PRA – Projeto de Arquivos

Revisão da linguagem C

Prof. Allan Rodrigo Leite

Organização básica de um computador

Processador

Memória principal

Canal de comunicação

Memória secundária

Dispositivos de entrada

Dispositivos de saída

Linguagens de programação

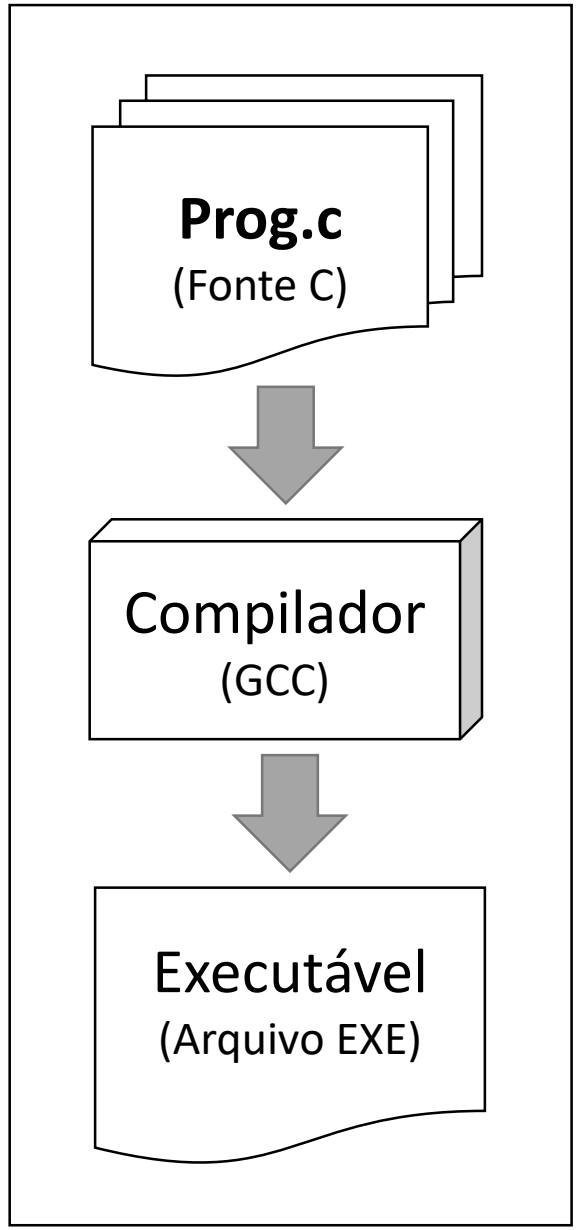
- Um grande avanço ocorreu na computação quando surgiram programas que traduziam instruções para **linguagem de máquina**
  - Primeira linguagem de programação surgiu na década de 50

- Instruções em **linguagens de programação** (**linguagens de alto nível**) são escritas de forma muito mais clara e legível para o programador

Compilador

- Nesta disciplina
  - Linguagem de programação: C
  - Código fonte: arquivos com extensão .c
  - Tradutor: compilador C (GCC)
    - Traduzir → compilar

- Para que o processo de tradução seja possível, é necessário que o compilador consiga identificar cada instrução no código fonte
  - Assim, o programador precisa seguir uma série de regras ao utilizar uma linguagem de programação



Bits e bytes

- **BIT (BInary DigiT)** – dígito binário
  - Menor unidade de informação que armazena somente um valor 0 ou 1

- **Byte (BinarY TErm)** – termo binário
  - Conjunto de 8 bits, com o qual pode-se representar os números, as letras, os sinais de pontuação, etc.

- **Palavra (Word)**
  - É a quantidade de bits que a CPU processa por vez
  - Nos computadores atuais, são comuns palavras de 32 ou 64 bits

Variáveis e constantes

- Tipos de dados

Tipo	Tamanho	Menor valor	Maior valor
char	1 byte	-128	+127
unsigned char	1 byte	0	+255
short int (short)	2 bytes	-32.768	+32.767
unsigned short int	2 bytes	0	+65.535
int (*)	4 bytes	-2.147.483.648	+2.147.483.647
long int (long)	4 bytes	-2.147.483.648	+2.147.483.647
unsigned long int	4 bytes	0	+4.294.967.295
float	4 bytes	-10^{38}	$+10^{38}$
double	8 bytes	-10^{308}	$+10^{308}$

