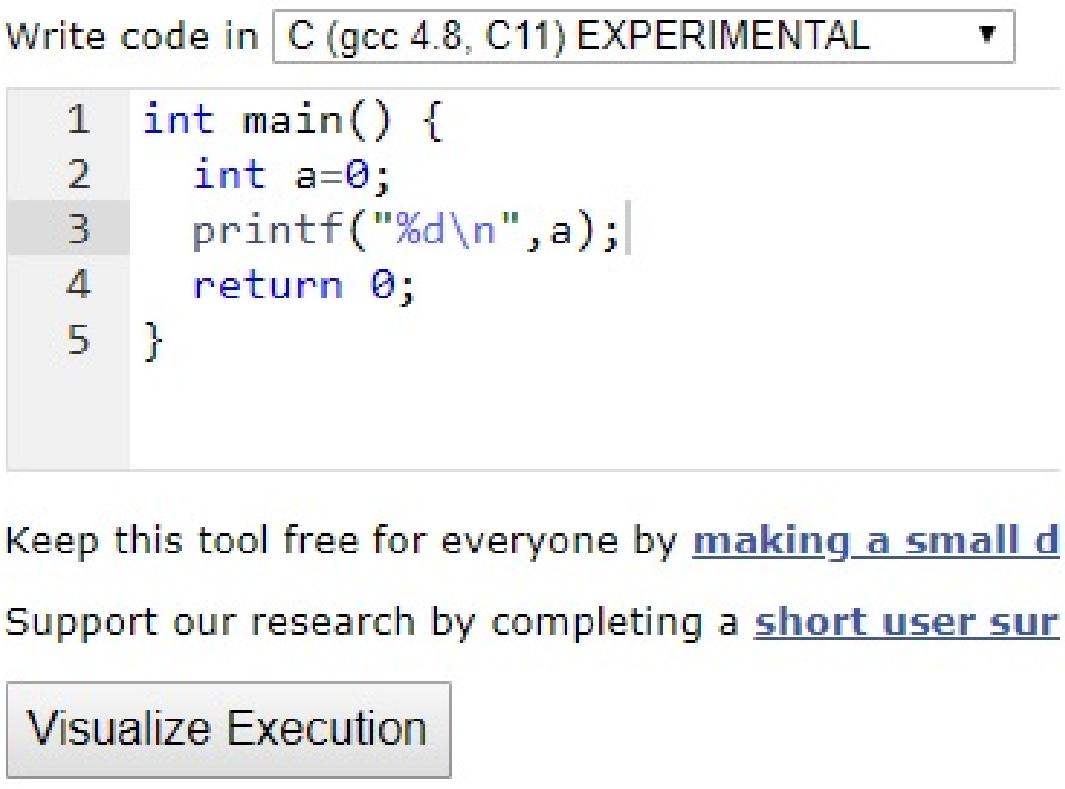
Revisão de Linguagem C

1. Abra o Dev C++ e faça um programa em C que leia 10 valores inteiros e os escreva em ordem crescente. Solução
2. Faça um programa em C que leia 5 datas de aniversário, cada uma formada por dia e mês, e as escreva em ordem cronológica crescente. Solução
3. Faça um programa em C que leia 5 datas de nascimento (dia, mês, ano) e as escreva em ordem cronológica crescente.
4. Faça um programa em C que leia, para 5 candidatos a deputado, seu cadastro, partido (1, 2 ou 3) e número de votos obtidos, e escreva a lista de candidatos em ordem crescente de partido e, para o mesmo partido, ordem crescente de votação. Escreva também o total de votos de cada um dos 3 partidos.
5. Faça um programa que leia uma lista de nomes e datas de nascimento, e escreva a lista de nomes e datas em ordem alfabética.

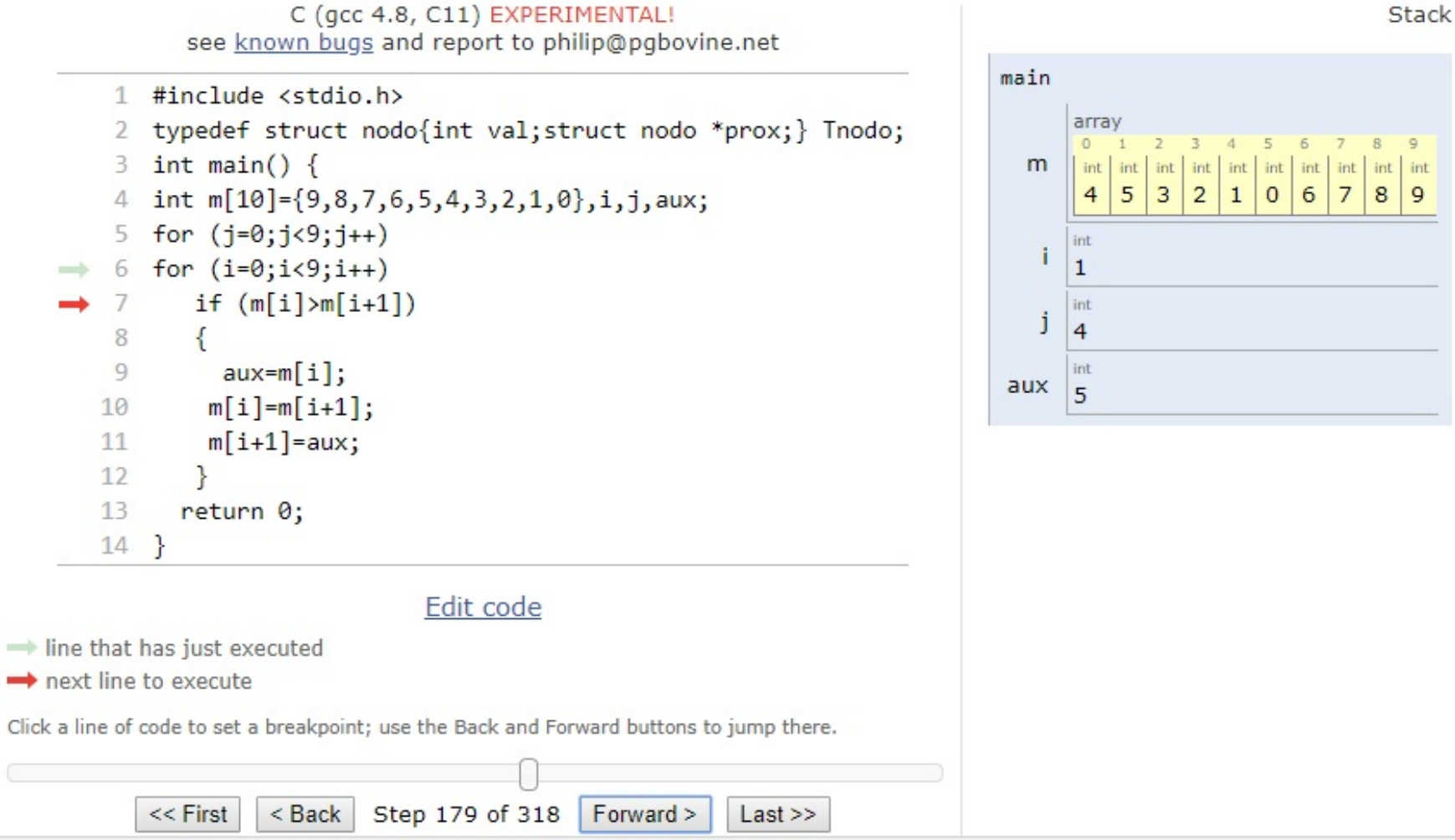
Ferramenta fortemente recomendada

- http://pythontutor.com/ → Start visualizing your code now
- Selecionar linguagem
- Editar código (p.ex. código abaixo)
- Visualize execution



- Ao clicar em "visualize execution"é feita uma simulação da execução do código (aparece uma mensagem pedindo para esperar 10 segundos) e é gerada uma sequência de frames sobre a qual é exibida a simulação.
  - Eventualmente o servidor pode estar sobrecarregado, ou a duração da simulação excede o limite de frames permitido, e não será possível fazer a simulação.
- Como a exibição é feita após a simulação, não é possível interação com o usuário (ou seja, não há comandos de entrada (scanf))
- Dados de teste podem ser inseridos na declaração de variáveis. Ex na lâmina seguinte:

.


C (gcc 4.8, C11) EXPERIMENTAL!
see known bugs and report to philip@pgbovine.net
1 #include <stdio.h>
2 typedef struct nodo{int val;struct nodo *prox;} Tnodo;
3 int main() {
4 int m[10]={9,8,7,6,5,4,3,2,1,0},i,j,aux;
5 for (j=0;j<9;j++)
6 for (i=0;i<9;i++)
7 if (m[i]>m[i+1])
8 {
9 aux=m[i];
10 m[i]=m[i+1];
11 m[i+1]=aux;
12 }
13 return 0;
14 }
Edit code
line that has just executed
next line to execute
Click a line of code to set a breakpoint; use the Back and Forward buttons to jump there.
<< First
< Back
Step 179 of 318
Forward >
Last >>
Stack
main
m
array
0 1 2 3 4 5 6 7 8 9
int int int int int int int int int int
4 5 3 2 1 0 6 7 8 9
i
int
1
j
int
4
aux
int
5

C (gcc 4.8, C11) EXPERIMENTAL!
see known bugs and report to philip@pgbovine.net

```
1  #include <string.h>
2  char n[5][10]={"Joao","Jose","Ana","Bea","Carlos"};
3
4  void main(){
5  char aux[10];
6  int i,j;
7  for (j=0;j<4;j++)
8     for (i=0;i<4;i++)
9        if (strcmp(n[i],n[i+1])>0)
10          {
11          strcpy(aux,n[i]);
12          strcpy(n[i],n[i+1]);
13          strcpy(n[i+1],aux);
14          }
15 for (i=0;i<5;i++)
16    printf("%s\n",n[i]);
17 }
```

Edit code

t has just executed
e to execute

of code to set a breakpoint; use the Back and Forward buttons to jump there.

<< First | < Back | Program terminated | Forward > | Last >>

' @pgbovine. Support with a small donation.

ıs improve this tool by clicking below whenever you learn something:

st cleared up a misunderstanding! | I just fixed a bug in my code!

Print output (drag lower right corner to resize)

```
Ana
Bea
Carlos
Joao
Jose
```

Stack

Global variables

n — array

	0	1	2	3	4	5	6	7	8	9
0	'A'	'n'	'a'	'\0'	'\0'	'\0'	'\0'	'\0'	'\0'	'\0'
1	'B'	'e'	'a'	'\0'	'\0'	'\0'	'\0'	'\0'	'\0'	'\0'
2	'C'	'a'	'r'	'l'	'o'	's'	'\0'	'\0'	'\0'	'\0'
3	'J'	'o'	'a'	'o'	'\0'	's'	'\0'	'\0'	'\0'	'\0'
4	'J'	'o'	's'	'e'	'\0'	's'	'\0'	'\0'	'\0'	'\0'

main

aux — array

0	1	2	3	4	5	6	7	8	9
'J'	'o'	'a'	'o'	'\0'	?	?	?	?	?

i: int 5

j: int 4

- Os botões First e Last permitem ir até o primeiro e o último frame da simulação.
- Os botões Back e Forward permitem avançar ou voltar um frame.
- É possível setar pontos de parada no código para que, ao chegar no ponto selecionado, a execução seja pausada.
  - Isso é feito clicando uma vez sobre a linha a ser inserido o break point. A linha ficará em vermelho.
  - Ao clicar em Forward, ao passar sobre o ponto de parada a simulação será interrompida.
  - Ao clicar em Last os breakpoints serão ignorados
  - Para remover o breakpoint, basta clicar novamente sobre a linha.

Variáveis compostas heterogêneas (Structures)

- Uma *structure* é uma coleção de várias variáveis, possivelmente de tipos diferentes, agrupadas em uma única estrutura.
- São utilizadas para poder manipular conjuntos de informações relacionadas entre si, como por exemplo, os dados que compõem uma data, ou os dados de um cliente de banco.
- Cada uma das variáveis que compõem uma *structure* é chamada de **campo** da structure.

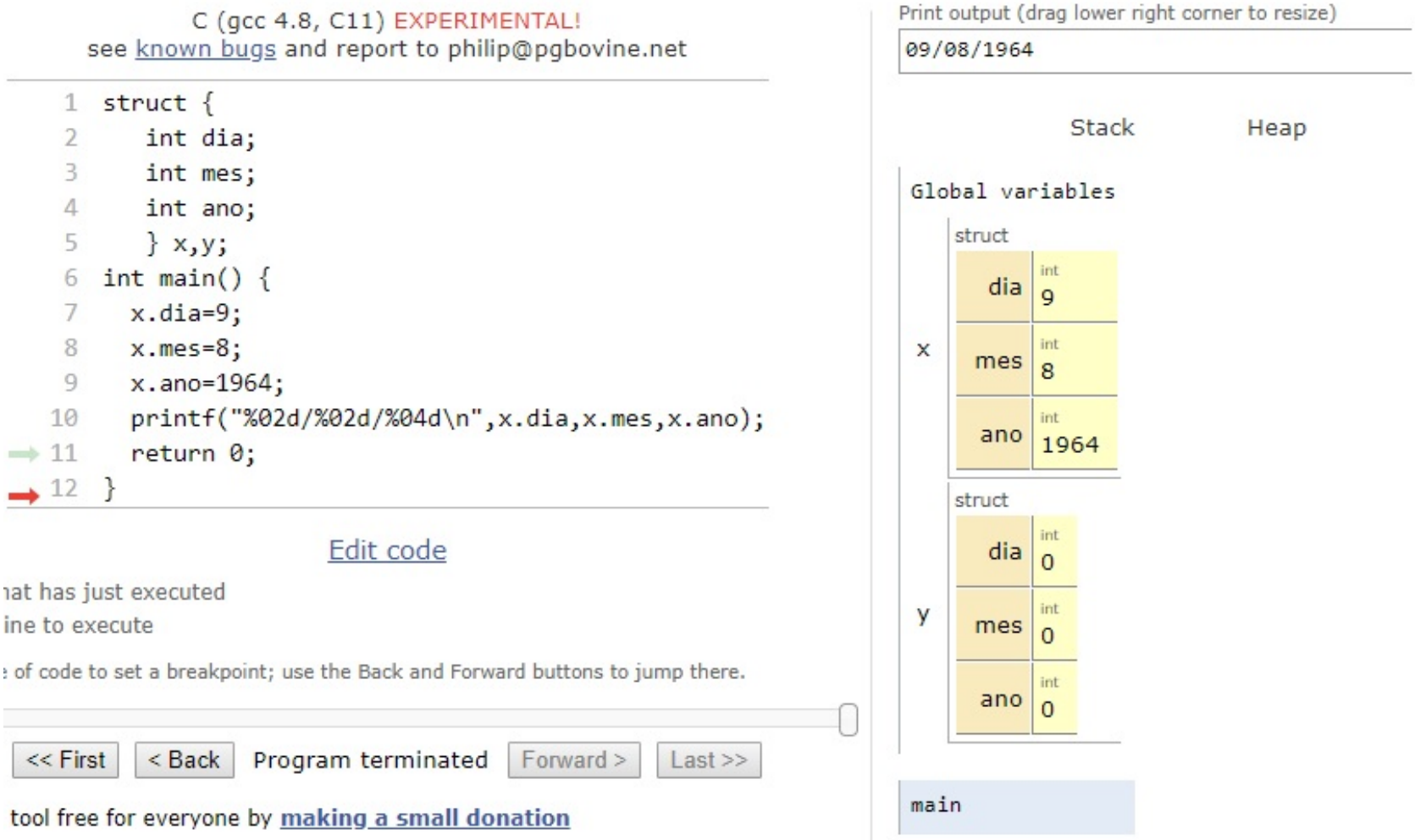
Índice

Uma variável do tipo structure é declarada da forma a seguir:

```
struct { lista de campos } nome;
```

O exemplo abaixo declara uma variável chamada x do tipo structure com três campos inteiros:

```
struct {
    int dia;
    int mes;
    int ano;
    } x;
```

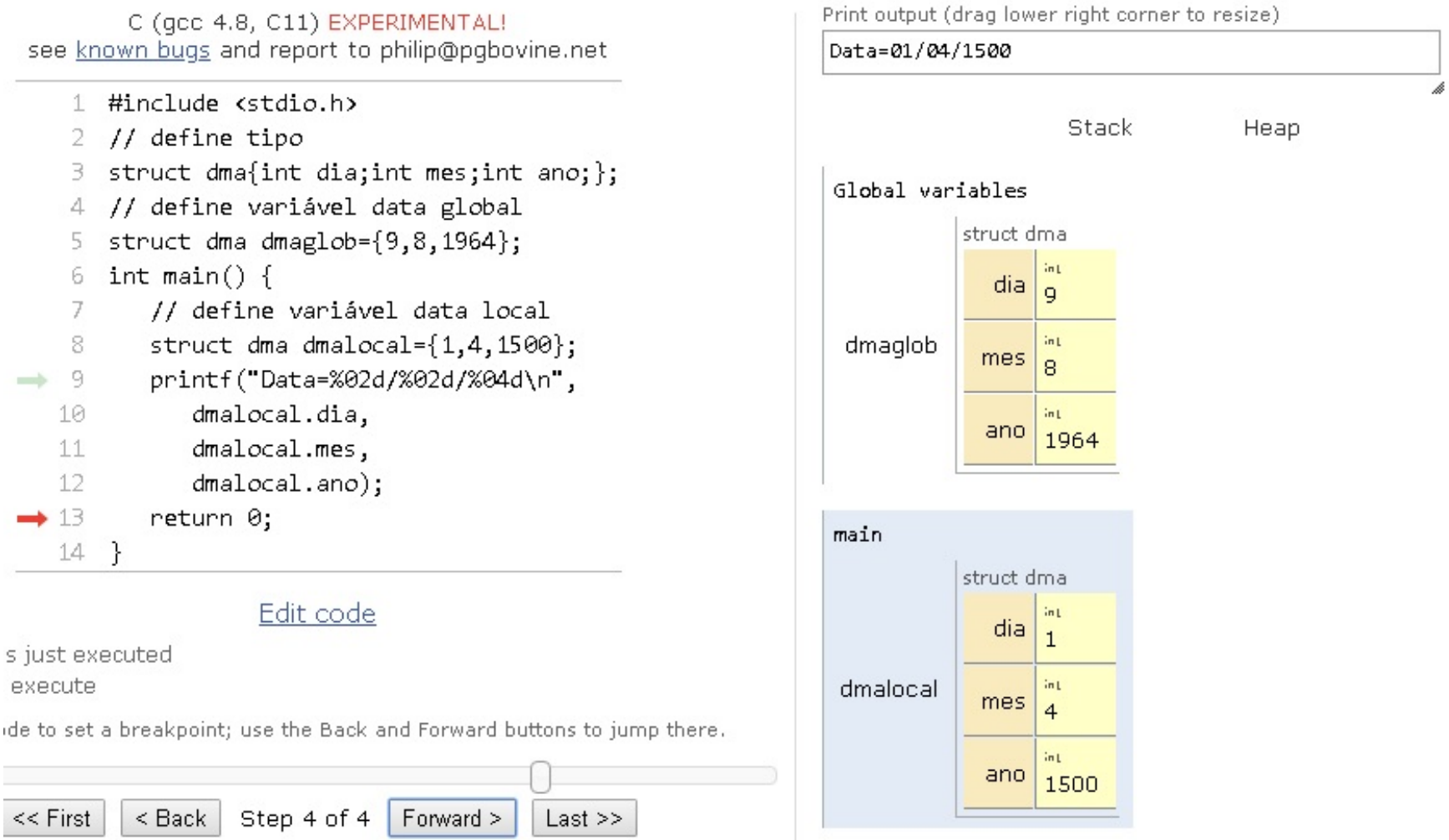

C (gcc 4.8, C11) EXPERIMENTAL!
see known bugs and report to philip@pgbovine.net
1 struct {
2 int dia;
3 int mes;
4 int ano;
5 } x,y;
6 int main() {
7 x.dia=9;
8 x.mes=8;
9 x.ano=1964;
10 printf("%02d/%02d/%04d\n",x.dia,x.mes,x.ano);
11 return 0;
12 }
Edit code
hat has just executed
ine to execute
of code to set a breakpoint; use the Back and Forward buttons to jump there.
<< First
< Back
Program terminated
Forward >
Last >>
tool free for everyone by making a small donation
Print output (drag lower right corner to resize)
09/08/1964
Stack
Heap
Global variables
x
struct
dia
int
9
mes
int
8
ano
int
1964
y
struct
dia
int
0
mes
int
0
ano
int
0
main

