Algoritmia e Programação

Linguagem C: conceitos básicos

Estrutura básica de um programa em C

```
#include <stdio.h>
#include "my_header.h"

#define PI (3.1415926)

int main()
{
   float raio, perimetro;

   printf("Raio: ");
   scanf("%f", &raio);
   perimetro = 2 * PI * raio;
   printf("Perímetro: %.3f\n", perimetro);

   return 0;
}
```

Estrutura básica de um programa em C

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

#define PI (3.1415926)

int main()
{
   float raio, perimetro;

   printf("Raio: ");
   scanf("%f", &raio);
   perimetro = 2 * PI * raio;
   printf("Perímetro: %.3f\n", perimetro);

   return 0;
}
```

Inclusão de ficheiros de cabeçalho (header).
Estes ficheiros contêm definições de tipos e declarações de funções e variáveis externas que podem ser adicionadas ao nosso programa.
<...> : indica ficheiro fornecido pelo compilador
"..." : indica ficheiro do projecto

Estrutura básica de um programa em C

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

#define PI (3.1415926)

int main()
{
   float raio, perimetro;

   printf("Raio: ");
   scanf("%f", &raio);
   perimetro = 2 * PI * raio;
   printf("Perímetro: %.3f\n", perimetro);

   return 0;
}
```

Definição de constantes simbólicas.
Definição de símbolos que podem ser utilizados ao longo do código, para representar um dado valor.
Permitem um único ponto para definir (e alterar) um valor em todo o código do programa.

Estrutura básica de um programa em C

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

#define PI (3.1415926)

int main()
{
    float raio, perimetro;

    printf("Raio: ");
    scanf("%f", &raio);
    perimetro = 2 * PI * raio;
    printf("Perímetro: %.3f\n", perimetro);

    return 0;
}
```

Função principal do programa.
As funções são identificadas por um nome (único em todo o projecto).
A função **`main()`** é, por definição, a função que inicia um programa. Também por definição, retorna um valor inteiro.
As instruções de uma função são delimitadas por um par de chavetas.

Estrutura básica de um programa em C

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

#define PI (3.1415926)

int main()
{
   float raio, perimetro;

   printf("Raio: ");
   scanf("%f", &raio);
   perimetro = 2 * PI * raio;
   printf("Perímetro: %.3f\n", perimetro);

   return 0;
}
```

Declaração de variáveis.
As variáveis são estruturas que permitem guardar dados.
Uma variável é identificada por um nome epode guardar um valor de um determinado tipo, indicado à esquerda na declaração.
Uma variável é visível somente na função em que é declarada.

Estrutura básica de um programa em C

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

#define PI 3.1415926

int main()
{
    float raio, perimetro;

    printf("Raio: ");
    scanf("%f", &raio);
    perimetro = 2 * PI * raio;
    printf("Perímetro: %.3f\n", perimetro);

    return 0;
}
```

Bloco de instruções da função.
As instruções podem ser operações aritméticas, lógicas e de controlo de fluxo, bem como chamadas de funções que realizam tarefas mais complexas.

Porquê incluir código de bibliotecas e módulos?

- A linguagem C somente fornece instruções de controlo de fluxo e operações aritméticas e lógicas.

- Operações aparentemente simples como escrever um carácter no ecrã não são assim tão simples de implementar.

  - A linguagem C estabelece uma biblioteca de funcionalidades padrão: a ***C standard library*** (ou ***libc***).

  - A funcionalidade implementada na biblioteca pode ser incorporada nos programas, sem ser necessário conhecer os detalhes da sua implementação: apenas é necessário conhecer a sua ***Interface de Programação de Aplicações*** *(ou* ***application programming interface — API****)*.

Declaração da API da libc

- Desde a norma C11 (2011), a API da libc encontra-se declarada em 29 ficheiros de cabeçalho.

- Nesta fase inicial da aprendizagem, os mais relevantes são:
  - **`<stdio.h>`** : define as operações de entrada e saída;
  - **`<math.h>`** : define funções matemáticas comuns;
  - **`<ctype.h>`** : define funções sobre caracteres;
  - **`<stdlib.h>`** : define funções de conversão numérica e geração de números pseudo-aleatórios, entre outros.

<stdio.h> : printf

- `int printf ( const char * format, ... );`
- Escreve o texto apontado por `format`.
- Para além do texto que deve ser escrito, `format` pode conter **especificadores de formato** identificados com o carácter **%**, substituídos pelos valores nos argumentos seguintes ao `format`. Exemplos:
  - `%d` : inteiro decimal com sinal
  - `%u` : inteiro decimal sem sinal
  - `%f` : real decimal, com sinal
  - `%E` : real decimal em notação científica

<stdio.h> : printf

- Os especificadores de formato podem ainda conter sub-especificadores opcionais para definir a forma como os valores devem ser representados.

- Exemplos:
  - **`%.2f`** : apresenta um real com duas casas decimais;
  - **`%6.2f`** : apresenta um real com seis algarismos e duas casas decimais, preenchendo com espaços à esquerda se o valor for menor;
  - **`%06d`** : apresenta um inteiro com, pelo menos, 6 algarismos, preenchendo com zeros à esquerda se o valor for menor.

- Mais informação: http://www.cplusplus.com/reference/cstdio/printf/

<stdio.h> : scanf

- `int scanf ( const char * format, ... );`
- Lê dados de acordo com o especificado em `format`, e coloca-os nas variáveis apontadas nos argumentos adicionais.
- `format` contém **especificadores de formato** que indicam como os valores inseridos devem ser interpretados. Os especificadores são semelhantes aos da função `printf`. Exemplos:
  - `%d` : inteiro decimal com sinal
  - `%u` : inteiro decimal sem sinal
  - `%f` : real decimal, com sinal
  - `%E` : real decimal em notação científica
- Mais informação em: http://www.cplusplus.com/reference/cstdio/scanf/

Tipos de dados

- A linguagem C define um conjunto básico de tipos de dados:
    - **`char`** : um byte, capaz de suportar um carácter.
    - **`int`** : um inteiro, da dimensão natural para um inteiro da arquitectura do computador (pelo menos 16 bits).
    - **`float`** : real de vírgula flutuante, com precisão simples.
    - **`double`** : real de vírgula flutuante com precisão dupla.

Tipos de dados

- O tipo **`int`** pode ser alterado para:
    - **`short int`** : pelo menos 16 bits, e menor do que um `int`.
    - **`long int`** : pelo menos 32 bits, maior do que um `short int`.
- Cada compilador é livre de definir as dimensões destes tipos, de acordo com a arquitectura do processador e com as regras apresentadas.
- Também é possível **`long double`** para vírgula flutuante com precisão estendida.

Tipos de dados

- Os tipos `int` e `char` podem ainda ser alterados para:
    - **`unsigned int`** : inteiro sem sinal (não negativo)
    - **`unsigned char`** : carácter sem sinal (não negativo).
- O `unsigned` pode ser combinado com short e long:
    - **`unsigned short int`**
    - **`unsigned long int`**

Constantes

- É má prática a utilização de "*números mágicos*" no código fonte.
    - Dificulta a leitura e compreensão do código.
    - Difícil de alterar o código de forma sistemática.
- A solução mais apropriada é atribuir um nome a cada número mágico, numa linha **`#define`**. Exemplo