(*) depende da máquina, sendo 4 bytes para arquiteturas de 32 bits

Variáveis e constantes

- Constante
  - Valor armazenado na memória
  - Possui um tipo, indicado pela sintaxe

```
123       /*constante inteira do tipo int*/
12.45     /*constante real do tipo double*/
1245e-2   /*constante real do tipo double*/
12.45F    /*constante real do tipo float*/
```

Variáveis e constantes

- Variável
  - Espaço na memória para armazenar um dado
  - Não é uma variável no sentido matemático

- Uma variável possui
  - Identificador: nome exclusivo para identificar e acessar o espaço de memória
  - Tipo: define a natureza do dado

Variáveis e constantes

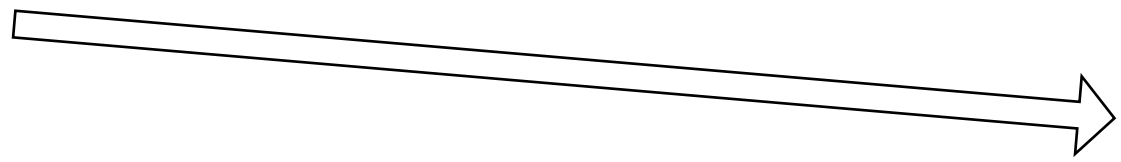
- Declaração de variável
  - Devem ser explicitamente declaradas
  - Podem ser declaradas em conjunto
  - Somente armazenam valores do mesmo tipo com que foram declaradas

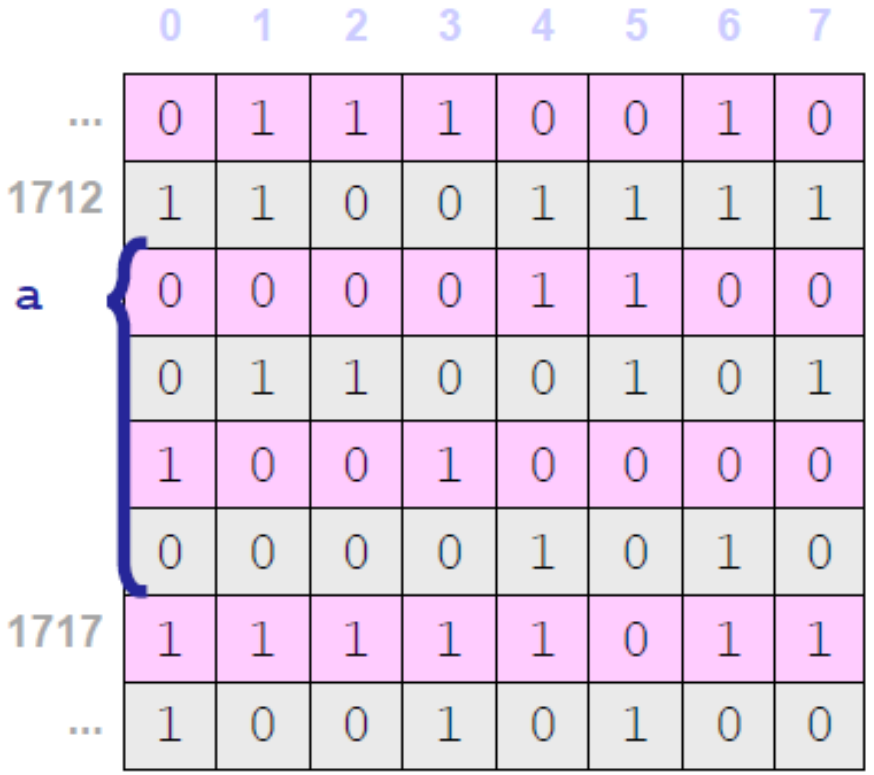
```
char a;   /*declara uma variável do tipo char*/
int b;    /*declara uma variável do tipo int*/
float c;  /*declara uma variável do tipo float*/
int d, e; /*declara duas variáveis do tipo int*/
```

Inicialização de variáveis

- Quando uma variável é declarada, seu valor inicial não é modificado e seu conteúdo é desconhecido
  - Comumente estes valores iniciais desconhecidos são chamados de **lixo**

```
int main() {
    int a;
    int b;
    b = a;
    a = 10;
    return 0;
}
```