- Para trabalhar com *structures*, pode-se associar um nome a um tipo de *structure*, que pode ser posteriormente utilizado em diversas declarações.
- Por exemplo, para a *structure* anterior pode-se usar o nome dma:

```
struct dma {
    int dia;
    int mes;
    int ano;
    };
struct dma x;   /* um registro x do tipo dma */
struct dma y;   /* um registro y do tipo dma */
```

Para se referenciar um campo de uma structure, utiliza-se a notação estrutura.campo.

```
x.dia = 31 /* o campo dia da estrutura x recebe 31 */
x.mes = 8;
x.ano = 1998;
```


C (gcc 4.8, C11) EXPERIMENTAL!
see known bugs and report to philip@pgbovine.net
1 #include <stdio.h>
2 // define tipo
3 struct dma{int dia;int mes;int ano;};
4 // define variável data global
5 struct dma dmaglob={9,8,1964};
6 int main() {
7 // define variável data local
8 struct dma dmalocal={1,4,1500};
9 printf("Data=%02d/%02d/%04d\n",
10 dmalocal.dia,
11 dmalocal.mes,
12 dmalocal.ano);
13 return 0;
14 }
Edit code
s just executed
execute
ıde to set a breakpoint; use the Back and Forward buttons to jump there.
<< First
< Back
Step 4 of 4
Forward >
Last >>
Print output (drag lower right corner to resize)
Data=01/04/1500
Stack
Heap
Global variables
dmaglob
struct dma
dia
int
9
mes
int
8
ano
int
1964
main
dmalocal
struct dma
dia
int
1
mes
int
4
ano
int
1500

- Diferentemente de vetores e matrizes, variáveis do tipo estrutura podem receber o valor de outra do mesmo tipo em uma única atribuição.
- Ex:

```
struct dma {
    int dia;
    int mes;
    int ano;
    } d1,d2;

d1=d2; /* atribui todos os campos ao mesmo tempo */
```

- Da mesma forma, structures podem ser passadas para funções e retornadas por funções.
- Exemplo: A função abaixo recebe a data de início de um evento e a duração do evento em dias.
- Ela devolve a data de fim do evento.

```
struct dma fim_evento (struct dma datainicio,
    int duracao){
    struct dma datafim;
    . . .
    . . .
    datafim.dia = ...
    datafim.mes = ...
    datafim.ano = ...
    return datafim;
    }
```

- O código deve levar em conta a existência de meses com 31 dias, de meses com 30 dias, com 29 dias etc. Eis como essa função poderia ser usada:

```
int main( ) {
        struct dma a, b;
        int d;
        scanf( "%d %d %d", &a.dia, &a.mes, &a.ano);
        scanf( "%d", &d);
        b = fim_evento( a, d);
        printf( "%d %d %d\n", b.dia, b.mes, b.ano);
        }
```

Exercícios

1. Refaça os exercícios 2, 3 e 4 utilizando structures;
2. Complete o código da função fim_evento acima.
3. Escreva uma função que receba dois structs do tipo dma, cada um representando uma data válida, e devolva o número de dias que decorreram entre as duas datas.
4. Escreva uma função que receba um número inteiro que representa um intervalo de tempo medido em minutos e devolva o correspondente número de horas e minutos (por exemplo, converte 131 minutos em 2 horas e 11 minutos). Use uma struct como a seguinte:

```
struct hm {
int horas;
int minutos;
};
```

- Em C pode-se redefinir um tipo de dado dando-lhe um novo nome.
- Essa forma de programação simplifica a referência ao tipo.
- Para redefir-se o nome de um tipo de dado usa-se o comando **typedef**, que provém de *type definition*.
- **typedef** faz o compilador assumir que o novo nome é um certo tipo de dado, e então, passamos a usar o novo nome da mesma forma que usaríamos o antigo.

- A sintaxe do **typedef** é:

  typedef especificação_do_tipo_original nome_novo;

- Ex:

  typedef unsigned int **uint**;
  typedef int **boolean**;
  typedef struct dma{int d;int m;int a;} **tdata**;

- Pode-se definir o nome de uma estrutura de dados (struct) de duas maneiras:
  - a) Definindo o nome da estrutura e só depois definindo a estrutura; p.ex:

```
typedef struct estrutura1 MinhaEstrutura;
struct estrutura1 {
        int var1;
        float var2;
};
```

  - b) Definindo a estrutura ao mesmo tempo que define o nome.

```
typedef struct estrutura1 {
        int var1;
        float var2;
} MinhaEstrutura;
```

- Tendo uma estrutura de dados definida, pode-se utilizá-la dentro de outra estrutura de dados.
- Por exemplo:

```
typedef struct estrutura1 MinhaEstrutura;
struct estrutura1 {
    int var1;
    float var2;
    };

struct estrutura2 {
    int var3;
    MinhaEstrutura var4;
    struct estrutura1 campo5;
    };
```

- Como mostra o exemplo anterior, pode-se usar tanto o novo nome (MinhaEstrutura) quanto o tipo definido (struct estrutura1)
- Ex:

```
typedef struct data {
    unsigned short dia;
    unsigned short mes;
    unsigned int ano;
    } Data;

typedef struct aniversario {
    char nome[50];
    Data nascimento;
    } Aniversario;
```

- Uma estrutura pode conter como campo uma outra estrutura.
- Para isso é conveniente que a estrutura interna tenha sido anteriormente definida (com typedef ou que tenha sido definido um nome para ela).
- Ex:

```
typedef struct {int dia,mes,ano;} date;

typedef struct {
    char nome[30];
    date nascimento;
    } reg;

struct teste2 {int a,b;};
struct teste3 {
    struct teste2 x;
    int y;
    };
struct teste3 z;
```

- Estruturas podem ser inicializadas na declaração, da mesma forma como é feito com variáveis escalares e arrays.
- Para inicializar, a lista de valores iniciais dos campos é delimitada por .
- Esse tipo de atribuição só pode ser feito na declaração.

```
date v2={9,8,1964};
reg lista[2]={{"José",{9,8,1963}},
              {"João",{9,8,1964}}};
reg x={"João",{9,8,1964}};
```

C (gcc 4.8, C11) EXPERIMENTAL!
see known bugs and report to philip@pgbovine.net

```
1  #include <stdio.h>
2  typedef struct {int dia,mes,ano;} date;
3
4  typedef struct {
5     char nome[10];
6     date nascimento;
7     } reg;
8  reg reg1={"Joao",{1,4,1500}};
9
10 struct teste2 {int a,b;};
11 struct teste3 {
12    struct teste2 x;
13    int y;
14    };
15 struct teste3 z;
16 int main() {
17    return 0;
18 }
```

Edit code

as just executed
o execute

ode to set a breakpoint; use the Back and Forward buttons to jump there.

<< First | < Back | Step 1 of 1 | Forward > | Last >>

Global variables

reg1 (struct reg)
- nome: array [0..9] of char = 'J' 'o' 'a' 'o' '\0' '\0' '\0' '\0' '\0' '\0'
- nascimento (struct date)
  - dia: int 1
  - mes: int 4
  - ano: int 1500

z (struct teste3)
- x (struct teste2)
  - a: int 0
  - b: int 0
- y: int 0

main

1. Defina uma structure chamada cliente que contenha as seguintes informações:
   - Nome (char [30])
   - Data de nascimento (structure com dia, mês e ano)
   - Saldo: float
2. Crie um array de 10 elementos inicializados na declaração.
3. Liste em ordem alfabética do nome do cliente o nome e saldo de todos os clientes com saldo maior que R$10.000,00
4. Idem, em ordem de data de nascimento

Exercícios

1. Crie uma estrutura aluno contendo o seu nome (char 30), matrícula (char) e uma lista de até 5 disciplinas (cada disciplina é char 7) e turmas (cada turma é char 3) em que está matriculado. Crie uma lista com 10 alunos inicializando-a na declaração.
2. Faça um programa que leia o código de uma disciplina e o código da turma e liste os alunos da tuma.

- structures podem ser passadas como argumentos para funções, e podem ser retornados por funções. Para isso é necessário que o tipo de structure tenha sido definido antes.
- 1) Faça uma função MaisRico ( ) que receba um array de clientes e retorne o cliente com o maior saldo. Faça duas versões, uma que o retorne como valor de retorno da função, e outra que o retorne como parâmetro de retorno.
- 2) Faça uma função ListaClientes ( ) que recebe um array de clientes e os escreve (nome, data de nascimento e saldo) na ordem em que estão na lista
- 3) Faça uma função OrdenaPorSaldo ( ) que recebe um array de clientes e ordena a lista de clientes por ordem decrescente de saldo. Chame-a e utilize a função ListaClientes para verificar se a ordenação.
- 4) Idem para OrdenaPorNome ( ) e OrdenaPorDataNascimento ( ).

- A estrutura struct tm é uma *structure* definida em time.h e contem uma data e hora com os seguintes campos:

Nome do Campo	Tipo	Significado	Intervalo
tm_sec	int	seconds after the minute	0-61*
tm_min	int	minutes after the hour	0-59
tm_hour	int	hours since midnight	0-23
tm_mday	int	day of the month	1-31
tm_mon	int	months since January	0-11
tm_year	int	years since 1900	
tm_wday	int	days since Sunday	0-6
m_yday	int	days since January 1	0-365
tm_isdst	int	Daylight Saving Time flag	

- The Daylight Saving Time flag (tm_isdst) is greater than zero if Daylight Saving Time is in effect, zero if Daylight Saving Time is not in effect, and less than zero if the information is not available.
- * tm_sec é geralmente 0-59. O intervalo extra é para acomodar segundos "bisextos"em alguns sistemas
- time_t é um alias para um unsigned type, definido em time.h.

O código abaixo busca a hora e data do sistema e escreve valores numéricos correspondentes a cada elemento que a compõe (hora, minuto, segundo, dia, mes, ano e dia da semana). Altere-o para que escreva na tela a data e hora no formato:
Segunda-feira, 23 de abril de 1500, 16:23:47.

```
/* exemplo da função localtime */
#include < stdio.h >
#include < time.h >
#include < stdlib.h >
int main ()
{
time_t rawtime;
struct tm * timeinfo;
time ( &rawtime ); /* busca o número de segundos desde 1/1/1970 */
timeinfo = localtime ( &rawtime ); /* converte para uma estrutura de data-hora */
printf("hora=%d\n",timeinfo->tm_hour);
printf("minuto=%d\n",timeinfo->tm_min);
printf("segundo=%d\n",timeinfo->tm_sec);
printf("dia=%d\n",timeinfo->tm_mday);
printf("mes=%d\n",timeinfo->tm_mon+1);
printf("ano=%d\n",timeinfo->tm_year+1900);
printf("dia da semana=%d\n",timeinfo->tm_wday+1);
system("pause");
return 0;
}
```

Ponteiros

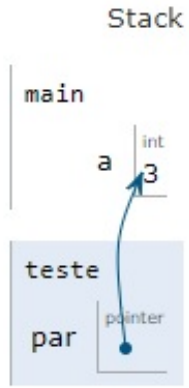
- Um ponteiro é uma variável que contém um endereço de memória. Esse endereço é normalmente uma posição de outra variável na memória ou o endereço de uma estrutura em memória alocada dinamicamente (durante a execução).
- Usam-se ponteiros principalmente para:
  - Fazer passagem de parâmetros por referência em funções;
  - Acessar estruturas alocadas dinamicamente;

Índice

- O conteúdo de um ponteiro é um endereço de memória. Normalmente o endereço não tem muito significado para a visualização ou depuração de um programa.
- Por essa razão, para visualizar o conteúdo de um ponteiro normalmente representa-se como uma seta apontando para a variável cujo endereço o ponteiro contem (diz-se que o ponteiro "aponta"para a variável).

C (gcc 4.8, C11)
EXPERIMENTAL! known bugs/limitations

```
 1  void teste(int *par)
 2  {
 3  *par=0;
 4  }
 5
 6  int main() {
 7  int a=3;
 8  teste(&a);
 9    return 0;
10  }
```



- Uma declaração de ponteiros consiste no formato:

  <tipo> *<nome da variável>

  onde tipo é qualquer tipo válido em C.
- Exs:

```
int *pta;   // pointer para valor inteiro
float *ptf; // pointer para float
```

- O tipo base do ponteiro define que tipo de variáveis o ponteiro pode apontar. Basicamente, qualquer tipo ponteiro pode apontar para qualquer lugar, na memória. Mas para compatibilidade entre diferentes ponteiros em atribuições ambos devem apontar para o mesmo tipo base.
- Os operadores utilizados para trabalhar com ponteiros são * e &. O *, quando colocado antes do ponteiro em uma referência, referencia a posição de memória apontada por ele.
- Ex:

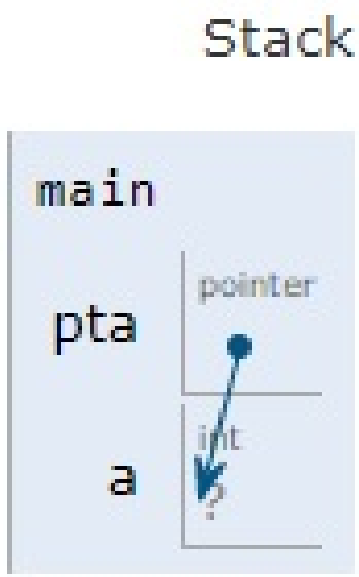
```
int *a;
*a=3; // a posição de memória apontada por a recebe 3
```

E o operador &, aplicado a uma variável, retorna o endereço dela.
Ex:

```
int a, *pta;
pta=&a;      // pta recebe o endereço da variável a.
*pta=5; //    a variável apontada por pta (ou seja, a variável a, recebe 5)
```

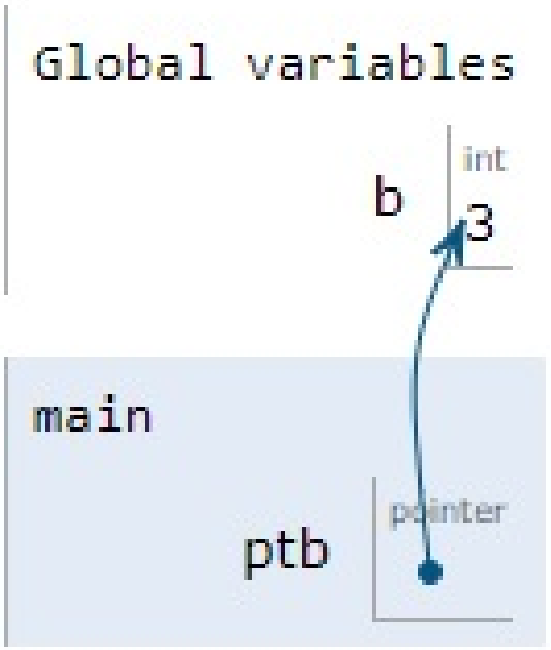
C (gcc 4.8, C11)
ENTAL! known bugs/limitations

```
1  int main() {
2    int *pta;
3    int a;
4    pta=&a;
5    return 0;
6  }
```



C (gcc 4.8, C11)
NTAL! known bugs/limit

```
1  int b;
2  int main() {
3    int *ptb;
4    ptb=&b;
5    *ptb=3;
6    return 0;
7  }
```



```
1) Seja o seguinte trecho de programa:
int i=3,j=5;
int *p, *q;
p = &i;
q = &j;
```

Qual é o valor das seguintes expressões ?

a) p = = &i;

b) *p - *q

c) **&p

d) 3* - *p/(*q)+7

resposta: 1, -2, 3, 6

Precedência de Operadores

- Qual a função de cada operador na expressão a seguir?