```
#define PI 3.1415926
```

    - O nome `PI` pode ser utilizado no resto do código em vez do valor.
- Os nomes de constantes simbólicas são escritos totalmente em maiúsculas, por convenção.

Declaração de variáveis

- Na linguagem C, todas as variáveis têm que ser declaradas, antes da sua primeira utilização.

- Uma declaração contém um tipo de dados, e uma lista de uma ou mais variáveis desse tipo. Exemplos:

```
int a, b, c;

float x, y, z;
```

- O nome de uma variável pode conter letras, algarismos e o carácter '_'.

- O primeiro carácter do nome de uma variável tem que ser uma letra, obrigatoriamente.

- Os nomes das variáveis são escritos em minúsculas, por convenção.

Operadores aritméticos

- Operadores binários (dois operandos):
  - **+** : soma (e.g. `z = x + y;`)
  - **–** : subtracção (e.g. `z = x - y;`)
  - `*` : multiplicação (e.g. `z = x * y;`)
  - `/` : divisão (e.g. `z = x / y;`)
  - `%` : resto da divisão inteira (e.g. `resto = x % y;`)
    - Exclusivo para operandos inteiros!

Operadores aritméticos

- Operadores unários (apenas um operando):
  - **+** : idêntico (e.g. `z = +100`)
  - **−** : simétrico (e.g. `z = -x`)

Operadores de incremento e decremento

- Operador de incremento (**++**) permite incrementar em uma unidade o valor de uma variável:
  - **y = x++** : retorna o valor de x e depois incrementa-o em uma unidade.
  - **y = ++x** : incrementa o valor de x e depois retorna o valor já incrementado.

Operadores de incremento e decremento

- Operador de decremento (**––**) permite decrementar em uma unidade o valor de uma variável:
  - **y = x––** : retorna o valor de x e depois decrementa-o em uma unidade.
  - **y = ––x** : decrementa o valor de x e depois retorna o valor já decrementado.

Operadores relacionais e lógicos

- Operadores relacionais devolvem um valor lógico — **verdadeiro** ou **falso**:
  - **>** : maior (e.g. *x > y*)
  - **>=** : maior ou igual (e.g. *x >= y*)
  - **<** : menor (e.g. *x < y*)
  - **<=** : menor ou igual (e.g. *x <= y*)
  - **==** : igual (e.g. *x == y*)
  - **!=** : diferente (e.g. *x != y*)

Operadores relacionais e lógicos

- Operadores lógicos combinam valores lógicos, devolvendo um valor lógico:
    - **!** : negação / NOT (e.g. `!x`)
        - Converte em **zero** (i.e. **falso**) um operando não-nulo.
        - Converte em **um** (i.e. **verdadeiro**) um operando nulo.
    - **&&** : e / AND (e.g. `x == 0 && y != 0`)
    - **||** : ou / OR (e.g. `x == 0 || y > 0`)

Operadores de atribuição

- Os operadores de atribuição permitem atribuir o valor à sua direita para a variável à sua esquerda:
  - **=** : atribuição simples (e.g. `x = y * 4;`)

Operadores de atribuição

- Os operadores de atribuição aritmética realizam a operação aritmética com os valores à sua esquerda e à sua direita, atribuindo o resultado à variável à esquerda:
  - **+=** : e.g. `x += 4` (equivale a `x = x + 4;`)
  - **-=** : e.g. `x -= 4` (equivale a `x = x - 4;`)
  - ***=** : e.g. `x *= 4` (equivale a `x = x * 4;`)
  - **/=** : e.g. `x /= 4` (equivale a `x = x / 4;`)
  - **%=** : e.g. `x %= 4` (equivale a `x = x % 4;`)

Prioridades dos operadores

- A linguagem C estabelece regras de prioridade e associação na execução dos operadores quando combinados numa expressão:
  - `!` `++` `--` `+` (unário) `-` (unário)
  - `*` `/` `%`
  - `+` `-`
  - `<` `<=` `>` `>=`
  - `==` `!=`
  - `&&`
  - `||`
  - `=` `+=` `-=` `*=` `/=` `%=`