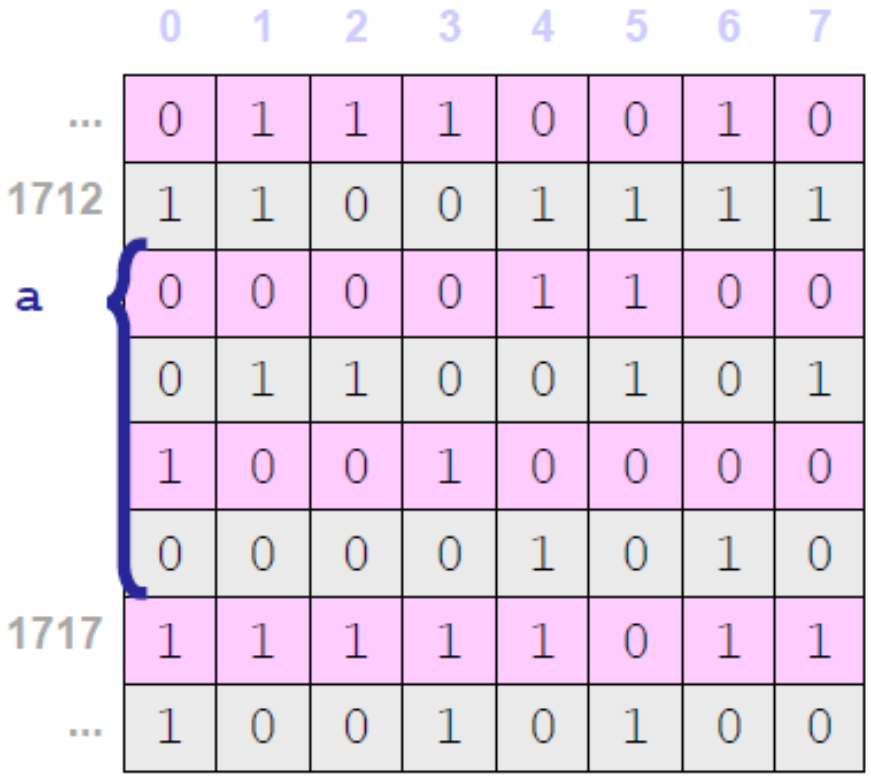




Inicialização de variáveis

- Para evitar problemas, nenhuma variável deve ser utilizada antes de ser inicializada
  - No exemplo abaixo, `b` recebe `11000110010110010000000001010` (em decimal, `207982602`)
  - Em outra execução, possivelmente o valor será outro

```
int main() {
    int a;
    int b;
    b = a;
    a = 10;
    return 0;
}
```



Expressões

- Combinação de variáveis, constantes e operadores que, quando avaliada, resulta em um valor

- Atribuição
  - Variável recebe um determinado valor

- Expressão aritmética, incremento ou decremento
  - Resulta em um número (inteiro ou real)

- Expressão lógica ou relacional
  - Resulta em VERDADEIRO ou FALSO

Expressões aritméticas

Tipo	Operador	Descrição	Inteiros	Reais
Unário	–	Sinal negativo	`-2`	`-2.0`
			`-a`	`-b`
Binário	+	Adição	`a + 2`	`b + 2.0`
	–	Subtração	`a - 2`	`b - 2.0`
	*	Multiplicação	`a * 2`	`b * 2.0`
	/	Divisão	`a / 2`	`b / 2.0`
	%	Módulo	`a % 2`	Operação não definida para reais

Operadores e expressões

- Operadores de atribuição (=, +=, -=, *=, /=, %=)
  - A linguagem C trata uma atribuição como uma expressão
  - A ordem é da direita para a esquerda
  - C oferece uma notação compacta para atribuições em que a mesma variável aparece nos dois lados

```
i += 2;      //é equivalente a i = i + 2
x *= y + 1; //é equivalente a x = x * (y + 1)
```

Operadores e expressões

- Operadores de incremento e decremento (++, −−)
  - Incrementa ou decrementa uma unidade de valor de uma variável
  - Estes operadores não se aplicam a expressão
  - O incremento ou decremento pode ocorrer antes ou depois do uso da variável

```
n++; //incrementa uma unidade em n, depois de ser usado
++n; //incrementa uma unidade em n, antes de ser usado

n = 5;
x = n++;
x = ++n;
a = 3;
b = a++ * 2;
```