  3* - *p/(*q)+7
  - * = multiplicação
  - - = subtração unária (troca de sinal)
  - * = indireção (referência a posição apontada)
  - / = divisão
  - + = soma
- Qual a ordem em que os operadores serão avaliados?
  - Tabela de Precedência de Operadores

Tabela de precedência de operadores em C

Prec.	Símbolo	Função	Associatividade
1	++ - -	Incremento e decremento pósfixo	→
1	()	Parênteses (chamada de função)	
1	[ ]	Elemento de array	
1	.	Seleção de campo de structure	
1	->	Seleção de campo de structure a partir de ponteiro	
2	++ - -	Incremento e decremento prefixo	←
2	+ -	Adição e subtração unária	
2	! ~	Não lógico e complemento	
2	(tipo)	Conversão de tipo de dado	
2	*	Desreferência	
2	&	Referência (endereço de elemento)	
3	* / %	Multiplicação, divisão, e módulo (resto)	→
4	+ -	Adição e subtração	
5	<< >>	Deslocamento de bits à esquerda e à direita	
6	< <=	"menor que"e "menor ou igual que"	
6	> >=	"maior que"e "maior ou igual que"	
7	= = !=	"Igual a"e "diferente de"	
8	&	E bit a bit	
9	^	Ou exclusivo para bits	
10	|	Ou para bits	
11	&&	E lógico	
12	||	Ou lógico	
13	c ? t : f	Condição ternária	←
14	=	Atribuição	
14	+= -=	Atribuição por adição ou subtração	
14	*= /= %=	Atribuição por multiplicação, divisão ou módulo (resto)	
14	<<= >>=	Atribuição por deslocamento de bits	
14	&= ^= |=	Atribuição por operações lógicas	
15	,	vírgula	→

Precedência de Operadores

Qual a função de cada operador na expressão a seguir?

3* - *p/(*q)+7

- * = multiplicação (precedência 3)
- - = subtração unária (precedência 2)
- * = indireção (precedência 2)
- / = divisão (precedência 3)
- + = soma (precedência 4)
- Em primeiro lugar será avaliado o operadores de indireção mais à direita, resultando a expressão:

  3* - *p/5+7
- Em seguida será avaliado o outro operador de indireção, já que tem mesma precedência da subtração unária e é associativo à direita, resultando

  3* - 3/5+7
- Em seguida será aplicada a subtração unária e em seguida será efetuada a multiplicação, resultando

  -9/5+7
- Agora é efetuada a divisão, resultando

  -1+7
- Que resulta:

  6

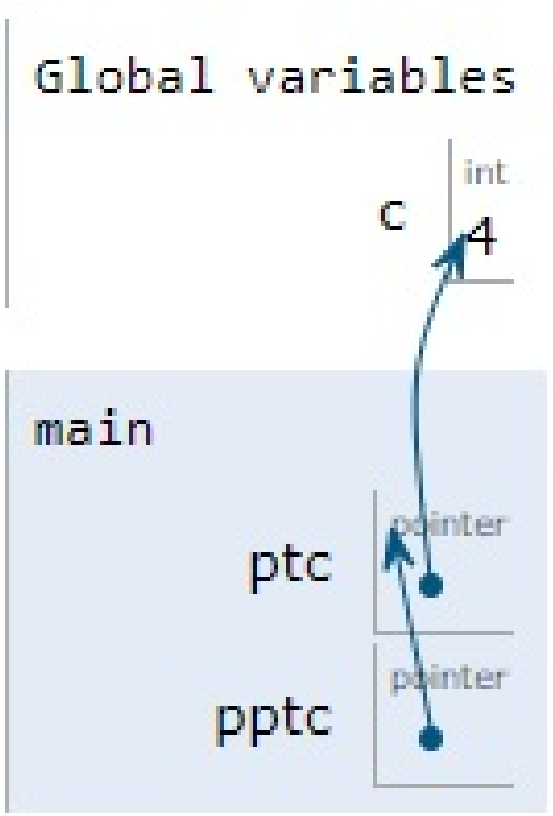
- Pode-se escrever o valor de um ponteiro (endereço para o qual ele aponta) utilizando-se o especificador %p (mostra em hexa, 32 bits) ou %x (mostra em hexa, sem mostrar os 0's não significativos).
- Qual será a saída deste programa supondo que i ocupa o endereço 4094 na memória?

```
main() {
        int i=5, *p;
        p = &i;
        printf("%x %d %d %d %d \n", p,*p+2,**&p,3**p,**&p+4);
        printf("%d\n",(*p)++ + *q);
        printf("%d\n",i);
        printf("%d\n",++(*p) + *q);
}
```

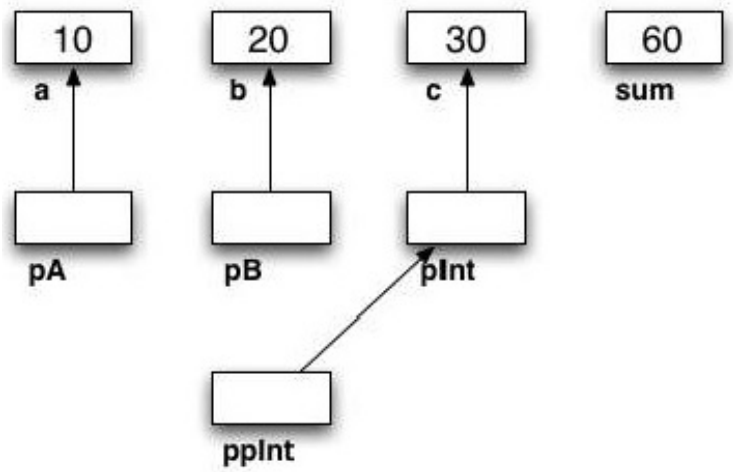
- O tipo base de um pointer pode ser um outro pointer (pointer para pointer).

NTAL! known bugs/limita

```
int c;
int main() {
  int *ptc,**pptc;
  ptc=&c;
  pptc=&ptc;
  **pptc=4;
  return 0;
}
```



```
void pointers2pointers()
  {
  int a=10, b=20, c=30, sum=0;
  int *pA=&a,*pB=&b,*pInt;
  int **ppInt = &pInt;
  (*ppInt)=pA;
  sum += (**ppInt);
  (*ppInt)=pB;
  sum+=(*pInt);
  *ppInt=&c;
  sum+=(**ppInt);
  printf("Soma é %d\n",sum);
  }
```



Um ponteiro pode ser usado no lado direito de um comando de atribuição para passar seu valor para outro ponteiro.

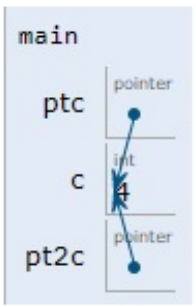
```c
#include <stdio.h>
void main() {
    int x;
    int *p1,*p2; /* declaração do ptr p1 e p2 com o tipo base int. */
    p1 = &x;
    p2 = p1;
    printf("%p",p2); /* escreve o endereço de x em 32 bits, não seu valor */
    return 0;
    }
```

C (gcc 4.8, C11)
MENTAL! known bugs/limitations

```c

int main() {
  int *ptc,c,*pt2c;
  ptc=&c;
  pt2c=ptc;
  *ptc=4;
  return 0;
}
```

Stack



- Agora, tanto p1 quanto p2 apontam para x.
- O endereço de x é mostrado usando o modificador de formato de printf() %p, que faz com que printf() apresente um endereço no formato usado pelo computador hospedeiro.

Aritmética de Ponteiros

- Existem apenas duas operações aritméticas que podem ser usadas com ponteiros: adição e subtração.
- Os operadores permitidos no caso seriam: +, -, ++, - -.
- O incremento é sempre realizado do tamanho básico de armazenamento do tipo base.
- Isto é, se o tipo base for um inteiro e incrementarmos em uma unidade, o apontador apontará para o próximo inteiro (no caso do inteiro ocupar 4 bytes o incremento será de 4 bytes), no caso de um caractere (char) será de um byte.

Número de bytes dos tipos em C

tipo	Tamanho
short int	2
int	4
long int	4
long long int	8
double	8
float	4

```
int *ptri=3000;
char *ptrc=4000;
float *ptrf=4000;
ptri++; /* ptri apontará para o endereço 3004 */
ptrc++; /* ptrc apontará para o endereço 4001 */
ptrf++; /* ptrf apontará para o endereço 5004 */
ptri = ptri - 10; /* ptri apontará para o endereço 2964 */
ptrc = ptrc - 10; /* ptrc apontará para o endereço 3991 */
ptrf = ptrf - 10; /* ptrf apontará para o endereço 4964 */
```

- Além de adição e subtração entre um ponteiro e um inteiro, nenhuma outra operação aritmética pode ser efetuada com ponteiros.
- Isto é, não se pode multiplicar ou dividir ponteiros; não se pode aplicar os operadores de deslocamento e de mascaramento bit a bit com ponteiros e não se pode adicionar ou subtrair o tipo float ou o tipo double a ponteiros.

- Se a e b são variáveis inteiras e pA e pB ponteiros para int, quais das seguintes expressões de atribuição são ilegais?

  a) pA = &a;

  b) *pB = &b;

  c) pA = &*&a;

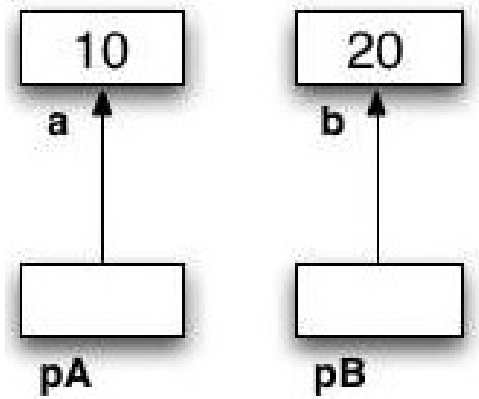
  d) a = (*&)b;

  e) a = *&b;

  f) a = *&*&b;

  g) pB = *pA;

  h) a = (*pA)++ + *pB



- Resposta: as ilegais são B, D, G

- 6. Assumindo que pulo[] é um vetor do tipo int, quais das seguintes expressões referenciam o valor do terceiro elemento da matriz?
  - a) *(pulo + 2)
  - b) *(pulo + 4)
  - c) pulo + 4
  - d) pulo + 2
- 7. Supor a declaração: int mat[4], *p, x; Quais expressões são válidas? Justifique:
  - a) p = mat + 1;
  - b) p = mat++;
  - c) p = ++mat;
  - d) x = (*mat)++;

Alocação dinâmica de memória

- **Pointers** são utilizados para referenciar variáveis (normalmente structures, vetores e matrizes) alocadas dinamicamente, ou seja durante a execução do programa.
- As funções utilizadas para isso são malloc(), calloc() e free().
- Protótipos

```
#include < stdlib.h >
void * calloc ( size_t nmemb , size_t size );
void * malloc ( size_t size );
void free (void * ptr );
```

Pode-se considerar que size_t é equivalente a int.

Índice

Descrição

- **calloc** reserva memória para um vetor de nmemb elementos de tamanho size bytes cada e retorna um ponteiro para a memória reservada. A memória é inicializada com zero.
- **malloc** reserva size bytes e retorna um ponteiro para a memória reservada. A memória não é inicializada com zero.
- **free** libera a área de memória apontada por ptr, que foi previamente reservada por uma chamada a uma das funções malloc() ou calloc(). Comportamento indefinido ocorre se a área já foi liberada antes ou as funções não foram chamadas antes. Se o valor de ptr é NULL nada é executado.

- Retorno
  - Em **calloc()** e **malloc()**, o valor retornado é um ponteiro para a memória alocada, que é alinhada corretamente para qualquer tipo de variável ou NULL se o pedido não pode ser atendido.
  - O ponteiro retornado é do tipo void *. Para que ele possa ser atribuído a ponteiros de outro tipo é necessário fazer a conversão de tipo.
  - Para saber o tamanho da área necessária para uma variável, pode-se utilizar a função sizeof( ).
  - Ex:

    ```
    struct cliente *pcli=(struct cliente *)malloc(sizeof(struct
    cliente));
    ```

- **free()** não retorna nenhum valor.

Comparação de Ponteiros

- É possível comparar dois ponteiros em uma expressão relacional.

```
if (p<q)
        printf("p aponta para uma memória mais baixa que q
\n!");
```

- Geralmente, a utilização de comparação entre ponteiros é quando os mesmos apontam para um objeto comum.
- Exemplo disto é a pilha, onde é verificado se os ponteiros de início e fim da pilha estão apontando para a mesma posição de memória, significando pilha vazia.

Ponteiros e Matrizes

- Existe uma estreita relação entre matrizes e ponteiros. C fornece dois métodos para acessar elementos de matrizes: aritmética de ponteiros e indexação de matrizes.
- Aritmética de ponteiros pode ser mais rápida que indexação de matrizes.
- Normalmente utilizam-se ponteiros para acessar elementos de matrizes devido à velocidade de acesso.
- Exemplo
  - char str[80], *p1;
  - p1 = str;
  - Para acessar a string str pode-se utilizar estes dois mecanismos:
    - str[4] /* indexação de matrizes */
  - ou
    - *(p1 + 4) /* aritmética de ponteiros */
  - Os dois comandos devolvem o quinto elemento.
  - *(matriz + índice) é o mesmo que matriz[índice].

- Para uma melhor compreensão ou facilidade de programação as funções de indexação trabalham com ponteiros (como mostra a implementação abaixo, da função puts()).

```
/* Indexa s como uma matriz */
void put(char *s){
        register int t;
        for (t=0;s[t]; ++t) putchar(s[t]);
}
/* Acessa s como um ponteiro */
void put(char *s) {
        while (*s) putchar(*s++);
}
```

tabela

No caso da passagem de parâmetros é possível tratar uma matriz como se fosse um ponteiro.

```
#include <stdlib.h>
#include <stdio.h>
#include <string.h>
void le_tab(int *p) {
   register int i;
   for(i=0; i<20; i++)
      scanf("%d",(p+i));
}
void mostra_tab(int *p) {
   register int i;
   for(i=0; i<20; i++)
      printf("%d",*(p+i));
}
void main(void) {
   int mat[20];
   le_tab(mat);
   mostra_tab(mat);
   }
```

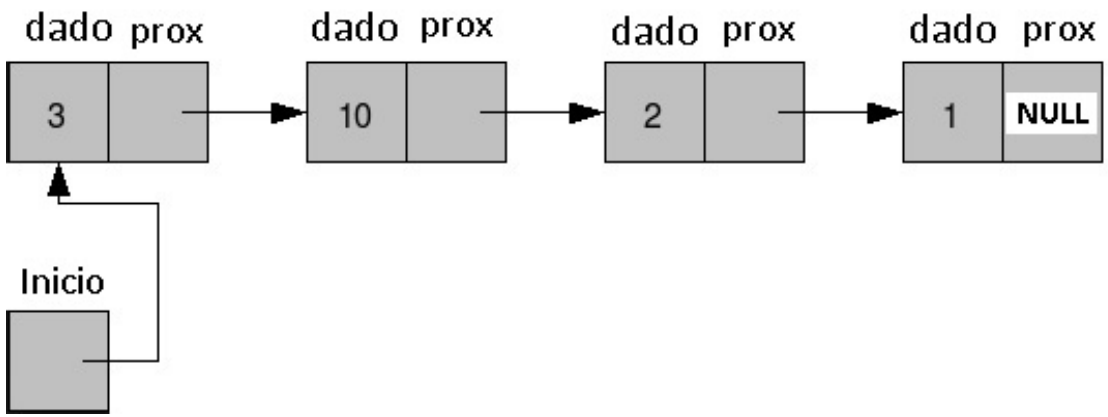
Listas Simplesmente Encadeadas

- Uma lista simplesmente encadeada é uma seqüência de objetos alocados dinamicamente, cada qual fazendo referência ao seu sucessor na lista.
- Vantagens:
  - Permite o crescimento dinâmico da lista
  - Diminui o esforço computacional nas operações de inserção e remoção de nós

Índice

Características

- Cada nó é criado na memória conforme necessário;
- Cada nó deverá conter o dado propriamente dito e a indicação lógica para o nó seguinte (ponteiro);
- Os nós não estão alocados sequencialmente na memória.
- Assim, a relação de ordem é somente lógica;
- Para marcar o final da lista, a referência (ponteiro) no último nó da lista deverá ter valor nulo.



Operações sobre listas simplesmente encadeadas

- Definição do tipo nodo (apenas a definição do tipo, não cria nenhuma variável):

```
struct nodo {
        char nome[10];
        struct nodo *prox;};
```

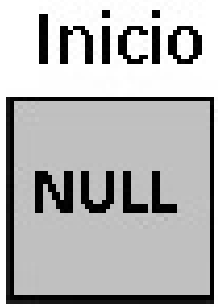
- ou

```
typedef struct nodo{int valor;struct nodo *prox;} Tnodo;
```

- Declaração do pointer que apontará para o início da lista e inicialização com NULL:

  struct nodo *inicio=**NULL**;

- Obs: NULL é uma constante do tipo pointer definida na biblioteca stdio.



1) Inserção de um elemento em uma lista vazia

(a) struct nodo *inicio;

(b) inicio=(Tnodo *)malloc(sizeof(Tnodo));

(c) strcpy(inicio->valor,"Joca");

(d) inicio->prox=NULL;

Figura: (a)

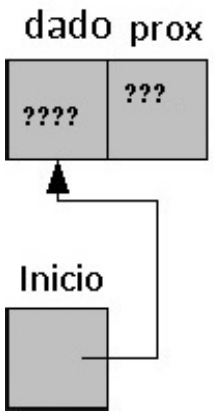


Figura: (b)

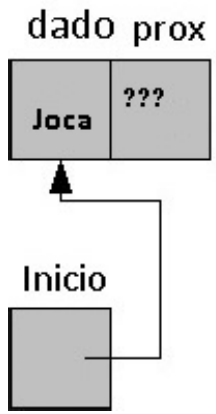


Figura: (c)

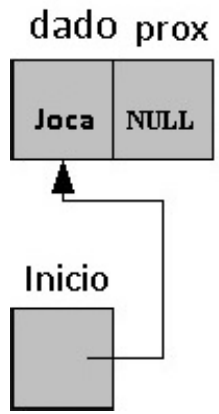


Figura: (d)

2) Inserção de valor no início da lista

(a) Tnodo *ptaux;

(b) ptaux=(Tnodo *)malloc(sizeof(Tnodo));

(c) ptaux->prox=inicio;

(d) strcpy(ptaux->nome,"Zé");

(e) inicio=ptaux;

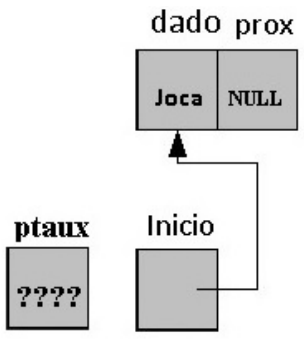


Figura: (a)

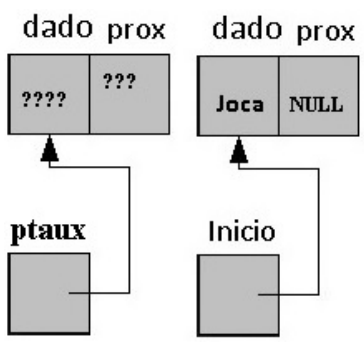


Figura: (b)

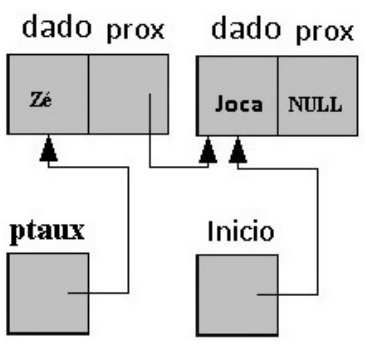


Figura: (c,d)

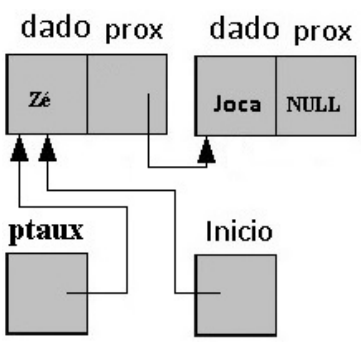


Figura: (e)

- Obs: O código de inserção no início da lista funciona também para inserção em lista vazia, se o pointer do início da lista contiver NULL.
- No código abaixo as operações de inserção no início da lista foram colocados dentro de uma função. A função funciona?

```
void ins_inic(Tnodo *inicio,char nome[])
   {
   Tnodo *ptaux;
   ptaux=(Tnodo *)malloc(sizeof(Tnodo));
   ptaux->prox=inicio;
   strcpy(ptaux->nome,nome);
   inicio=ptaux;
   }
```

3) Faça uma função que receba um apontador de lista e escreva todos os nomes da lista.

```
void lista(struct nodo *inicio)
{
    struct nodo *ptaux=inicio;
    while (ptaux!=NULL)
        {
        printf("%s\n",ptaux->nome);
        ptaux=ptaux->prox;
        }
}
```

- Como ficaria uma versão recursiva???

