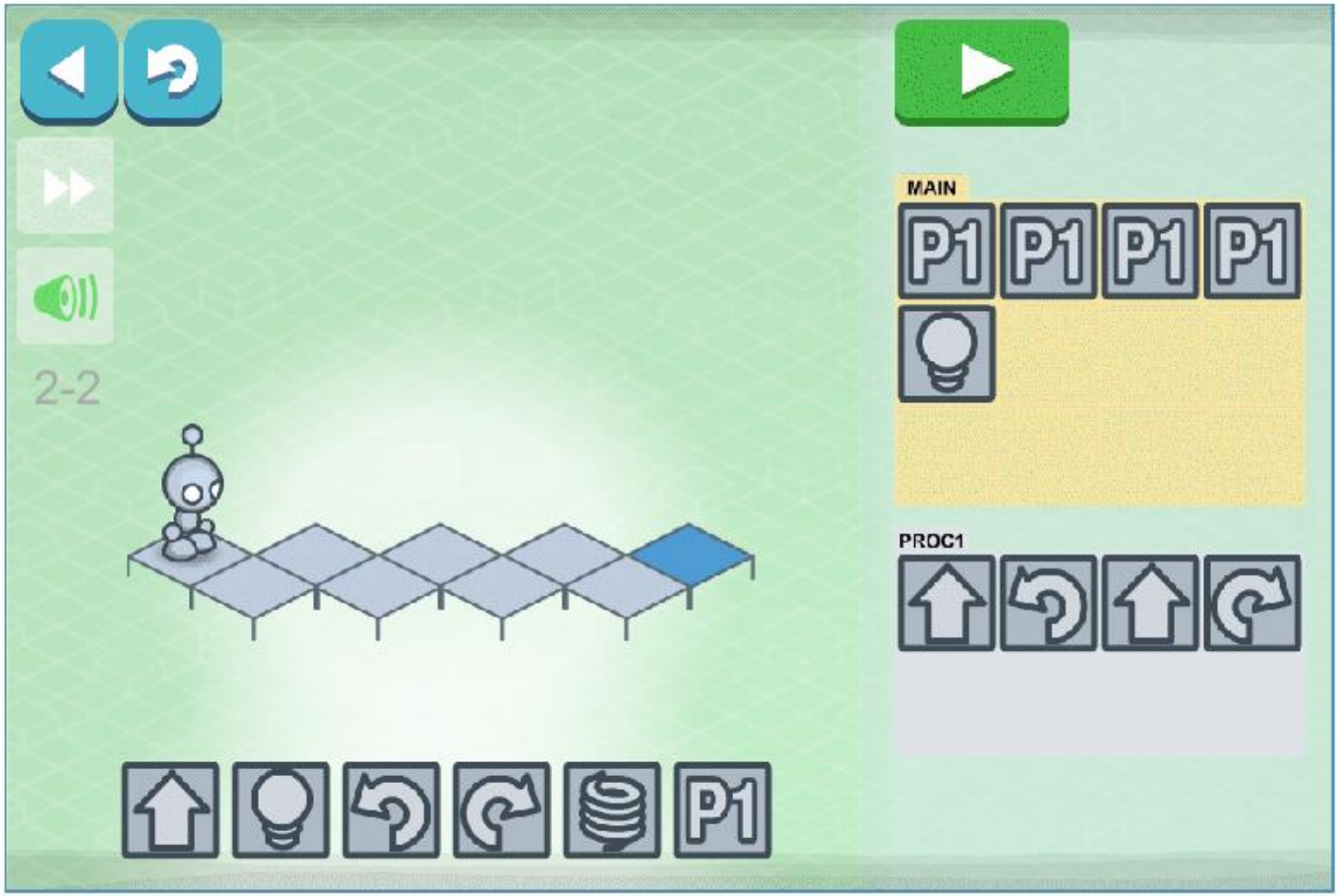
1. Procedimentos
2. Funções

Baseado em:

- http://www.ic.unicamp.br/~islene/mc102/aula14/

MOTIVAÇÃO

Identificar um subproblema (que se repete) e descrever um subalgoritmo para representá-lo



MOTIVAÇÃO

Quando queremos resolver um problema geralmente dividimos ele em subproblemas.