Operadores e expressões

- Operadores de incremento e decremento (++, −−)
  - Incrementa ou decrementa uma unidade de valor de uma variável
  - Estes operadores não se aplicam a expressão
  - O incremento ou decremento pode ocorrer antes ou depois do uso da variável

```
n++; //incrementa uma unidade em n, depois de ser usado
++n; //incrementa uma unidade em n, antes de ser usado

n = 5;
x = n++;       //x recebe 5 e n é incrementado para 6
x = ++n;       //n é incrementado para 7 e x recebe 7
a = 3;
b = a++ * 2;  //a é incrementado para 4 e b recebe 6
```

Operadores e expressões

- Operadores relacionais (<, <=, ==, >=, >, !=)
  - O resultado será 0 ou 1
  - Em C é equivalente a FALSO (igual a 0) ou VERDADEIRO (diferente de 0)

```
int a, b;
int c = 23;
int d = c + 4;

c < 20
d > c
```

Operadores e expressões

- Operadores relacionais (<, <=, ==, >=, >, !=)
  - O resultado será 0 ou 1
  - No C é equivalente a FALSO (igual a 0) ou VERDADEIRO (diferente de 0)

```
int a, b;
int c = 23;
int d = c + 4;

c < 20 //retorna 0
d > c  //retorna 1
```

Operadores e expressões

- Operadores lógicos (&&, ||, !)
  - A avaliação ocorre da esquerda para a direita
  - A avaliação para quando o resultado for conhecido, antes mesmo de completar a expressão

```
int a, b;
int c = 23;
int d = c + 4;

a = (c < 20) || (d > c);
b = (c < 20) && (d > c);
```

Operadores e expressões

- Operadores lógicos (&&, ||, !)
  - A avaliação ocorre da esquerda para a direita
  - A avaliação para quando o resultado for conhecido, antes mesmo de completar a expressão

```
int a, b;
int c = 23;
int d = c + 4;

a = (c < 20) || (d > c); //1 e as duas expressões são validadas
b = (c < 20) && (d > c); //0 e só a primeira expressão é validada
```

Operadores e expressões

- Função `sizeof`
  - Retorna o número de bytes ocupados por um tipo de dados

```
int a = sizeof(float); //armazena 4 em a
```

- Conversão de tipo
  - Conversão de tipo é automática na avaliação de uma expressão
  - Conversão de tipo pode ser requisitada explicitamente

```
float f;
float f = 3;        //f recebe 3.0F, conversão realizada de forma implícita
int g, h;
g = (int) 3.5;      //3.5 é convertido (e arredondado) para int
h = (int) 3.5 % 2; //conversão realizada antes do módulo
```

Entrada e saída

- Função `printf`
  - Possibilita a saída de valores conforme o formato especificado

```
printf(formato, expr1, expr2, ..., exprN);
```

```
printf("%d %g", 33, 5.3);
//imprimirá na console a linha "33 5.3"
```

```
printf("Inteiro = %d Real = %g", 33, 5.3);
//imprimirá na console a linha "Inteiro = 33 Real = 5.3"
```