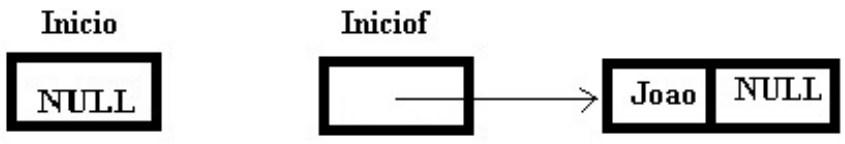
- 4) Faça uma função que receba um apontador de lista (possivelmente vazia) e um nome, e insira o nome no início da lista. Utilize a função para inserir 10 nomes. Utilize a função anterior para listar a lista resultante (ATENÇÃO, O CÓDIGO ABAIXO ESTÁ **ERRADO**).

```
void ins_inic(struct nodo *iniciof,char N[10])
    {
    struct nodo *ptaux=(struct nodo*)malloc(sizeof(struct
nodo));
    strcpy(ptaux->nome,N);
    ptaux->prox=iniciof;
    iniciof=ptaux;
    }
```

- O parâmetro foi chamado de iniciof para salientar que este é o nome que ele tem dentro da função (o parâmetro **formal**).
- O parâmetro *iniciof* está sendo passado por referência ou por valor?
- Isso funciona?

- Considere a chamada ins_inic(inicio, "Joao")
- Ao entrar na função é feita uma cópia do valor do parâmetro real (inicio) para o parâmetro formal (iniciof)
- A situação fica assim:

- Ao final da função, antes de retornar, a situação é:



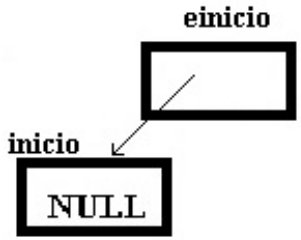
- E imediatamente após o retorno o pointer *iniciof* deixa de existir e o nodo fica perdido, sem ninguém a apontar para ele:

- Para que o pointer inicio possa ser alterado dentro da função é necessário passar o endereço dele.
- Assim, a função ao invés de receber o valor do pointer, recebe o endereço do pointer (vamos chama-lo de *einicio*, de "endereço de início")
- E o cabeçalho da função seria:

  void ins_inic(struct nodo **einicio,char N[10])
- E na chamada passa-se o endereço de inicio ao invés de seu valor:

  ins_inic(&inicio,"Joca")

- Considere a chamada ins_inic(&inicio, "Joca")
- Ao entrar na função o endereço da variável **inicio** é passado como parâmetro e colocado em **einicio**, que passa a apontar para **inicio**
- A situação fica assim:



- Dentro da função, qualquer referência ao pointer **inicio**, é feita através de **einicio**, usando o operador *.
- Normalmente usa-se o nome inicio (head, cabeça ou equivalente) no nome do parâmetro, apesar de, na realidade, o parâmetro conter o endereço do apontador do início da lista.

- Na versão abaixo o parâmetro inicio está sendo passado por referência.

```
void ins_inic(struct nodo **einicio,char N[10])
    {
    struct nodo *ptaux=(struct nodo*)malloc(sizeof(struct
nodo));
    strcpy(ptaux->nome,N);
    ptaux->prox=*einicio;
    *einicio=ptaux;
    }
```

Passagem de pointer por referência

- Quando uma função deve alterar o valor de um parâmetro, o mesmo deve ser passado por referência, isto é, ao invés de passar o valor do parâmetro, passa-se o endereço do mesmo para que, através desse endereço, possa-se alterar o valor do parâmetro.
- Faça um programa que leia 20 números inteiros, inserindo os números pares em uma lista e os números ímpares em outra lista. Escreva ao final as duas listas geradas.

5) Inserção no final da lista

```
ptaux=(struct nodo*)malloc(sizeof(struct nodo));
strcpy(ptaux->nome,"Juca");
ptaux->prox=NULL;
if (inicio==NULL)
        inicio=ptaux;
else
        {
        ptaux2=inicio;
        while (ptaux2->prox!=NULL)
                ptaux2=ptaux2->prox;
        ptaux2->prox=ptaux;
        }
```

- Como ficaria em uma função? E a versão recursiva???

Inserção recursiva no final da lista

```
void ins_fim(struct nodo **inicio, char nome[10])
    {
    if (*inicio==NULL)
        {
        *inicio=(struct nodo*)malloc(sizeof(struct nodo));
        (*inicio)->prox=NULL;
        strcpy((*inicio)->nome,nome);
        }
    else ins_fim(&(*inicio)->prox,nome);
    }
```

- Exercícios do Hacker Rank
  - Print the Elements of a Linked List
  - Insert a Node at the Tail of a Linked List
  - Insert a node at the head of a linked list
  - Insert a node at a specific position in a linked list
  - Delete a Node
  - Print in Reverse
  - Compare two linked lists
  - Merge two sorted linked lists
  - Get node value (a partir do último elemento da lista)

Inserção ordenada

6) Escreva uma função não-recursiva que receba um apontador para o início de uma lista e um nome, e insira um nodo com o nome recebido na posição correta, de modo a manter a lista ordenada em ordem alfabética.
7) Faça uma função que receba um apontador para um nodo, e um nome, e insira um nodo com o nome recebido, entre o nodo atual e o próximo.

```
void ins_ord(struct nodo **inicio, int val)
    {
    struct nodo *ptaux,*ptnovo=(struct nodo *)malloc(sizeof(struct nodo));
    ptnovo->valor=val;
    ptnovo->prox=*inicio;
    if (*inicio==NULL || (*inicio)->valor>=val)
        {
        *inicio=ptnovo;
        return;
        }
    ptaux=*inicio;
    while (ptaux->prox!=NULL && ptaux->prox->valor<val)
        ptaux=ptaux->prox;
    ptnovo->prox=ptaux->prox;
    ptaux->prox=ptnovo;
    }
```

- 8) Faça uma função que receba um apontador do início da lista e remova da lista o primeiro elemento da mesma. Assuma que a lista possui pelo menos um elemento.
- 9) Faça uma função que receba um apontador do início de uma lista e remova da lista o último elemento da mesma. Faça uma versão recursiva e uma não recursiva. Escreva a lista após a remoção para verificar que a remoção foi feita corretamente.
- 10) Escreva uma função que receba um apontador para o início de uma lista e um nome, e remova a primeira ocorrência do nome da lista.
- 11) Escreva uma função que receba um apontador para o início de uma lista e um valor, e remova todas as ocorrências do valor da lista.

Remoção do início da lista

- 8) Faça uma função que receba um apontador do início da lista e remova da lista o primeiro elemento da mesma.

```
void rem_inic(struct nodo **inicio)
    {
    struct nodo *ptaux=*inicio;
    *inicio=ptaux->prox;
    free(ptaux);
    }
```

Remoção do Final da Lista

9) Faça uma função que receba um apontador do início de uma lista e remova da lista o último elemento da mesma. Faça uma versão recursiva e uma não recursiva. Escreva a lista após a remoção para verificar que a remoção foi feita corretamente.

```
void rem_fim(struct nodo **inicio)
    {
    if (*inicio==NULL) return;
    struct nodo *ptaux=*inicio;
    while (ptaux->prox!=NULL)
        {
        pant=ptaux;
        ptaux=ptaux->prox;
        }
    free(ptaux);
    pant->prox=NULL;
    }
```

Remoção recursiva do final

```
void rem_fim(struct nodo **inicio)
    {
    if (*inicio==NULL) return;
    if ((*inicio)->prox)==NULL)
        {
        free(*inicio);
        *inicio=NULL;
        }
    else rem_fim(&(*inicio)->prox);
    }
```

Remoção recursiva em qualquer lugar da lista

10) Escreva uma função que receba um apontador para o início de uma lista e um nome, e remova a primeira ocorrência do nome da lista.

```
void remove(struct nodo **inicio, char N[10])
{
struct nodo*ptaux;
if (*inicio==NULL) return;
if (strcmp((*inicio)->nome,N)!=0)
      {
      remove(&((*inicio)->prox),N);
      return;
      }
ptaux=*inicio
(*inicio)=(*inicio)->prox;
free(ptaux);
}
```

11) Escreva uma função que receba um apontador para o início de uma lista e um valor, e remova todas as ocorrências do valor da lista.

```
void remove(struct nodo **inicio, int val)
{
if (*inicio==NULL) return;
if ((*inicio)->valor==val)
        {
        ptaux=(*inicio)->prox;
        free(*inicio);
        *inicio=ptaux;
        remove(&(*inicio));
        }
else remove(&(*inicio)->prox);
}
```

- 12) Faça uma função que receba um apontador de início de lista e um valor, e procure o valor na lista, retornando 0 se o valor não existe na lista, e 1 se o valor existe.
- 13) Faça uma função para desalocar todos os elementos de uma lista. Faça uma versão recursiva e uma não recursiva.
- 14) Faça uma função **recursiva** que receba um apontador do início da lista e um nome, e verifique se o nome está na lista, retornando o apontador para o nodo que contem o nome, e NULL caso o nome não esteja na lista.
- 15) Idem à anterior, mas considerando que os elementos da lista estão em ordem crescente.

12) Faça uma função que receba um apontador de início de lista e um valor, e procure o valor na lista, retornando 0 se o valor não existe na lista, e 1 se o valor existe.

```
int procura(struct nodo *inicio, char N[10])
    {
    struct nodo *ptaux=inicio;
    while (ptaux!=NULL)
        {
        if (strcmp(ptaux->nome,N)==0) return 1;
        ptaux=ptaux->prox;
        }
    return 0;
    }
```

- Faça com que a função retorne o pointer para o nodo com o valor encontrado. Retorne NULL se a lista não contiver o valor.

- 13) Faça uma função para desalocar todos os elementos de uma lista. Faça uma versão recursiva e uma não recursiva.
- Não recursiva:

```
void desaloca (struct nodo **inicio)
    {
    struct nodo *ptaux;
    while (*inicio!=NULL)
        {
        ptaux=*inicio;
        *inicio=(*inicio)->prox;
        free(ptaux);
        }
    *inicio=NULL;
    }
```

- recursiva:

```
void desaloca (struct nodo **inicio)
{
if (*inicio==NULL) return;
desaloca(&(*inicio)->prox);
free(*inicio);
*inicio=NULL;
}
```

14) Faça uma função **recursiva** que receba um apontador do início da lista e um nome, e verifique se o nome está na lista, retornando o apontador para o nodo que contem o nome, e NULL caso o nome não esteja na lista.

```
struct nodo *procura(struct nodo *inicio,char N[10])
    {
    if (inicio==NULL) return NULL;
    if (strcmp(inicio->nome,N)==0) return inicio;
    return procura(inicio->prox,N);
    }
```

15) Idem à anterior, mas considerando que os elementos da lista estão em ordem crescente.

```
struct nodo *procura(struct nodo *inicio,char N[10])
{
if (inicio==NULL) return NULL;
if (strcmp(inicio->nome,N)==0) return inicio;
if (strcmp(inicio->nome,N)>0) return NULL;
return procura(inicio->prox,N);
}
```

- 16) Faça uma função recursiva que retorne o número de nodos de uma lista.
- 17) Escreva uma função recursiva que verifica se duas listas dadas são iguais, ou melhor, se têm o mesmo conteúdo e retorne:

16) Faça uma função recursiva que retorne o número de nodos de uma lista.

```
int conta(struct nodo *inicio)
{
if (inicio==NULL) return 0;
return 1+conta(inicio->prox);
}
```

- 17) Escreva uma função recursiva que verifica se duas listas dadas são iguais, ou melhor, se têm o mesmo conteúdo e retorne:
  - 1 - Se são iguais
  - 0 - Se não são iguais

```
int compara(struct nodo *inicio1, struct nodo *inicio2)
    {
    if (inicio1==NULL && inicio2==NULL) return 1;
    if (inicio1==NULL || inicio2==NULL) return 0;
    if (strcmp(inicio1->nome,inicio2->nome)!=0) return 0;
    return compara(inicio1->prox,inicio2->prox);
    }
```

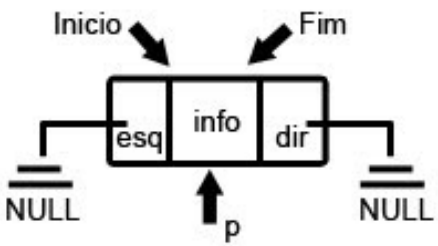
- 18) Escreva uma função que copie um vetor para uma lista encadeada. Faça duas versões: uma iterativa e uma recursiva.
- 19) Escreva uma função que copie uma lista encadeada para um vetor. Faça duas versões: uma iterativa e uma recursiva.
- 20) Escreva uma função que faça uma cópia de uma lista dada. Faça duas versões, uma iterativa e uma recursiva.

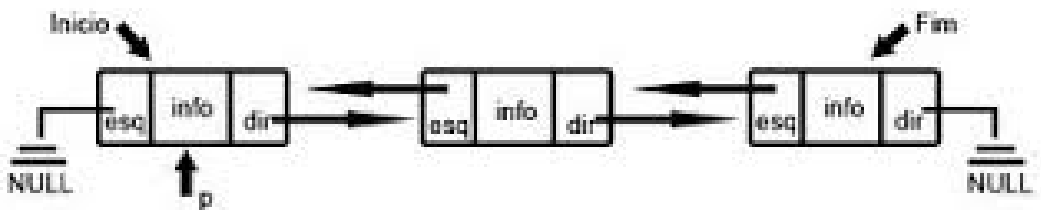
- 21) Escreva uma função que concatena duas listas encadeadas (isto é, "amarra"a segunda no fim da primeira).
- 22) Escreva uma função que inverte a ordem das células de uma lista encadeada (a primeira passa a ser a última, a segunda passa a ser a penúltima etc.). Faça isso sem usar espaço auxiliar; apenas altere os ponteiros. Dê duas soluções: uma iterativa e uma recursiva.
- 23) [Mínimo] Escreva uma função que encontre o nodo que possui o menor nome (considerando ordenação em ordem alfabética). Faça duas versões: uma iterativa e uma recursiva.

Listas duplamente encadeadas

- Podem ser utilizadas quando é necessário percorrer as listas nos dois sentidos ou é necessário fazer inserções e remoções nas duas pontas.
- Cada nodo possui um apontador para o próximo elemento da lista e um para o elemento anterior na lista.
- São mais complexas de gerenciar.

```
typedef struct nodo {char nome[10];
                     struct nodo *prox,*ant;} Tnodo;
```





Operações sobre listas duplamente encadeadas

- 1) Inserção no início da lista
- 2) Inserção no fim da lista
- 3) Listar elementos a partir do início
- 4) Listar elementos a partir do fim
- 5) Inserção em qualquer posição da lista, de modo a mantê-la ordenada
- 6) Remoção do início da lista
- 7) Remoção do último da lista
- 8) Remoção de um elemento em qualquer posição.
- Obs: Em todas as funções de inserção e remoção devem ser passados (por referência) os pointers para o inicio (primeiro nodo) e o fim (último nodo) da lista.