- Ler *n* velocidades, encontrar o maior e menor velocidade, a mediana e o desvio padrão da sequência
  - Subproblema #1: Ler *n* velocidades
  - Subproblema #2: Ordenar o vetor
  - Subproblema #3: Encontrar maior
  - Subproblema #4: Encontrar menor
  - Subproblema #5: Encontrar mediana
  - Subproblema #6: Calcular o desvio padrão

SUBPROBLEMAS

- A linguagem C permite descrever blocos de instruções para auxiliar na descrição desses **sub-algoritmos**
- Possibilita exportar o bloco para ser chamado por outras partes do programa
- Idéia central das bibliotecas

PROCEDIMENTOS

- Procedimentos são estruturas que agrupam um conjunto de comandos, que são executados quando o procedimento é chamado.
  - **Não retorna um valor!**
- Todo procedimento tem um **nome** e é delimitado por *colchetes* ( )
- Os **parâmetros** de entrada do procedimento estão dentro dos colchetes e são delimitados por vírgula

Exemplos:

- `scanf("%d", &x);`
- `printf("%d", x);`
- `gets(y);`
- `puts(y);`

FUNÇÕES

- Funções são procedimentos que retornam um único valor ao final de sua execução.
  - Tem que retornar um valor!
- Toda função tem um nome e é delimitado por *colchetes* ( )
- Os parâmetros de entrada da função estão dentro dos colchetes e são delimitados por vírgula

Exemplos:

- `x = strlen(c);`
- `y = strcpm(a, b);`

POR QUE UTILIZAR PROC/FUNÇÕES

- Permitir o **reaproveitamento** de código já construído (por você ou por outros programadores);
- **Suproblemas** – Evitar que um trecho de código seja repetido várias vezes dentro de um mesmo programa;
- Permitir a **alteração** ou substituição de um trecho de código de uma forma mais rápida. Com o uso de uma função é preciso alterar apenas dentro da função que se deseja;

PROC/FUNÇÕES EM C

- **Todo programa C é baseado em procedimentos e funções**
- A função **main( )** indica o bloco principal de código a ser executado pelo programa
- Já utilizamos procedimentos e funções: **scanf, printf, gets, puts, strlen, strcmp**

ESTRUTURA DE UM PROCEDIMENTO

void = nulo

parâmetros

chaves

```
void nome_funcao(tipo p1, tipo p2, ..., tipo pn)
{
    // comandos                    // Corpo do Procedimento
}
```

ESTRUTURA DE UM PROCEDIMENTO

void = nulo

parâmetros

chaves

```
void nome_funcao(tipo p1, tipo p2, ..., tipo pn)
{
    // comandos                    // Corpo do Procedimento
}
```

```
void imprimir_logo(int turma)
{
    printf("Engenharia de Transportes¥n");
    printf("Turma: %d¥n", turma);
}
```

ESTRUTURA DE UM PROCEDIMENTO

```
#include <stdio.h>

void imprimir_logo(int turma)
{
    printf("Engenharia de Transportes¥n");
    printf("Turma: %d¥n", turma);
}


int main()
{
    imprimir_logo(2);
    imprimir_logo(3);
}
```

parâmetros

// Principal

ESTRUTURA DE UMA FUNÇÃO

parâmetros

chaves

```
tipo_retorno nome_funcao(tipo p1, tipo p2, ..., tipo pn)
{
    // comandos
    return valor_de_retorno;
}
```

// Corpo da Função

retorno

ESTRUTURA DE UMA FUNÇÃO

parâmetros

chaves

```
tipo_retorno nome_funcao(tipo p1, tipo p2, ..., tipo pn)
{
    // comandos
    return valor_de_retorno;
}
```