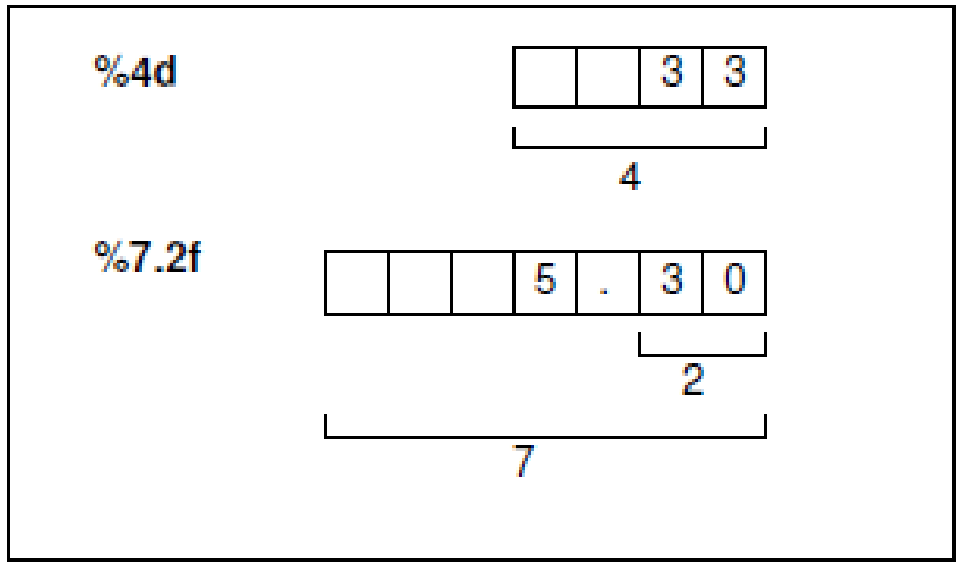
Entrada e saída

- Formatos do `printf`
  - `%c` especifica um `char`
  - `%d` especifica um `int`
  - `%u` especifica um `unsigned int`
  - `%f` especifica um `double` ou `float`
  - `%e` especifica um `double` ou `float` em formato científico
  - `%g` especifica um `double` ou `float` em um formato mais apropriado (`%f` ou `%e`)
  - `%s` especifica uma cadeia de caracteres

- Caractere de escape
  - `\n` quebra de linha
  - `\t` tabulação
  - `\”` caractere “
  - `\\` caractere \

Entrada e saída

- Tamanhos de campo `printf`



Entrada e saída

- Função `scanf`
  - Captura valores fornecidos via teclado

```
scanf(formato, var1, var2, ..., varN);
```

```
int n;
scanf("%d", &n);
//valor inteiro digitado pelo usuário é armazenado em n
```

Entrada e saída

- Formatos do `scanf`

`%c`	especifica um `char`
`%d`	especifica um `int`
`%u`	especifica um `unsigned int`
`%f, %e, %g`	especifica um `float`
`%lf, %le, %lg`	especifica um `double`
`%s`	especifica uma cadeia de caracteres

- Caracteres diferentes dos especificadores
  - Servem para cercar a entrada

```
scanf("%d:%d",&hora,&minuto);
```

Controle de fluxo

- Tomada de decisão
  - Função para qualificar a temperatura

Se a temperatura for menor que 20°, então está frio
Se a temperatura estiver entre 21° e 29°, então está agradável
Se a temperatura for maior 30°, então está quente

- Comando `if`
  - Comando básico para definir desvios ou tomada de decisão
  - Se a condição for verdadeira (diferente de 0), executa o bloco 1
  - Se a condição for falsa (igual a 0), executa o bloco 2

```
if (<expr>) { <bloco 1> } [ else { <bloco 2> } ]
```

Controle de fluxo

```c
#include <stdio.h>
int main()
{
  int temp;
  printf("Digite a temperatura: ");
  scanf("%d", &temp);
  if (temp < 30)
    if (temp > 20)
      printf(" Temperatura agradável \n");
  else
    printf(" Temperatura quente \n");
  return 0;
}
```

Controle de fluxo

```c
#include <stdio.h>
int main()
{
  int temp;
  printf("Digite a temperatura: ");
  scanf("%d", &temp);
  if (temp < 30) {
    if (temp > 20)
      printf(" Temperatura agradável \n");
  } else
    printf(" Temperatura quente \n");
  return 0;
}
```

Controle de fluxo

- Estrutura de bloco
  - Declaração de variáveis
    - Só podem ocorrer no início do corpo da função ou bloco
    - Esta restrição não existe em versões mais recentes do C (C99)
  - Escopo de uma variável
    - Uma variável declarada dentro de um bloco é válida apenas no próprio bloco
    - Após o término da execução do bloco, a variável deixa de existir

```
if (n > 0) {
  int i;
  ...
}
//a variável i não existe mais
```

Construção de laços

- Fatorial de número inteiro não negativo
  - `n! = n x (n – 1)  x (n – 2) x …`

- Calculo não recursivo de `fatorial(n)`
  - Comece com `k = 1` e `f = 1`
  - Faça enquanto `k <= n`
    - `f` recebe `f * k`
    - Incrementa `k`

Construção de laços

- Comando **`while`**
  - Enquanto <expr> for verdadeira, o <bloco de comandos> é executado
  - Quando <expr> for falsa, o laço termina

```
while (<expr>)
{
  <bloco de comandos>
}
```