1 - Inserção no início de lista

```
void ins_inic(Tnodo **inicio,Tnodo **fim,char nome[])
   {
   Tnodo *ptaux;
   ptaux=(Tnodo *)malloc(sizeof(Tnodo));
   ptaux->prox=*inicio;
   ptaux->ant=NULL;
   if (*inicio!=NULL)
      (*inicio)->ant=ptaux;
   else
      *fim=ptaux;
   strcpy(ptaux->nome,nome);
   *inicio=ptaux;
   }
```

2 - Inserção no fim da lista

```
void ins_fim(Tnodo **inicio,Tnodo **fim,char nome[])
   {
   Tnodo *ptaux;
   ptaux=(Tnodo *)malloc(sizeof(Tnodo));
   ptaux->prox=NULL;
   ptaux->ant=*fim;
   if (*fim!=NULL)
      (*fim)->prox=ptaux;
   else
      *inicio=ptaux;
   strcpy(ptaux->nome,nome);
   *fim=ptaux;
   }
```

3 - Lista elementos a partir do início

```
struct nodo_d {struct nodo_d *prox, *ant; int val;};
void mostra_cresc(struct nodo_d *inic)
{
printf("Crescente\n");
while (inic)
      {
      printf("%d\n",inic->val);
      inic=inic->prox;
      }
}
```

4 - Lista elementos a partir do fim

```
void mostra_dec(struct nodo_d *fim)
{
printf("Decrescente\n");
while (fim)
      {
      printf("%d\n",fim->val);
      fim=fim->ant;
      }
}
```

5 - Inserção ordenada em duplamente encadeada

```
void insere_dupla(struct nodo **inicio, struct nodo **fim, int val)
   {
   struct nodo *ptaux=(struct nodo *)malloc(sizeof(nodo));
   ptaux->val=val;
   if (*inicio==NULL || (*inicio)->val>val)
      {
      ptaux->prox=*inicio;
      ptaux->ant=NULL;
      if (*inicio!=NULL) (*inicio)->ant=ptaux;
      else *fim=ptaux;
      *inicio=ptaux;
      return;
      }
   struct nodo *ptaux2=*inicio;
   while (ptaux2->prox!=NULL && ptaux2->prox->val<val) ptaux2=ptaux2->prox;
   ptaux->prox=ptaux2->prox;
   ptaux->ant=ptaux2;
   ptaux2->prox=ptaux;
   if (ptaux->prox==NULL) // inserção no final da lista
      (*fim)=ptaux;
   else
      ptaux->prox->ant=ptaux;
   return;
   }
```

5a - Versão Recursiva

```
void insere_dupla(struct nodo_d **inicio, struct nodo_d **fim, int val)
{
struct nodo_d *ptaux;
if (*inicio==NULL || (*inicio)->val>val)
   {
   ptaux=(struct nodo_d *)malloc(sizeof(nodo_d));
   ptaux->val=val;
   ptaux->prox=*inicio;
   if (*inicio!=NULL) // inserção no inicio da lista
      {
      ptaux->ant=(*inicio)->ant;
      (*inicio)->ant=ptaux;
      (*inicio)=ptaux;
      return;
      }
   ptaux->ant=*fim;
   (*inicio)=(*fim)=ptaux;
   return;
   }
insere_dupla(&((*inicio)->prox),fim,val);
}
```

6 - Remoção do início

```
int  remove_inic(struct nodo **inic, struct nodo **fim)
   {
   if (*inic==NULL) return 0;
   struct nodo *ptaux=*inic;
   *inic=(*inic)->prox;
   if (*inic==NULL) *fim=NULL;
   else (*inic)->ant=NULL;
   free(ptaux);
   return 1;
   }
```

7 - Remoção do fim

```
int  remove_fim(struct nodo **inic, struct nodo **fim)
   {
   if (*fim==NULL) return 0;
   struct nodo *ptaux=*fim;
   *fim=(*fim)->ant;
   if (*fim==NULL) *inic=NULL;
   else (*fim)->prox=NULL;
   free(ptaux);
   return 1;
   }
```

8 - Exclusão na posição (versão recursiva)

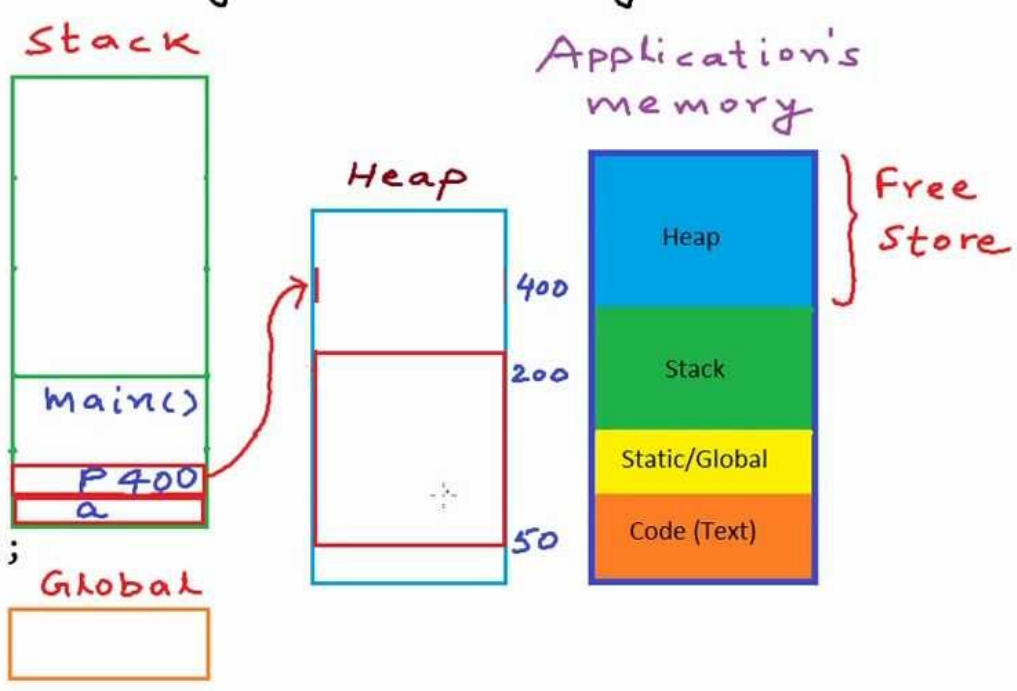
```
int exclui_dupla(struct nodo_d **inicio, struct nodo_d **fim, int val)
   {
   struct nodo_d *ptaux,*ant,*prox;
   if (*inicio==NULL) return 0; // lista vazia
   if (val==(*inicio)->val)
      {
      ptaux=*inicio;
      prox=(*inicio)->prox;
      ant=(*inicio)->ant;
      if (prox!=NULL) prox->ant=(*inicio)->ant;
      else (*fim)=(*inicio)->ant;
      if (ant!=NULL) ant->prox=(*inicio)->prox;
      else (*inicio)=(*inicio)->prox;
      free(ptaux);
      return 1;
      }
   return exclui_dupla(&((*inicio)->prox),fim,val);
   }
```

```
int main ( )
{
struct nodo_d *inic=NULL, *fim=NULL;
insere_dupla(&inic,&fim,9);
insere_dupla(&inic,&fim,7);
insere_dupla(&inic,&fim,8);
insere_dupla(&inic,&fim,11);
insere_dupla(&inic,&fim,6);
insere_dupla(&inic,&fim,5);
insere_dupla(&inic,&fim,4);
insere_dupla(&inic,&fim,10);
mostra_cresc(inic);
mostra_dec(fim);
exclui_dupla(&inic,&fim,4);
mostra_cresc(inic);
mostra_dec(fim);
exclui_dupla(&inic,&fim,7);
mostra_cresc(inic);
mostra_dec(fim);
exclui_dupla(&inic,&fim,11);
mostra_cresc(inic);
mostra_dec(fim);
system("pause");
}
```

Onde são alocadas variáveis?

- Variáveis globais, locais, parâmetros de funções e variáveis alocadas dinamicamente são alocadas em regiões diferentes da memória:
  - Variáveis locais são alocadas na pilha (stack) do sistema.
  - Variáveis globais são alocadas em um segmento de dados na carga do programa pelo sistema operacional.
    - No linux, variáveis globais inicializadas na declaração são alocadas no segmento de dados e variáveis globais não inicializadas são alocadas no segmento BSS.
    - A pilha cresce "para baixo". A cada empilhamento o stack pointer é decrementado
    - Variáveis alocadas dinamicamente são alocadas no heap (toda a memória física e virtual disponível)

```c
#include<stdio.h>
#include<stdlib.h>
int main()
{
  int a; // goes on stack
  int *p;
  p = (int*)malloc(sizeof(int));
  *p = 10;
  free(p);
 p = (int*)malloc(20*sizeof(int));
}
```



```
long long int l;
short x;
int a;
char b;
float c;

int main()
{
long long int ll;
short int xl;
int al;
char bl;
float cl;
printf("&l=%08x\n",&l);
printf("&x=%08x\n",&x);
printf("&a=%08x\n",&a);
printf("&b=%08x\n",&b);
printf("&c=%08x\n",&c);
printf("&ll=%08x\n",&ll);
printf("&xl=%08x\n",&xl);
printf("&al=%08x\n",&al);
printf("&bl=%08x\n",&bl);
printf("&cl=%08x\n",&cl);
system("pause");
}
```

```
&l=00430030
&x=00430038
&a=0043003c
&b=00430040
&c=00430044

&ll=0022fe48
&xl=0022fe46
&al=0022fe40
&bl=0022fe3f
&cl=0022fe38
```

```
typedef struct nodo {int valor; struct nodo *prox;} Tnodo;
Tnodo *inicg=NULL;
Tnodo *inic2;

void ins_inic(Tnodo **einic, int valor)
{
Tnodo *ptaux=(Tnodo *)malloc(sizeof(Tnodo));
ptaux->valor=valor;
ptaux->prox=*einic;
*einic=ptaux;
printf("␣ptaux␣=␣%08x\n",ptaux);
printf("&ptaux␣=␣%08x\n",&ptaux);
printf("&ptaux->valor␣=␣%08x\n",&ptaux
printf("&ptaux->prox␣=␣%08x\n",&ptaux-
printf("einic=%08x\n",einic);
printf("&einic=%08x\n",&einic);
printf("*einic=%08x\n",*einic);
printf("**einic=%08x\n",**einic);
}

int main()
{
Tnodo *inicio=NULL;
printf("␣inicg␣=␣%08x\n",inicg);
printf("&inicg␣=␣%08x\n",&inicg);
printf("␣inic2␣=␣%08x\n",inic2);
printf("&inic2␣=␣%08x\n",&inic2);
printf("␣inicio␣=␣%08x\n",inicio);
printf("&inicio␣=␣%08x\n",&inicio);
ins_inic(&inicio,10);
system("pause");
}
```

```
inicg  = 00000000
&inicg  = 00430030
inic2  = 00000000
&inic2  = 00430038
inicio  = 00000000
&inicio  = 0022fe48

ptaux  = 003476c0
&ptaux  = 0022fe08
&ptaux->valor  = 003476c0
&ptaux->prox  = 003476c8
einic=0022fe48
&einic=0022fe20
*einic=003476c0
**einic=0022fdf0
```

- Faça uma função que receba duas listas duplamente encadeadas l1 e l2, cada lista representando um número com uma quantidade ilimitada de dígitos, cada nodo contendo um dígito, e efetue a soma dos dois números gerando uma terceira lista l3 com o resultado. Use essa função em um programa que leia dois números grandes (mínimo 20 dígitos) e efetue a soma dos dois.
- Faça uma função que receba duas listas duplamente encadeadas l1 e l2, cada lista representando um número com uma quantidade ilimitada de dígitos, cada nodo contendo um dígito, e some o número em l2 ao número em l1, deixando o resultado da soma em l1.

- Idem, para a subtração dos dois números (l1-l2). Assuma que o número em l1 é maior.
- Idem para comparação entre os dois números, retornando:
  - -1 - se l1<l2
  - 0 - se l1=l2
  - 1 - se l1>l2
- Faça uma função que receba um inteiro N entre 0 e 9 e uma lista duplamente encadeada M representando um número com uma quantidade ilimitada de dígitos, um dígito por nodo, e gere uma segunda lista P com o resultado do produto de M por N, um dígito por posição.
- Faça uma função que receba duas listas duplamente encadeadas l1 e l2, cada lista representando um número com uma quantidade ilimitada de dígitos, cada nodo contendo um dígito, e efetue a multiplicação dos dois números gerando uma terceira lista l3 com o resultado. Use essa função em um programa que leia dois números grandes (mínimo 20 dígitos) e efetue a multiplicação dos dois.

- Faça um programa que leia 10 números, cada número com uma quantidade ilimitada de dígitos, e ordene-os em ordem crescente.
- Faça um programa que gere uma lista com os primeiros 100.000 números primos. Para testar se um número é primo, verifique se é divisível por algum dos números já presentes na lista.
- Faça um programa que crie uma lista simplesmente encadeada com um milhão de elementos inseridos no final. Faça uma versão recursiva e uma não recursiva. Compare o desempenho das duas.

- Medição de tempo em trechos de código pode ser feita utilizando a função time( ) da biblioteca time.h, que retorna o número de segundos transcorridos desde 1/1/1970.
- Pode ser utilizada da seguinte forma:

```c
#include <time.h>
#include <stdlib.h>
#include <stdio.h>

int main(void)
{
time_t t0,t1;
t0=time(NULL);

// trecho de código a ser medido

t1 = time(NULL);
printf("tempo:%d\n",t1-t0);
}
```

Atividades no site hackerrank.com

- Acesse o site: https://www.hackerrank.com/
- Efetue um cadastro no site ("sign up", canto superior direito da tela)
- Selecione a categoria de problemas de listas encadeadas (Domains -> Data Structures -> Linked Lists)
- Resolva todos os exercícios de listas encadeadas
- Observações:
  - Nos exercícios de listas encadeadas é necessário somente desenvolver a função pedida. Não é necessário desenvolver o programa inteiro.
  - Os parâmetros de início e fim da lista são passados por valor, e seu valor atualizado (quando ocorrer alteração do início ou fim da lista) são retornados.

- Por exemplo, a função para remoção de um nodo em uma dada posição (posição 0 representa remoção do primeiro nodo da lista) ficaria:

```
Node* Delete(Node *head, int position)
   {
   if (position==0)
      {
      Node *ptaux=head->next;
      free(head);
      return ptaux;
      }
   head->next=Delete(head->next,position-1);
   return head;
   }
```

Arquivos

- São conjuntos de dados armazenados em meio não volátil (normalmente disco). O acesso a um arquivo é sempre feito através de uma variável do tipo arquivo (FILE *).
- Ex: FILE *arqent, *arqvotos;
- Leitura e escrita em arquivos texto.
- O acesso a arquivos em C é feito através de variáveis do tipo "descritores de arquivo", que são structures que mantém a informação sobre arquivos em utilização. O acesso a um descritor de arquivo é feito através de um apontador para ele, declarado no programa através do tipo FILE * (stdlib.h)
- Ex: FILE *arqin,*arqout;

Tratamento de Arquivos

- Para utilizar um arquivo, inicialmente deve-se associar a variável arquivo ao arquivo em disco. Isso é feito com o comando FOPEN.

  arquivo=fopen(Nome do arquivo, modo de abertura)

- E os principais modos de abertura são:
  - "wt- escrita texto
  - "rt- leitura texto
- Ex.

  arqent=fopen("c:\\notas","wt");

- Para gravar um dado em um arquivo texto, usa-se o comando fprintf, que funciona como um printf, apenas especificando como 1º parâmetro o arquivo onde será escrito o dado.
- Ex: fprintf(arquivo,"%d %d\n",a,b);
- Antes de terminar o programa, todos os arquivos devem ser fechados (fclose(arqout))

Exercícios:

1) Faça um programa que crie um arquivo numeros.dat na raiz do drive C ("C:\\"), com os números de 1 a 1000.

2) Faça um programa que leia 20 números do teclado e escreva os números pares no arquivo pares.dat e os ímpares no arquivo impares.dat, ambos na raiz do drive C.

3) Faça um programa que escreva os 100 primeiros números primos em um arquivo primos.dat.

Leitura de arquivos texto

- A leitura de um arquivo texto é feita atraves do comando
- fscanf(arquivo,especificador_de_tipo,lista_de_variaveis).
- Ex:

```
fscanf(arqin,"%d",&voto)
```

A função fscanf retorna o número de elementos lidos. Em caso de fim de arquivo, ela retorna EOF. Assim, para ler um arquivo até o fim pode-se fazer:

```
while(fscanf(arqin,"%d",&num)!=EOF)
   printf("%d\n",num);
```

- Outra função útil para a leitura de arquivos é a função feof(arquivo), que retorna verdadeiro se já foi encontrado o fim do arquivo.
- Uma forma de utilizá-lo é:

```
fscanf(arq,"%d",&num)
while (feof(arq)==FALSE)
    {
    fscanf(arq,"%d",&num);
    printf("%d\n",num);
    }
```

- A leitura de um arquivo texto pode ser feita linha a linha, com o comando fgets (string, tamanho, arquivo). A função fgets retorna o apontador para o string lido (mesmo string que foi enviado como primeiro parâmetro), ou NULL se o arquivo chegou ao fim. O caracter '\n' da quebra de linha é mantido no final da string e normalmente deve ser removido.
- Para ler um arquivo inteiro a função pode ser usada assim:

```
while (fgets(linha,200,arqin))
   {
   // processa a linha lida
   }
```

1) Faça um programa que leia o arquivo resumo.txt contendo 1000 filmes no formato descrito na especificação do trabalho, e insira cada filme em uma lista ordenada de filmes, e cada ator em uma lista ordenada de atores. Liste ao final as duas listas.

Funçoes para manipulação de strings (colocar no início #include < *string.h* >)

- **strcmp**(str1,str2) - compara dois strings em ordem lexicográfica e retorna:
  - 0 - se os strings são iguais
  - $\leq$0 - se str1 é menor que str2 (considerando ordem lexicográfica)
  - $\geq$0 - se str1 é maior que str2 (considerando ordem lexicográfica)

ex:

```
if (strcmp(nome1,nome2)<0)
    printf("%s",nome1)
else printf("%s",nome2);
```

- 2) Faça um programa que leia dois nomes e escreva-os em ordem alfabética.
- Pode-se declarar uma lista de strings como uma matriz de caracteres:
  - char nomes[20][15]; // declara uma lista de 20 strings, cada um com até 14 caracteres (e mais um espaço para o delimitador de fim de string)
- Pode-se inicializar um string na declaração:

  char nome[20]="João";
- também pode-se inicializar uma lista de strings:

  char nomes[5][15]={"João","Maria","Zefa","Juca","Jeca"};
  char unidades[10][10]={"zero","um","dois","tres","quatro",
  "cinco","seis","sete","oito","nove"};

- **strcpy**(destino, origem) - copia o string de origem para o string de destino.

  Ex:

  ```
  char nome[20];
  strcpy(nome,"João"); // copia "João"para o string nome
  ```

- **strlen**(str) - retorna o tamanho do string em str.

  Ex:

  ```
  a=strlen("João"); // a variável "a"recebe o valor 4
  ```

- **strcat**(str1,str2) - concatena uma cópia de str2 no final de str1.