// Corpo da Função

retorno

```
float maior(float a, float b)
{
    if (a > b) {
        return a;
    } else {
        return b;
    }
}
```

ESTRUTURA DE UMA FUNÇÃO

```
#include <stdio.h>

float maior(float a, float b)
{
    if (a > b) {
        return a;
    } else {
        return b;
    }
}

int main()
{
   float m = maior(2, 3);
   printf("%f", m);
}
```

ESTRUTURA DE UMA FUNÇÃO

parâmetros

```
float maiorVelocidade(float v[10])
{
    float maior = v[0];
    int i = 1;
    while (i < 10) {
        if (v[i] > maior) {
            maior = v[i];
            i++;
        }
    }
    return maior;
}
```

chaves

retorno

ESTRUTURA DE UMA FUNÇÃO

parâmetros

chaves

```
int det2(int matriz[2][2])
{
    int dp = matriz[0][0] * matriz[1][1];
    int ds = matriz[0][1] * matriz[1][0];
    int det = dp - ds;
    return det;
}
```

retorno

CORPO DE UM PROC/FUNÇÕES

- Toda função deve ser declarada antes da função main para ser utilizada
- Toda função deve ter um tipo. Esse tipo determina qual será o tipo de seu valor de retorno.
- Os parâmetros de uma função determinam qual será o seu comportamento (semelhante a uma função matemática).

$$f(x) = x^2 + x$$

```
float f(float x)
{
   int y = (x * x) + x
     return y;
}
```

CORPO DE UM PROC/FUNÇÕES

- Toda função deve ser declarada antes da função main para ser utilizada
- Toda função deve ter um tipo. Esse tipo determina qual será o tipo de seu valor de retorno.
- Os parâmetros de uma função determinam qual será o seu comportamento (semelhante a uma função matemática).

$$f(x) = x^2 + x$$

```
float f(float x)
{
   return (x * x) + x
}
```

```c
#include <stdio.h>

int main()
{
    int a[2][2] = {{10, 20}, {-1, 2}};
    int det = det2(a);
    printf("%d", det);
    return 0;
}

int det2(int matriz[2][2])
{
    int dp = matriz[0][0] * matriz[1][1];
    int ds = matriz[0][1] * matriz[1][0];
    int det = dp - ds;
    return det;
}
```

```c
#include <stdio.h>

int det2(int matriz[2][2]);

int main()
{
    int a[2][2] = {{10, 20}, {-1, 2}};
    int det = det2(a);
    printf("%d", det);
    return 0;
}

int det2(int matriz[2][2])
{
    int dp = matriz[0][0] * matriz[1][1];
    int ds = matriz[0][1] * matriz[1][0];
    int det = dp - ds;
    return det;
}
```

protótipo

CORPO DE UM PROC/FUNÇÕES

- O corpo do proc/função especifica quais comandos ela irá executar quando for chamada.
  (geralmente resolver um subproblema)

- Ele processa os parâmetros (se houver), realiza outras tarefas e gera saídas (se necessário)
  - Parâmetros são a entrada do proc/função

- Similar a função main( )

VARIÁVEIS EM PROC/FUNÇÕES

- Variáveis passadas como parâmetros indicam quais são os valores com os quais a função irá trabalhar.
- Esses valores são copiados para os parâmetros da função, que pode manipulá-los.
- Os parâmetros passados pela função não necessariamente possuem os mesmos nomes que os parâmetros que a função espera.
- Esses parâmetros serão mantidos intocados durante a execução da função

```c
#include <stdio.h>

int det2(int matriz[2][2])
{
    int dp = matriz[0][0] * matriz[1][1];
    int ds = matriz[0][1] * matriz[1][0];
    int det = dp - ds;
    return det;
}

int main()
{
    int a[2][2] = {{10, 20}, {-1, 2}};
    int det = det2(a);
    printf("%d", det);
    return 0;
}
```

```c
#include <stdio.h>

int incrementa(int x)
{
    x = x + 1;
    return x;
}

int main()
{
    int x = 50;
    incrementa(x);
    printf("%d", x);
    return 0;
}
```