Construção de laços

```
int main () {
  int k, n;
  int f = 1;
  printf("Digite um numero inteiro nao negativo:");
  scanf("%d", &n);
  k = 1;
  while (k <= n) {
    f = f * k; /* f = f * k é equivalente a f *= k */
    k = k + 1; /* k = k + 1 é equivalente a k++ */
  }
  printf(" Fatorial = %d \n", f);
  return 0;
}
```

Construção de laços

- Comando **`for`**
  - Forma compacta para definir laços

```
for (<expr inicial>; <condição>; <expr incremento>)
{
  <bloco de comandos>
}
```

Construção de laços

```
int main () {
   int k, n;
   int f = 1;
   printf("Digite um numero inteiro nao negativo:");
   scanf("%d", &n);
   for (k = 1; k <= n; k++) {
      f *= k;
   }
   printf(" Fatorial = %d \n", f);
   return 0;
}
```

Construção de laços

- Comando **`do while`**
  - Condição de parada é avaliada ao final do bloco
  - Portanto, sempre executará uma vez o bloco de repetição

```
do
{
  <bloco de comandos>
} while (<condição>);
```

Construção de laços

```
int main () {
  int k, n;
  int f = 1;
  do {
    printf("Digite um numero inteiro nao negativo:");
    scanf("%d", &n);
  } while (n < 0);
  for (k = 1; k <= n; k++) {
    f *= k;
  }
  printf(" Fatorial = %d \n", f);
  return 0;
}
```

Construção de laços

- Comandos **`break`** e **`continue`**
  - **`break`** termina o laço de repetição
  - **`continue`** termina a iteração atual e vai para a próxima

```c
int i;
for (i = 0; i < 10; i++) {
  if (i == 5) break;
  printf("%d ", i);
}

for (i = 0; i < 10; i++) {
  if (i == 5) continue;
  printf("%d ", i);
}
```

Construção de laços

- Comando **`switch`**
  - Seleciona um bloco entre várias opções

```
switch (<expr>) {
  case <opção 1>:
    <bloco 1>;
    break;
  case <opção 2>:
    <bloco 2>;
    break;
  default:
    <bloco padrão>;
    break;
}
```

Variáveis e ponteiros

- Variáveis comuns
  - Possuem um endereço de memória pré-definido

- Ponteiros
  - Apontam para um endereço de memória

- Operadores unários &, * e **
  - Operador &
    - Endereço de ...
    - Utilizado para retornar o endereço de memória de uma variável
  - Operador *
    - Conteúdo de ...
    - Utilizado para indicar o endereço de memória apontado
  - Operador **
    - Ponteiro de ponteiro

Alocação dinâmica

- Função `malloc(size_t n)`
  - Aloca dinamicamente um espaço

```
int *a = malloc(sizeof(int)); //aloca dinamicamente 4 bytes – int
*a = 10;
```

- Criando vetores dinâmicos

```
int *a = malloc(3 * sizeof(int)); //aloca um vetor 4 bytes em cada índice
a[0] = 10;
a[1] = 20;
a[2] = 30;
```

Alocação dinâmica

- Memória estática
  - Armazena instruções do programa, variáveis globais e estáticas
  - É possível definir o tamanho da memória estática antes de executar o programa
- Memória dinâmica
  - Armazena variáveis locais, pilhas de execução e memória alocada dinamicamente
  - Tamanho da memória pode variar de acordo com o fluxo de execução do programa
  - A memória livre é utilizada para alocação dinâmica
    - Pode crescer ou diminuir de através das funções `malloc`, `realloc` ou `free`

Estruturas

- Possibilita a criação de estruturas complexas
  - São formadas por um conjunto de atributos
  - Utiliza-se a instrução **`struct`** para definição de tipos complexos

- Exemplo

```
struct estrutura {
  definição da estrutura
}

struct ponto {
  int x;
  int y;
}

struct ponto p;
p.x = 10;
p.y = 5;
```

Definindo novos tipos de dados

- Definição de novos tipos baseados em estruturas
  - Permite a alocação dinâmica de uma maneira mais usual
  - O tipo de dados é definido pela estrutura definida por **`struct`**
  - Utiliza-se a instrução **`typedef`** para definição de novos tipos
  - Observação
    - O operador de união utilizado em alocação dinâmica é "`->`" ao invés de "`.`"