  Ex:

  ```
  strcpy(st1,"João");
  strcat(st1,"da Silva");//st1 ficará com "João da Silva"
  ```

- char * itoa (int value, char * str, int base); - Converte um inteiro para um string (stdlib.h, mas não é padrão de C)
- int atoi(const char *str) - converte um string para um inteiro
- sscanf - le um valor a partir de um string, no mesmo formato que o scanf. Ex:
  - char linha[ ]="123 23.2 Joao";
  - sscanf(linha,"%d%f%s",&x,&y,z);
- sprintf - um printf, só que a saída é colocada em um string
  - sprintf(linha,"x=%d y=%f z=%s\n",x,y,z);

- char * strstr(char *st1, char *st2) - verifica se o segundo string ocorre no primeiro.
- long int strtol(const char *str, char **endptr, int base) - converte a parte inicial de um string em um valor long, e atualiza o pointer endptr para apontar para o próximo caracter após a parte convertida (está na stdlib).
  - char linha[]="12 15 18",*pl=linha;
  - printf("%d\n",strtol(pl,&pl,10));
  - printf("%d\n",strtol(pl,&pl,10));
  - printf("%d\n",strtol(pl,&pl,10));

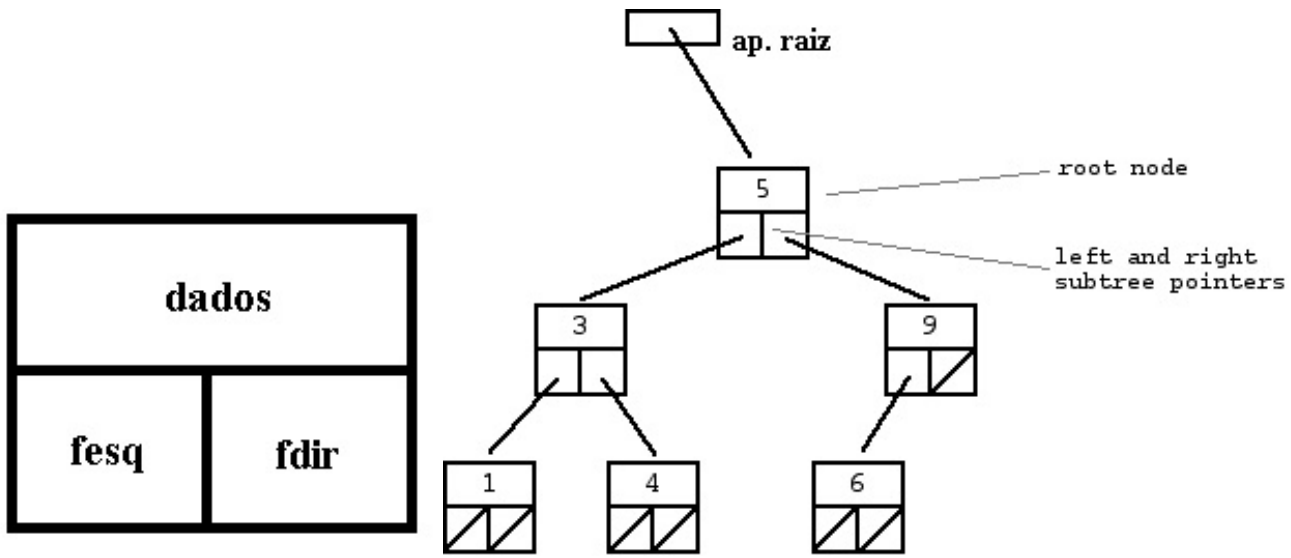
- char *strtok (char *st,char delimitadores) - retorna o substring que ocorre até o primeiro delimitador da lista de delimitadores. Ex: strtok("123+14","+-*") retorna o substring "123", que é o substring até o primeiro delimitador da lista "+-*". No caso, o substring até a primeira ocorrência de '+'.
- As chamadas subsequentes de strtok devem passar NULL como primeiro parâmetro. Quando não houver mais substrings que atendam à condição, a função retorna NULL;
- A cada chamada a função coloca '\0' no delimitador atualizando um pointer estático para a posição seguinte, preparando para a próxima chamada.
- Ex:

```
char linha[]="Joao da Silva,Maria da Silva,Jose da Silva";
char *p=strtok(linha,",");
while (p)
    {
    printf("%s\n",p);
    p=strtok(NULL,",");
    }
```

Árvores Binárias

- Uma árvore binária é uma estrutura de dados hierárquica em que cada nodo possui dois apontadores, um para a subárvore à sua esquerda e um para a subárvore à direita.

  struct nodo {int dado; struct nodo *fesq, *fdir;};

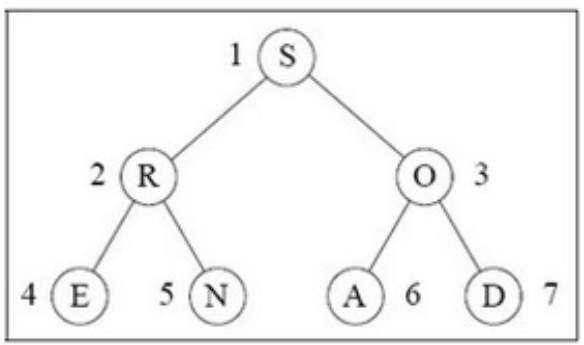


- Uma árvore **binária**:
  - Ou não tem elemento algum (árvore vazia).
  - Ou tem um elemento distinto, denominado raiz, com dois ponteiros para duas estruturas diferentes, denominadas sub-árvore esquerda e sub-árvore direita.
- O **grau** de um nó é o número de filhos do nó.
- Os nós de uma árvore binária possuem graus zero, um ou dois.
- Um nó de grau zero é denominado **folha**.

Representação de árvore binária por vetor

- Uma das maneiras mais simples de representar árvores binárias em linguagens de programação é por meio de arranjos unidimensionais (vetores).
- Caso a raiz esteja na posição zero, dado um nó de índice i qualquer, os seus filhos terão índices 2i+1 e 2i+2 e o seu pai terá índice $\left\lfloor \frac{i-1}{2} \right\rfloor$.
- Caso a raiz esteja na posição um, os filhos terão índices 2i e 2i+1 e o pai terá índice $\left\lfloor \frac{i}{2} \right\rfloor$.

1	2	3	4	5	6	7
S	*R*	*O*	*E*	*N*	*A*	*D*



- Essa implementação é utilizada para representar árvores completas ou quase completas.
- Heaps, que são árvores binárias quase completas são implementadas na forma de um vetor de uma maneira bastante eficiente.
- Apesar da simplicidade, essa representação requer uma grande quantidade de memória contígua para o armazenamento de árvores grandes, o crescimento desta é díficil de implementar e manter e pode haver grande quantidade de desperdício de memória.

- Normalmente implementa-se árvores utilizando estruturas alocadas dinamicamente em memória. Em C, um nodo com um campo de informação inteira e apontadores para a subárvore esquerda e direita pode ser declarado como:
- struct nodo {int valor; struct nodo *fesq,*fdir;}

Inserção em árvore binária de busca

- As operações sobre árvores normalmente são implementadas de forma recursiva. Uma função para inserção em uma árvore de pesquisa ordenada poderia ser:

```
void inserir(struct No **pRaiz, int numero){
    if (*pRaiz == NULL){
        * pRaiz = (struct No *) malloc(sizeof(struct No));
        (*pRaiz)->pEsquerda = NULL;
        (*pRaiz)->pDireita = NULL;
        (*pRaiz)->numero = numero;
        }
    else if(numero <(*pRaiz)->numero)
            inserir(&(*pRaiz)->pEsquerda, numero);
        else
            inserir(&(*pRaiz)->pDireita, numero);
}
```

- Uma função de inserção usando o padrão utilizado no hackerrank, em que o pointer para a raiz é passado por valor e atualizado no retorno da função:

```
node * insert(node * root, int value)
    {
    if (root==NULL)
        {
        root=(node *)malloc(sizeof(node));
        root->left=root->right=NULL;
        root->data=value;
        return root;
        }
    if (value>root->data) root->right=insert(root->right,value);
    if (value<root->data) root->left=insert(root->left,value);
    return root;
    }
```

- O percurso pelos nodos de uma árvore binária pode ser feito em três ordens:
  - In-ordem: Visita-se a subárvore da esquerda, a raiz, e depois a subárvore da direita.
  - Pré-ordem: Visita-se primeiro a raiz, depois a subárvore da esquerda, e depois a subárvore da direita.
  - Pós-ordem: Visita-se primeiro a subárvore da esquerda, depois a subárvore da direita e por fim a raiz.

```
void in_ordem(struct nodo *raiz)
   {
   if (raiz==NULL) return;
   in_ordem(raiz->esq);
   printf("␣%d␣",raiz->valor);
   in_ordem(raiz->dir);
   }
```

```
void pre_ordem(struct nodo *raiz)
   {
   if (raiz==NULL) return;
   printf("␣%d␣",raiz->valor);
   pre_ordem(raiz->esq);
   pre_ordem(raiz->dir);
   }
```

- 1) Faça uma função para inserção de valores em uma árvore binária. Use-a para inserir 10 valores.
- 2) Implemente funções para percorrer a árvore nas três ordens vistas, mostrando o conteúdo de cada nodo.
- A **profundidade** de um nó é a distância deste nó até a raiz.
  - A profundidade da raiz é 0.
  - Um conjunto de nós com a mesma profundidade é denominado nível da árvore.
  - A maior profundidade de um nó, é a **altura** da árvore.
  - Uma árvore que tem somente a raiz tem altura igual a 0.
  - Uma árvore que não tem nem a raiz tem altura igual a -1.

- 3) Faça uma função recursiva que receba a raiz de uma árvore e retorne a altura da árvore.
- 4) Uma árvore **estritamente binária** é uma árvore na qual todo nó tem zero ou duas folhas. Faça uma função recursiva que receba a raiz de uma árvore e retorne:
  - 1 : Se ela é estritamente binária
  - 0 : Em caso contrário
- 5) Faça uma função recursiva que receba a raiz de uma árvore e retorne a quantidade de nodos
- 6) Faça uma função recursiva que receba a raiz de uma árvore e retorne a soma dos nodos.
- 7) Faça uma função recursiva que receba a raiz de uma árvore e retorne o maior valor na árvore.

- 8) Faça uma função que receba a raiz de uma árvore e o número de um nível e liste todos os nodos daquele nível.
- 9) Faça uma função que receba a raiz de uma árvore e liste todos os nodos em ordem de nível (primeiro a raiz, depois os de nível 1 e assim por diante)

Percurso de árvore binária para um nível específico

```
void lista_nivel(Tnodo *raiz, int nivel)
   {
   if (raiz==NULL || nivel<0) return;
   if (nivel==0) {printf("␣%d␣",raiz->nivel);return;}
   lista_nivel(raiz->fesq,nivel-1);
   lista_nivel(raiz->fdir,nivel-1);
   }
```

Percurso de árvore binária por níveis

```
void lista_nivel(Tnodo *raiz)
{
Tnodo *Fila[100];
int IF=0,FF=1;
Fila[0]=raiz;
while (IF<FF)
   {
   Tnodo *aux=Fila[IF++];
   printf("%d",aux->valor);
   if (aux->fesq!=NULL) Fila[FF++]=aux->fesq;
   if (aux->fdir!=NULL) Fila[FF++]=aux->fdir;
   }
}
```

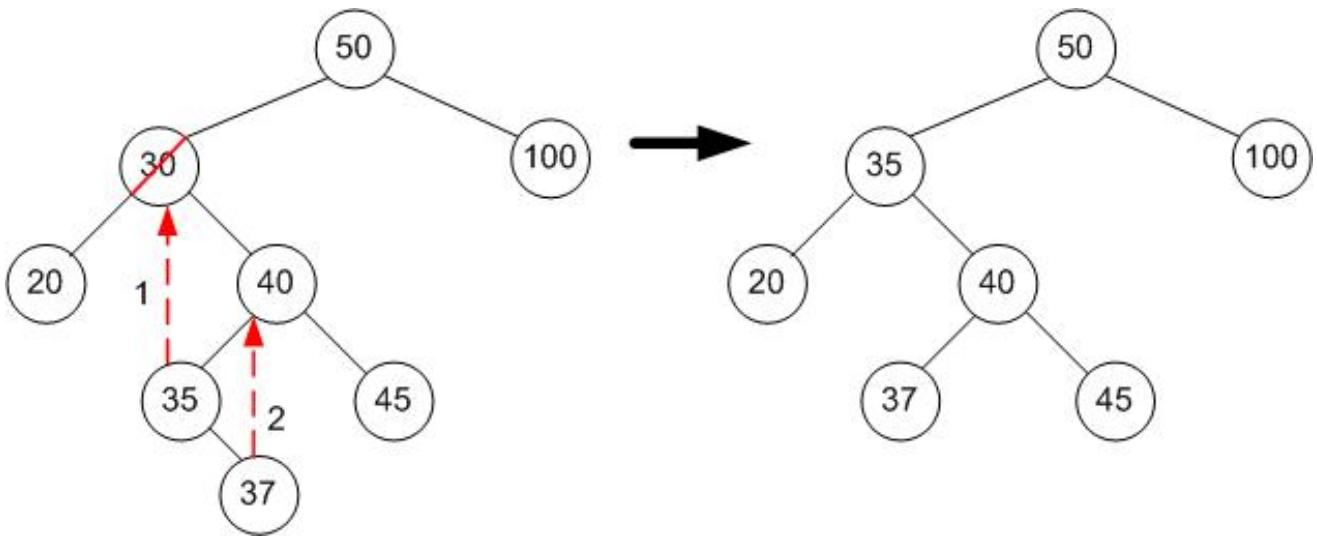
- 10) Faça uma função recursiva que receba a raiz de uma árvore e um valor, e pesquise o valor na árvore, retornando o pointer para o nodo onde se encontra o valor, e NULL, caso o valor não exista na árvore.
- 11) Escreva uma função que verifique se uma dada árvore binária é ou não é de busca retornando 1 - Se ela é uma árvore de busca; 0 - Se não é.
- 12) Escreva uma função que transforme um vetor crescente em uma árvore de busca balanceada.
- 13) Escreva uma função que transforme uma árvore de busca em um vetor crescente. Comece por alocar dinamicamente o vetor.

Exclusão

- A exclusão de um nó é um processo mais complexo. Para excluir um nó de uma árvore binária de busca, há de se considerar três casos distintos para a exclusão:
  - Exclusão na folha: A exclusão na folha é a mais simples, basta removê-lo da árvore.
  - Exclusão de um nó com um filho: Excluindo-o, o filho sobe para a posição do pai.
  - Exclusão de um nó com dois filhos

Exclusão de um nó com 2 filhos

- Neste caso, pode-se operar de duas maneiras diferentes. Pode-se substituir o valor do nó a ser retirado pelo valor sucessor (o nó mais à esquerda da subárvore direita) ou pelo valor antecessor (o nó mais à direita da subárvore esquerda), removendo-se aí o nó sucessor (ou antecessor).
- A figura a seguir elimina o nodo com o valor 30, substituindo-o pelo seu sucessor.)



- 14) Faça uma função que receba o apontador raiz de uma árvore binária de busca e um valor, e remova o nodo com o valor. Assuma que o nodo a ser removido tem grau 0 ou 1.
- 15) Idem, mas considerando a possibilidade de o nodo ter grau 2.

Remoção da Raiz

- Recebe uma árvore não vazia r. Remove a raiz da árvore, rearranja a árvore de modo que ela continue sendo de busca.

```
void removeraiz( struct no **r) {
struct no *p, *q;
if ((*r)->esq == NULL) {
      q = (*r)->dir;
      free(*r);
      *r=q;
      return
      }
p = *r; q = (*r)->esq;
while (q->dir != NULL) {
      p = q; q = q->dir;
      }
// q é o nó anterior a r na ordem e-r-d
// p é o pai de q
if (p != (*r)) {
      p->dir = q->esq;
      q->esq = (*r)->esq;
      }
q->dir = (*r)->dir;
free( *r);
*r=q;
return;
}
```

E se o valor a remover não está na raiz?

```
void remove(struct no **raiz, int valor)
    {
    if (*raiz==NULL) return;
    if (valor==(*raiz)->valor) removeraiz(raiz);
    else if (valor<(*raiz)->valor)
                remove(&(*raiz)->esq,valor);
        else remove(&(*raiz)->dir,valor);
    }
```

Distância entre dois strings

```
int minimum(int a, int b, int c)
{
if (a<=b && a<=c) return a;
if (b<=c) return b;
return c;
}
int LevenshteinDistance ( char s [],char t [])
   {
   int d[100][100];
   int m=strlen(s);
   int n=strlen(t);
   int i,j;
   for (i=0;i<=m;i++)
      d[i][0] = i; // deletion
   for (j=0;j<=n;j++)
      d[0][j] = j; // insertion
   for (j=1;j<=n;j++)
      for (i=1;i<=m;i++)
         if (s[i-1] == t[j-1])
            d[i][j] = d[i-1][j-1];
         else
            d[i][j] = minimum (
   d[i -1][j] + 1, // deletion
   d[i][j-1] + 1, // insertion
   d[i-1][j-1] + 1 // substitution
            );
   return d[m][n];
}
```

Árvores AVL (Balanceadas)

- Definição : A **altura** de uma árvore binária é o nível máximo de suas folhas (profundidade)
- Uma **árvore binária balanceada** (AVL) é uma árvore binária na qual as alturas das duas subárvores de todo nó nunca difere em mais de 1.
- O balanceamento de um NÓ é definido como a altura de sua subárvore esquerda menos a altura de sua subárvore direita.