VARIÁVEIS EM PROC/FUNÇÕES

- Um procedimento/função pode não ter parâmetros

- Isso é expressado utilizando a palavra void ou omitindo os parâmetros
  - Todo procedimento possui como retorno o tipo void

```
void imprimir_logo(void)
{
    printf("Universidade Federal de Goiás¥n");
    printf("Engenharia de Transportes¥n");
}
```

VARIÁVEIS EM PROC/FUNÇÕES

- Um procedimento/função pode não ter parâmetros

- Isso é expressado utilizando a palavra void ou omitindo os parâmetros
  - Todo procedimento possui como retorno o tipo void

```
void imprimir_logo()
{
    printf("Universidade Federal de Goiás¥n");
    printf("Engenharia de Transportes¥n");
}
```

VARIÁVEIS LOCAIS/GLOBAIS

- Uma variável é chamada local se ela foi declarada dentro de uma função. Nesse caso, ela existe somente dentro daquela função e após o término da execução da mesma, a variável deixa de existir.
- Uma variável é chamada global se ela for declarada fora de qualquer função (ou seja, no mesmo lugar onde funções, procedimentos e constantes são declarados).
  - Essa variável existe dentro de todas as funções e qualquer procedimento ou função pode alterá-la

```c
#include <stdio.h>

float velmax = 60;

int acima(float v)
{
    int r = 0;
    if (v > velmax) {
        r = 1;
    }
    return r;
}

int main()
{
    velmax = 80;
    printf("%d¥n", acima(70));

    velmax = 60;
    printf("%d¥n", acima(70));
}
```

```
int pinoLED = 8;

void setup()
{
    pinMode(pinoLED, OUTPUT);
}

void loop()
{
    digitalWrite(pinoLED, HIGH);
    delay(1000);

    digitalWrite(pinoLED, LOW);
    delay(1000);
}
```

```c
#include <stdio.h>
int a = 0;
int incrementa(int x)
{
    int a = 2;
    x = x + a;
    return x;
}

int main()
{
    a = 50;
    int x = incrementa(x);
    printf("%d", x);
    return 0;
}
```

CHAMANDO OUTRAS FUNÇÕES

- Pode-se invocar um procedimento ou função dentro de outro procedimento ou função.
- Já fazemos isso: printf, scanf, strlen, ...
- Exemplo:

$f(x) = x^2 + x$

$g(x) = 2x$

$f(g(x))$

```
int g(int x)
{
   return x * x;
}

int f(int x)
{
    int rgx = g(x)
    return (rgx * rgx) + rgx
}
```

CHAMANDO OUTRAS FUNÇÕES

- A recursão também é chamada de definição circular.
- Ela ocorre quando algo é definido em termos de si mesmo.
- Um exemplo clássico de função que usa recursão é o cálculo do fatorial de um número:
  - $!n = n \times (n-1)!$
  - Em que $0! = 1$

```c
#include <stdio.h>

int fatorial(int n)
{
    if (n == 0) {
        return 1;
    } else {
        return n * fatorial(n - 1);
    }
}

int main()
{
    int nf = fatorial(5);
    printf("%d", nf);
    return 0;
}
```

BIBLIOTECA <MATH.H>

Função	Descrição
sqrt(x)	Retorna o quadrado de x
pow(x, a)	Retorna x^n
exp(x)	Retorna e^x
log(x)	Retorna $\ln x$
log10(x)	Retorna $\log_{10} x$
sin(x)	Retorna o seno de x radianos
cos(x)	Retorna o cosseno de x radianos
tan(x)	Retorna a tangente de x radianos