- Exemplo

```
typedef struct ponto {
  int x;
  int y;
} Ponto;

Ponto *p = malloc(sizeof(Ponto));
p->x = 10;
p->y = 5;
```

Exercícios

1. Dado um vetor de números inteiros `v` de tamanho `n` e um número `k`, retorne verdadeiro se a soma de qualquer par de números em `v` for igual a `k`.
   - Exemplo: dado `v = [10,15,3,7]` e `k = 17`, a saída deve ser `true`, pois `10 + 7` é `17`

2. Dado um vetor de números inteiros `v`, retorne um novo vetor de forma que cada elemento no índice `i` seja o produto de todos os números na matriz original, com exceção de `i`.
   - Exemplo 1: dado `v = [1,2,3,4,5]`, a saída esperada é `[120,60,40,30,24]`
   - Exemplo 2: dado `v = [3,2,1]`, a saída esperada é `[2,3,6]`

Exercícios

3. Dado um vetor de números inteiros v, encontre o primeiro inteiro positivo ausente no vetor. Em outras palavras, deve ser retornado o menor inteiro positivo que não existe no vetor. A matriz pode conter duplicados e números negativos também. O algoritmo deve apresentar complexidade de tempo linear e de espaço constante.
   - Exemplo 1, dado `v = [3,4,-1,1]`, a saída esperada é `2`
   - Exemplo 2, dado `v = [1,2,0]`, a saída esperada é `3`

4. Dada um vetor inteiros v, retorne a maior soma dos números não adjacentes. Os números podem incluir 0 ou negativos no vetor.
   - Exemplo 1, dado `v = [2,4,6,2,5]`, a saída esperada é `13`, considerando `2 + 6 +5`
   - Exemplo 2, dado `v = [5,1,1,5]`, a saída esperada é `10`, considerando `5 + 5`

Exercícios

5. Considere uma escadaria com n degraus e você pode subir 1 ou 2 degraus por vez. Dado n, retorne o número de maneiras únicas de subir a escada.
   - Exemplo, dado $n = 4$, existem 5 maneiras exclusivas
     - `[1,1,1,1]`, `[2,1,1]`, `[1,2,1]`, `[1,1,2]`, `[2, 2]`

6. Dado um vetor de inteiros v com tamanho n e k com intervalo $1 \leq k \leq n$, calcule os valores máximos para cada subvetor de comprimento k gerado a partir do vetor v
   - Exemplo, dado $v = [10,5,2,7,8,7]$ e $k = 3$, a saída esperada será $[10,7,8,8]$, visto que:
     - `10 = max(10,5,2)`
     - `7 = max (5,2,7)`
     - `8 = max (2,7,8)`
     - `8 = max (7,8,7)`

Exercícios

7. Dadas duas listas encadeadas acíclicas de inteiros que se cruzam em algum ponto, localize o primeiro nó de interseção.
   - Exemplo, dado A = 3>7>8->10 e B = 99>1>8>10, a saída esperada será o valor 8

8. Dada uma série de colchetes, parênteses e chaves abertos ou fechados, retorne verdadeiro se a série estiver balanceada (bem formada).
   - Exemplo 1, dada a string `"([]) [] ({})"`, a saída deve ser `true`
   - Exemplo 2, dada a string `"([)]"` ou `"((()"`, a saída deve retornar `false`

Exercícios

9. *Run-length encoding* (RLE) é uma forma simples de compressão de textos. A ideia desta técnica é representar caracteres repetidos sucessivamente com um contador seguido pelo caractere. Dada uma string, retorne o texto resultante da aplicação da técnica RLE.
   - Exemplo, dada a string `"AAAABBBCCDAA"`, a saída compactada deve ser `"4A3B2C1D2A"`

10. *Power set* é um conjunto gerado a partir da combinação de todos seus subconjuntos. Dado um conjunto V, retorne o *power set* deste conjunto de entrada.
   - Exemplo: Dado `V = [1, 2, 3]`, a saída deve ser: `[]`, `[1]`, `[2]`, `[3]`, `[1,2]`, `[1, 3]`, `[2,3]`, `[1,2,3]`

PRA – Projeto de Arquivos

Revisão da linguagem C

Prof. Allan Rodrigo Leite