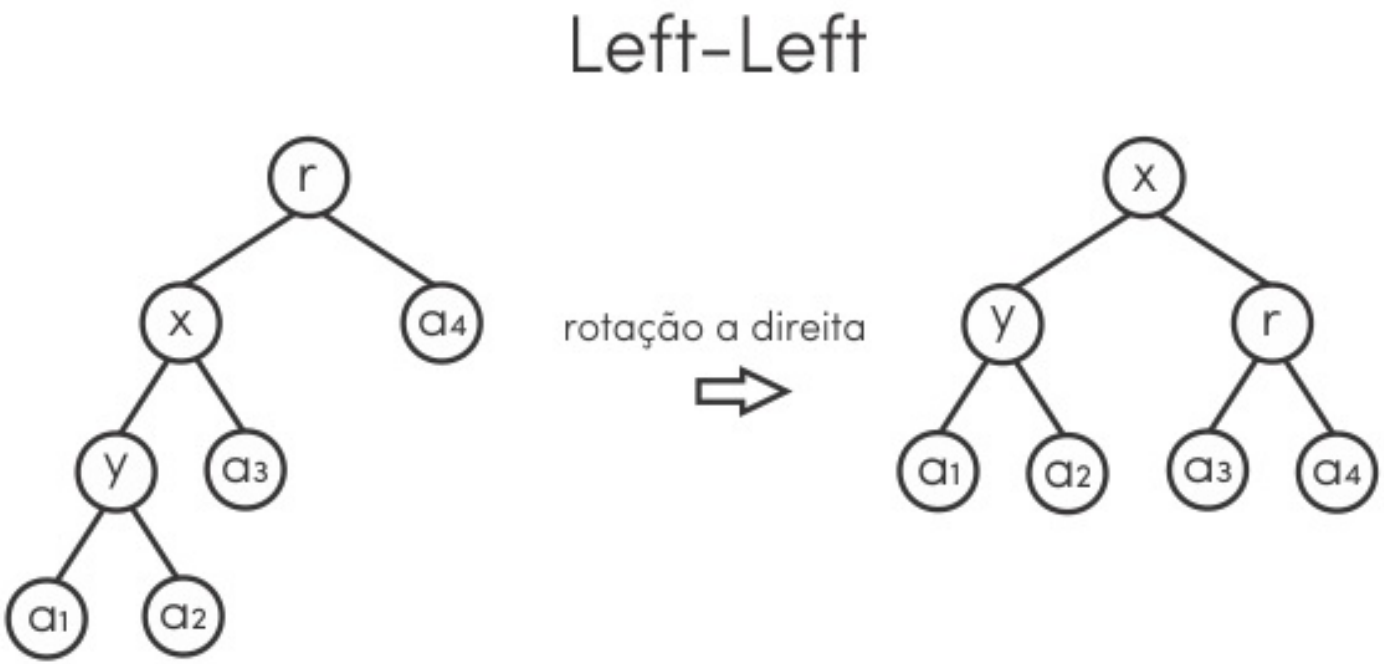
- Cada nó numa árvore binária balanceada (AVL) tem balanceamento de 1, -1 ou 0.
- Se o valor do balanceamento do nó for diferente de 1, -1 e 0. Essa árvore não é balanceada (AVL).
- Se a probabilidade de pesquisar um dado for a mesma para todos os dados, uma árvore binária balanceada determinará a busca mais eficiente.
- Mas os algoritmos de inserção e remoção já vistos até agora não garantem que essa árvore permanecerá balanceada.
- Para manter uma árvore balanceada, é necessário fazer uma transformação na árvore tal que:
  - 1. o percurso em ordem da árvore transformada seja o mesmo da árvore original (isto é, a árvore transformada continue sendo um árvore de busca binária);
  - 2. a árvore transformada fique balanceada.

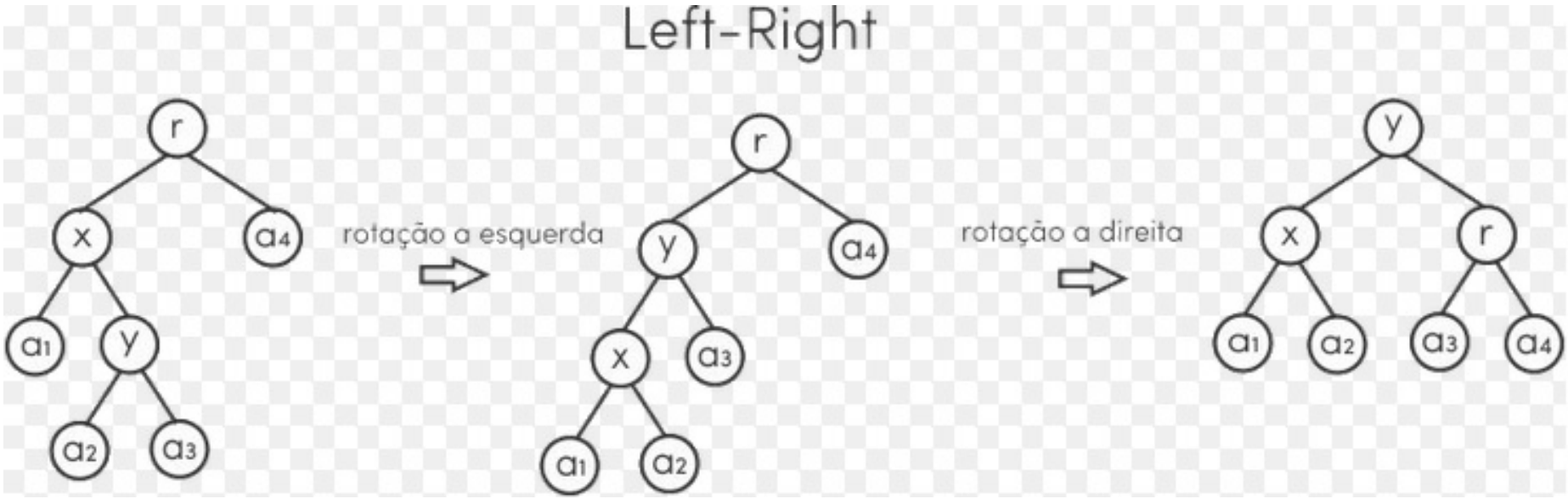
Operação de Rotação

- A transformação a ser feita na árvore tal que ela se mantenha balanceada é chamada de rotação.
- A rotação poderá ser feita à esquerda ou à direita dependendo do desbalanceamento que tiver que ser solucionado.
- A rotação deve ser realizada de maneira que as regras 1 e 2 sejam respeitadas.
- Dependendo do desbalanceamento a ser solucionado, apenas uma rotação não será suficiente para resolvê-lo.
- O desbalanceamento deve ser verificado e corrigido após cada inserção. Se a árvore estiver com desbalanceamento maior do que 1 desmonta-se a árvore e se constrói uma nova balanceada.

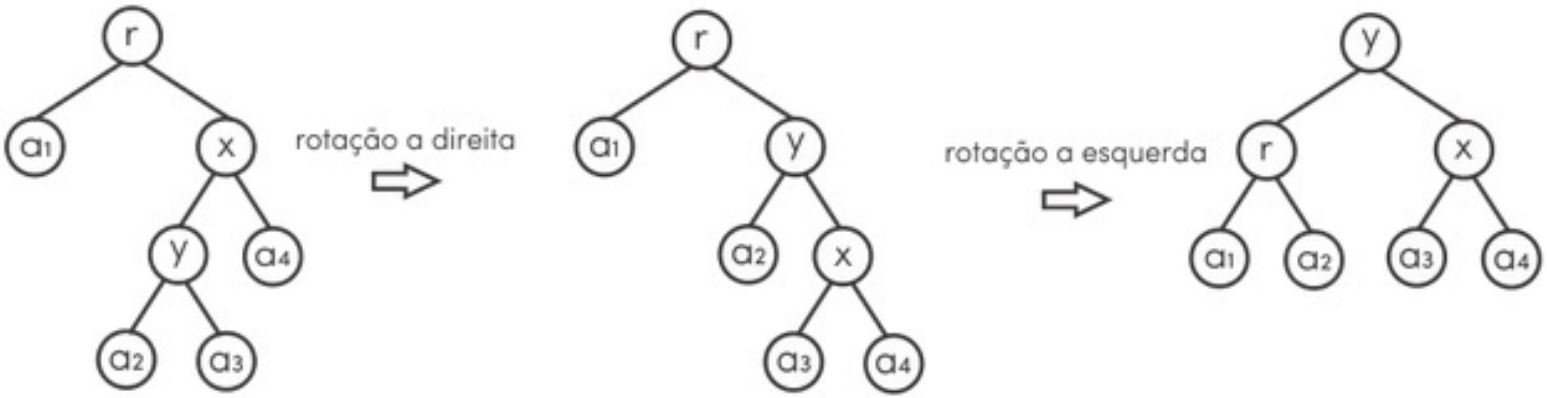
- Para o rebalanceamento da árvore é necessário calcular o Fator de Balanceamento para verificar qual rotação deve ser efetuada a fim de rebalanceá-la.
- FB = altura da subárvore esquerda - altura da subárvore direita
  - Se FB é negativo, as rotações são feitas à esquerda.
  - Se FB é positivo, as rotações são feitas à direita.
- Há dois tipos de ocorrências nos casos de balanceamento:
  - Caso 1: Nó raiz com FB 2 ou -2 com um filho (na direção de onde houve a inserção) com FB 1 ou -1 com o mesmo sinal, neste caso a solução é uma rotação simples.
  - Caso 2: Nó raiz com FB 2 ou -2 com um filho (na direção de onde houve a inserção) com FB -1 ou 1 os quais possuem sinais trocados, neste caso a solução é uma rotação dupla.
- Primeiro rotaciona-se o nó com fator de balanceamento 1 ou -1 na direção apropriada e depois rotaciona-se o nó cujo fator de balanceamento seja 2 ou -2 na direção oposta.

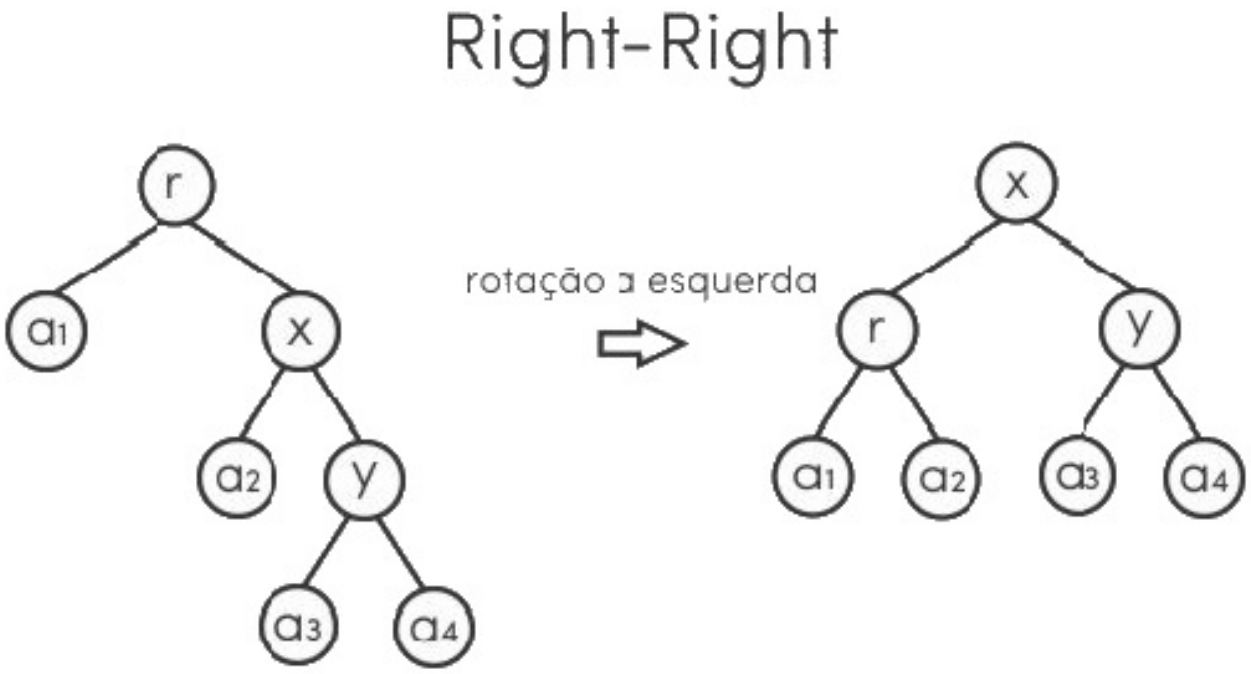
- Animação:
  https://www.cs.usfca.edu/ galles/visualization/AVLtree.html
- Desbalanceamento do pai e do filho são na direção esquerda (left-left). Correção é a rotação do pai à direita:




Left-Right
r
x
a4
a1
y
a2
a3
rotação a esquerda
r
y
a4
x
a3
a1
a2
rotação a direita
y
x
r
a1
a2
a3
a4

Right-Left


r
a1
x
y
a4
a2
a3
rotação a direita
r
a1
y
a2
x
a3
a4
rotação a esquerda
y
r
x
a1
a2
a3
a4


Right-Right
r
a1
x
a2
y
a3
a4
rotação à esquerda
x
r
y
a1
a2
a3
a4

- Funções para verificar balanceamento:

```
int altura(struct nodo *raiz)
{
if (raiz==NULL) return -1;
int altesq=altura(raiz->fesq);
int altdir=altura(raiz->fdir);
if (altesq>altdir) return altesq+1;
else return altdir+1;
}

int fb(struct nodo *raiz)
{
if (raiz==NULL) return 0;
return altura(raiz->fesq)-altura(raiz->fdir);
}
```

```
struct nodo* rot_esq(struct nodo *parent)
{
struct nodo *node1;
node1=parent->fdir;
parent->fdir = node1 ->fesq;
node1->fesq= parent;
return node1;
}
```

- Caso direita-esquerda - Se o fator de balanceamento da raiz é -2, então a subárvore da direita é mais alta que a subárvore da esquerda e o fator de balanceamento da subárvore da direita deve ser verificado.
- Se o fator de balanceamento da subárvore da direita é 1, então duas rotações diferentes são neessárias. A primeira rotação é uma rotação para a direita sobre a raiz da subárvore à direita. A segunda é uma rotação à esquerda sobre a raiz.
- O código para a rotação direita-esquerda é mostrado a seguir:

```
struct nodo* rot_dir_esq(struct nodo *parent)
{
struct nodo *node1;
node1=parent ->fdir;
parent->fdir = rot_dir(node1);
return rot_esq(parent);
}
```

- Caso esquerda-esquerda: se o fator de balanceamento da raiz é +2, então a subárvore esquerda é mais alta do que a direita e o fator de balanceamento da subárvore da esquerda deve ser verificado. Se o fator de balanceamento da subárvore da esquerda é +1 então uma rotação simples à direita sobre a raiz é necessária.

```
struct nodo* rot_dir(struct nodo *parent)
{
struct nodo *node1;
node1 = parent ->fesq;
parent ->fesq = node1 ->fdir;
node1 ->fdir = parent;
return node1;
}
```

- Caso esquerda-direita : Se o fator de balanceamento da raiz é +2, então a subárvore da esquerda é mais alta do que a direita e o fator de balanceamento da subárvore da esquerda deve ser verificado. Se o fator de balanceamento da subárvore da esquerda é -1, duas rotações são necessárias. A primeira rotação é uma rotação sobre a raiz da subárvore da esquerda. A segunda é uma rotação à direita sobre a raiz da árvore.

```
struct nodo* rot_esq_dir(struct nodo *parent)
   {
   struct nodo *node1;
   node1= parent ->fesq;
   parent->fesq = rot_esq(node1);
   return rot_dir(parent);
   }
```

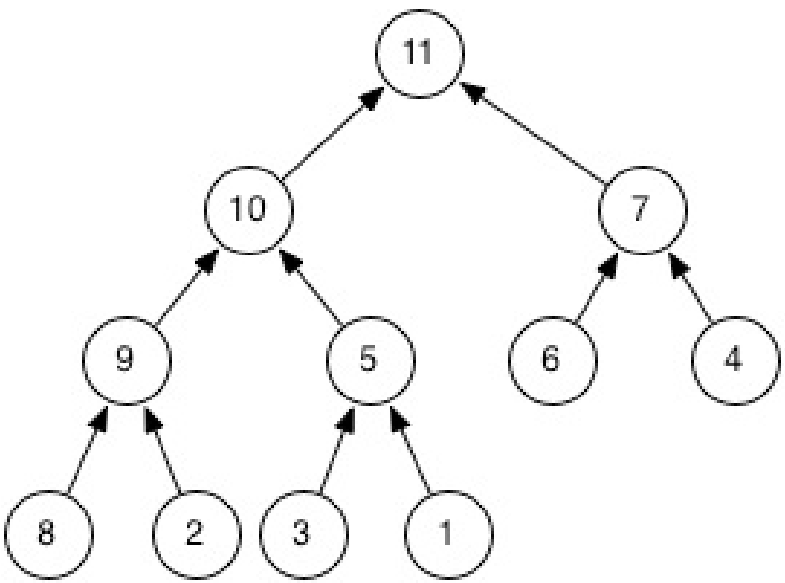
```
struct nodo* balancing(struct nodo *node)
   {
   int f_b= fb(node);
   if (f_b >1) {
      if (fb(node->fesq) >0)
         node=rot_dir(node);
      else
         node=rot_esq_dir(node);
      }
   else if(f_b < -1) {
            if(fb(node->fdir) >0)
               node=rot_dir_esq(node);
            else
               node=rot_esq(node);
            }
   return node;
   }
```

```
struct nodo* insere(struct nodo *root,int val)
   {
   if (root==NULL) {
      root = (struct nodo*) malloc(sizeof(struct nodo));
      root -> num = val;
      root -> fesq = NULL;
      root -> fdir = NULL;
      return root;
      }
   else if (val < root->num) {
            root->fesq = insere(root->fesq, val);
            root=balancing(root);
            }
        else if (val > root->num) {
                 root->fdir = insere(root->fdir, val);
                 root=balancing(root);
                 }
   return root;
   }
```

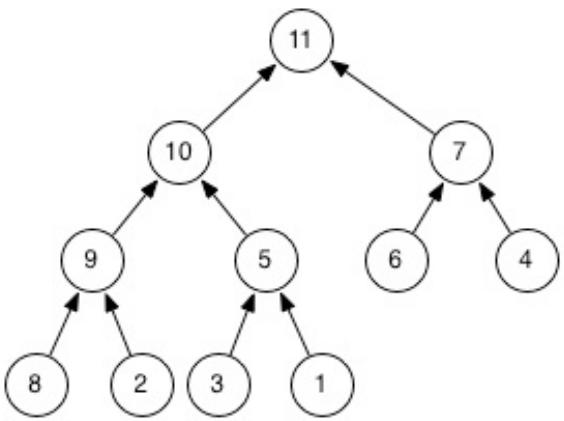
```
int main()
   {
   struct nodo *raiz = NULL;
   int i;
   for (i=0;i<20;i++)
      {
      raiz=insere(raiz,i);
      print_t(raiz);
      }
   system("pause");
   }
```

Heaps, priority queues e Heapsort

- Um **heap** é uma estrutura de dados organizada como árvore binária em que todos os níveis até o penúltimo estão completos, e o último nível é preenchido "em ordem", da esquerda para a direita.
- Além disso, o valor da chave em cada nodo deve ser maior (maxheap) ou menor (minheap) que o valor da chave nos dois filhos, e a ordem da chave entre os filhos não interessa.



- Um heap pode ser implementado sobre um vetor, onde cada elemento v[i] é pai dos elementos nas posições v[2*i+1] e v[2*i+2] e a raiz da árvore ocupa a primeira posição (v[0]).
- Assim, dado o índice de um nodo no vetor, os índices do pai e dos filhos à esquerda e à direita podem ser obtidos por:
  - Pai(i) : (i-1)/2
  - Esquerda(i) : i * 2 + 1
  - Direita(i) : i * 2 + 2
- Assim, o heap abaixo poderia ser representado pelo vetor [11 10 7 9 5 6 4 8 2 3 1]



- As principais operações sobre heaps são:
  - cria-heap: cria um heap vazio
  - heapify: cria um heap a partir de um conjunto de elementos
  - busca-maior (em um heapmax) ou busca-menor(em um heapmin)
  - remove-maior: remove a raiz de um heapmax
  - altera-valor-chave
  - insere: adiciona elemento a um heap

Inserção em Heap

- Pode ser implementada adicionando o novo elemento no final do heap e depois "empurrando-o"para cima enquanto ele for maior que o pai (no caso de um Heap Max) até alcançar sua posição.
- Qual o custo?

```
void corrigeSubindo(int index)
{
// Se index não é a raiz e o valor do index for maior do que o valor de seu pai,
// troca seus valores (index e pai(index)) e corrige para o pai
if (index > 0 && heap[index] > heap[pai(index)])
{
troca(heap[index], heap[pai(index)]);
corrigeSubindo(pai(index));
}
}
```

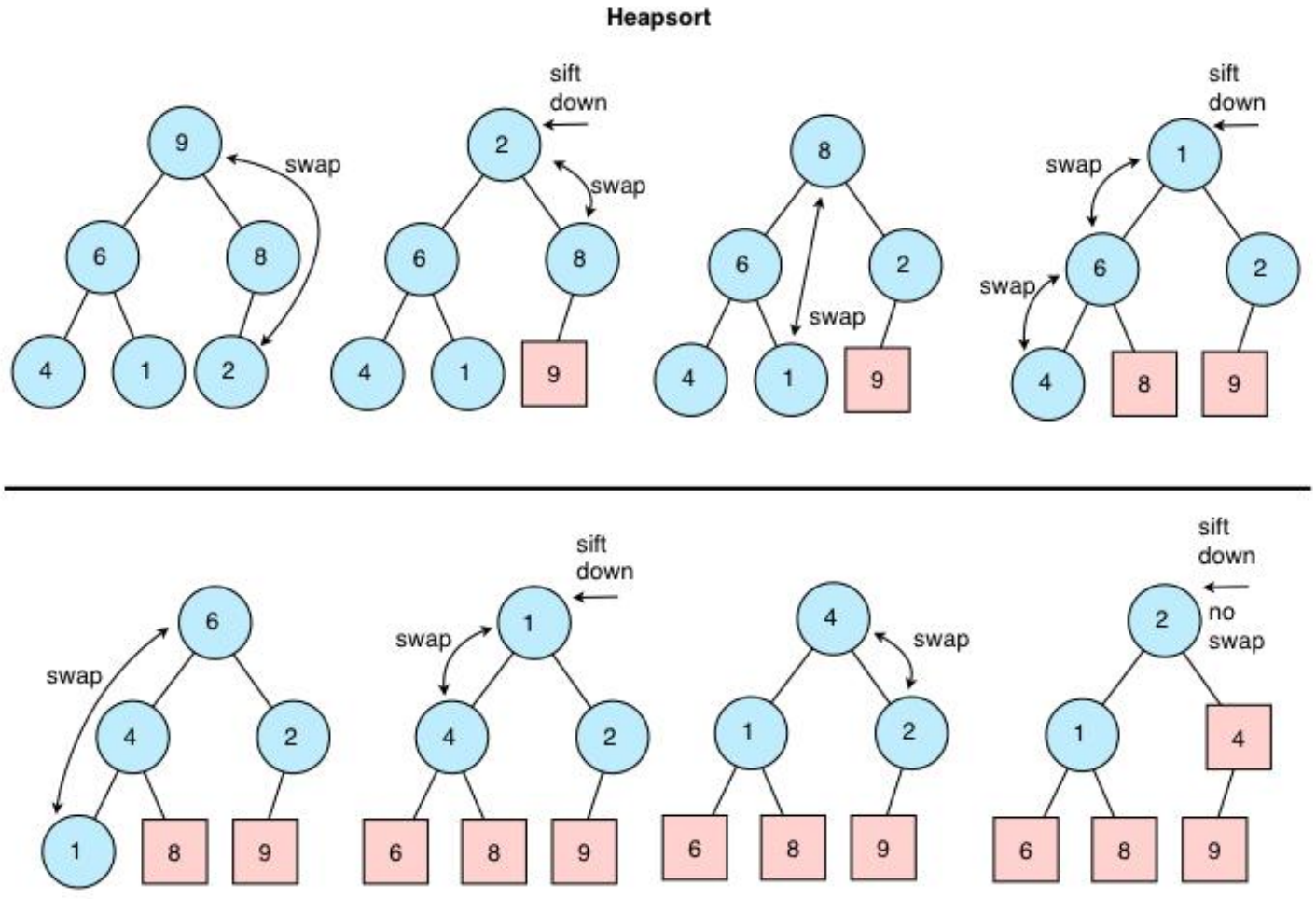
Remoção

- A remoção do maior valor da raiz (no caso do Heap Max) pode ser implementada removendo a raiz e substituindo-a pelo último elemento do Heap e "empurrando-o"para baixo na árvore até que o heap esteja corrigido.
- Qual o custo?

```
void corrigeDescendo(int index)
   {
   if (filhoE(index) < heap.size())     // Se index tem filho
      {
      // Seleciona o filho com menor valor (esquerda ou direita?)
      int filho = filhoE(index);
      if (filhoD(index) < heap.size() && heap[filhoD(index)] > heap[filhoE(index)])
         filho = filhoD(index);
      // Se o valor do pai é maior do que o valor do maior filho, terminamos
      if (heap[index] > heap[filho])
         return;
      // Caso contrário, trocamos o pai com o filho no heap e corrigimos agora para o fi
      troca(heap[index], heap[filho]);
      corrigeDescendo(filho);
      }
   }
```

- Heaps podem ser utilizados para implementar eficientemente filas de prioridades, já que o custo das principais operações (inserção, remoção) é O(log n) e o custo para obter o maior elemento é O(1).
- Filas de prioridade são utilizadas em diversos algoritmos clássicos tais como:
  - Algoritmo de dijstra para caminho mínimo
  - Algoritmos de Prim e Kruskal para árvore geradora mínima
  - Algoritmo de Branch-and-bound para buscar nodo de melhores possibilidades

- Heaps também são utilizados para implementar o algoritmo de ordenação Heapsort que basicamente é:
  - Coloque os dados a serem ordenados em um heap (custo O(n log n))
  - A cada iteração remova a raiz do heap (maior valor) e coloque no final do vetor, e reorganize o heap (O(log n))


Heapsort
sift down
swap
swap
swap
sift down
swap
swap
swap
sift down
swap
swap
sift down
no swap

- Implemente os algoritmos de ordenação Heapsort, Mergesort e Quicksort.
- Efetue medidas de tempo para ordenar vetores de valores aleatórios de valores inteiros entre 0 e 100.
- Teste para vetores a partir de 50.000 elementos, com incrementos no tamanho do vetor de 50.000 em 50.000.
- Teste os três algoritmos para os casos em que os valores do vetor já estão em ordem crescente e o caso em que estão em ordem decrescente.

- Faça um programa que leia o arquivo dados.txt (disponível no acervo da disciplina do UCS Virtual como "nomes e datas"), contendo 800.000 nomes e datas de nascimento no formato "nome,dd/mm/aa"e gere um arquivo "saida.txt"ordenado por ordem de data e, para cada data, por ordem de nome.
- Considere que há repetição de nomes e datas dentro do arquivo e os nomes podem conter espaços em branco.
- Meça o tempo de execução.
- Possibilidades a explorar:
  - Bubblesort
  - Quicksort
  - Mergesort
  - Heapsort
  - Árvore auto-balanceada

- Medição de tempo em trechos de código pode ser feita utilizando a função time( ) da biblioteca time.h, que retorna o número de segundos transcorridos desde 1/1/1970.
- Pode ser utilizada da seguinte forma:

```c
#include <time.h>
#include <stdlib.h>
#include <stdio.h>

int main(void)
{
time_t t0,t1;
t0=time(NULL);

// trecho de código a ser medido

t1 = time(NULL);
printf("tempo:%d\n",t1-t0);
}
```

Operadores bit a bit

- Os operadores bit a bit são comumente utilizados para trabalhar com dispositivos (pois os mesmos utilizam bytes ou palavras codificadas para comunicação), modo de armazenamento (um byte pode representar 8 informações binárias), e até compactação de dados (utilização de bits ociosos). A tabela a seguir mostra os operadores bit a bit suportados pela linguagem.

Operadores bit-a-bit

Operador	Ação
&	E (AND)
|	OU (OR)
^	OU exclusivo (XOR)
~	Complemento de um
<<	Deslocamento à esquerda
>>	Deslocamento à direita

- Os operadores bit a bit só podem ser utilizados sobre um byte ou uma palavra, isto é, aos tipos de dados char e int e variantes do padrão C. Operações bit não podem ser usadas em float, double, long double, void ou outros tipos mais complexos.

- E a tabela verdade de cada operador é:

A	B	A & B	A|B	A^B
0	0	0	0	0
0	1	0	1	1
1	0	0	1	1
1	1	1	1	0

A	~A
0	1
1	0

O Operador &

- O operador bit a bit & executa um e bit-a-bit para cada par de bits, produzindo um novo byte ou palavra.
- Para cada bit dos operandos, o operador & retorna o bit em 1 se ambos os bits dos operandos é 1.
- Caso algum bit dos operandos for 0, o operador retorna o bit 0.
- Este operador é mais utilizado para desligar bits (realizando a operação com um operando também chamado de máscara cujos bits que devam ser desligados estejam com valor 0, enquanto que os outros estão em 1) ou verificar se um determinado bit está ligado ou desligado (realizando a operação com uma máscara cujo bit que deva ser checado esteja com valor 1, enquanto que os outros estão em 0).

- Exemplo

```
unsigned char x = 7; /* 0000 0111 */
unsigned char y = 4; /* 0000 1010 */
unsigned char mascara = 252; /* 1111 1100 */
unsigned char res;
res = x & y; /* res = 0000 0010 */
res = y & mascara; /* res = 0000 1000 ? desligar os bits 0 e 1 */
res = y & 2 /* res = 0000 0010 ? bit ligado qdo res > 0 */
res = y & 4 /* res = 0000 0000 ? bit desligado qdo res == 0 */
```

- Para gerar uma máscara com 0 na posição p (considerando as posições como numeradas a partir de 0 da direita para a esquerda) e 1 nas outras posições pode-se fazer:
  - Mascara $= \tilde{}(1 << p)$

Operador "ou"bit-a-bit

- O operador bit a bit | executa um **ou lógico** para cada par de bits, produzindo um novo byte ou palavra.
- Para cada bit dos operandos, o operador | retorna o bit em 1 se algum dos bits dos operandos é 1.
- Caso ambos os bits dos operandos for 0, o operador retorna o bit 0.
- Este operador é mais utilizado para ligar (realizando a operação com um operando cujos bits que devam ser ligados estejam com valor 1, enquanto que os outros que não devem ser alterados estão em 0).

Exemplo

```
unsigned char x = 7; /* 0000 0111 */
unsigned char y = 4; /* 0000 1010 */
unsigned char mascara = 1; /* 0000 0001 */
unsigned char res;

res = x | y; /* res = 0000 1111 */
res = y | mascara; /* res = 0000 1011 ? ligar o bit 0 */
res = x | 8; /* res = 0000 1111 ? ligar o bit 3 */
```

- Para gerar uma máscara com 1 na posição p (considerando as posições como numeradas a partir de 0 da direita para a esquerda) e 0 nas outras posições pode-se fazer:
  - Mascara = 1 << p

- Não confunda os operadores relacionais && e || com & e |, respectivamente. Os operadores relacionais trabalham com os operandos como um único valor lógico (verdadeiro ou falso), e eles produzem somente dois valores 0 ou 1. Os operadores bit a bit podem produzir valores arbitrários pois a operação é realizada em nível de bit.

Operador de ou-exclusivo

- O operador bit a bit ^ executa um ou-exclusivo (XOR) lógico para cada par de bits, produzindo um novo byte ou palavra.
- Para cada bit dos operandos, o operador ^ retorna o bit em 1 se somente um dos bits dos operandos é 1.
- Caso os bits dos operandos forem iguais, o operador retorna o bit 0.
- Este operador é mais utilizado para inverter bits (realizando a operação com um operando cujos bits que devam ser invertidos estejam com valor 1, enquanto que os outros estão em 0).
- Trivia: o que resulta do comando abaixo?
  - a^=b^=a^=b;

Exemplo

```
unsigned char x = 7; /* 0000 0111 */
unsigned char y = 4; /* 0000 1010 */
unsigned char mascara = 3; /* 0000 0011 */
unsigned char res;
res = x ^ y; /* res = 0000 1101 */
res = y ^ mascara; /* res = 0000 1001 ? inverter os bits 0 e 1 */
res = y ^ 8; /* res = 0000 0010 ? inverter o bit 3 */
```

Operador de complemento~

- O operador bit a bit~executa um não lógico (complemento de 1) no valor a sua direita (operador unário), produzindo um novo byte ou palavra com os bits invertidos.
- Exemplo

```
unsigned char x = 7; /* 0000 0111 */
unsigned char res;
res = ~x; /* res = 1111 1000 */
res = ~127; /* res = 1000 0000 */
```

Operadores de deslocamento

- Os operadores de deslocamento, « e », movem todos os bits de um operando para a esquerda ou direita, respectivamente. A forma geral do comando de deslocamento é:
- Sintaxe:
  - operando << número de bits
  - operando >> número de bits
- Conforme os bits são deslocados para um direção, zeros ou uns são utilizados para preencher a extremidade contrária da direção (isto é, deslocamento para a direita coloca zeros ou uns nos bits mais significativos).

- No deslocamento à direita de valores com sinal (sem o modificador unsigned), o valor introduzido pela esquerda é um valor igual ao bit de sinal. Se o valor é negativo, com bit de sinal igual à 1, 1's são inseridos à esquerda.
- Valores inteiros são representados em **complemento de 2**.
  - Se o bit mais significativo é 1, o valor é negativo.
  - Pode-se obter o valor positivo correspondente invertendo todos os bits e somando 1 ao resultado.

- Estes operadores são utilizados para recebimento e envio de dados bit a bit (conversores analógico/digitais, portas seriais), e multiplicação (deslocamento para a esquerda) e divisão inteira (deslocamento para a direita) por 2.
- Os operadores de deslocamento podem ser utilizados com variáveis, constantes e até mesmo expressões. Entretanto, deve-se verificar a precedência de operadores quando trabalhando com expressões.
- Exemplo

```
unsigned char x = 7; /* 0000 0111 */
unsigned char res;
res = x << 1; /* res = 00001110 res = 14 */
res = x << 3; /* res = 01110000 res = 112 */
res = x << 2; /* res = 11000000 res = 192 */
res = x >> 1; /* res = 01100000 res = 96 */
res = x >> 2; /* res = 00011000 res = 24 */
```

Operadores bitwise de atribuição

- A Tabela a seguir mostra os operadores bit a bit de atribuição suportados pela linguagem.

Operador	Ação
x &= y	x = x & y
x |= y	x = x | y
x^= y	x = x^y
x >>= y	x = x >> y
x <<= y	x = x << y

Figura: Operadores bitwise reduzidos

- As expressões com este operadores são mais compactas e normalmente produzem um código de máquina mais eficiente.
- A execução da operação bit a bit ocorre por último após a avaliação da expressão à direita do sinal de igual.

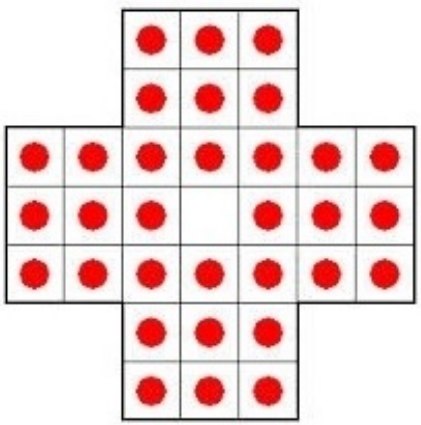
Exercícios

- 1) Uma forma de representar um conjunto é através de um vetor de bits. Por exemplo, um conjunto de valores entre 0 e 7 poderia ser representado como um unsigned char, onde cada bit i de 0 a 7 representaria a presença ou não do elemento i no conjunto. Assim, o conjunto 1,4,5 seria representado pelo char 00110010. Da mesma forma, um conjunto de valores entre 0 e 31 poderia ser representado por um unsigned int. Utilizando operadores binários, faça as funções a seguir:

1. void insere (unsigned int *conjunto, int valor) que insere um valor entre 0 e 31 em um conjunto.
2. int testa (unsigned int conjunto, int valor) que verifica se um dado valor está em um conjunto, retornando 0 ou 1.
3. void remove (unsigned int *conjunto, int valor) que remove um valor entre 0 e 31 de um conjunto.
4. void mostra (int conjunto) que mostra os valores que fazem parte do conjunto.
5. void mostrabin (int conjunto) mostra o valor do conjunto em binário.

- Idem, para as funções a seguir:
  1. unsigned int intersecao (unsigned int c1, unsigned int c2) : retorna um conjunto com os elementos comuns (intersecção) dos conjuntos c1 e c2;
  2. unsigned int uniao(unsigned int c1, unsigned int c2) : retorna um conjunto com os elementos que aparecem em pelo menos um dos conjuntos c1 e c2 (união);
  3. unsigned int diferença(unsigned int c1, unsigned int c2) : retorna um conjunto com os elementos que estão em c1 mas não estão em c2;
  4. void rota_esq (unsigned int *valor) : rotaciona os bits uma posição à esquerda, o bit mais significativo deve ir para a posição 0;
  5. void rota_dir (unsigned int *valor) : rotaciona os bits uma posição à direita, o bit 0 deve ir para o bit mais significativo;

- 2) O desafio "resta-um"é jogado em um tabuleiro no formato abaixo, que pode ser representado por uma matriz 7 x 7 na qual são usadas somente as posições representadas na figura. Cada posição da matriz tem 1 para as casas ocupadas e 0 para as casas não ocupadas. Faça uma função que receba uma matriz M[7][7] de inteiros iguais a 0 ou 1, e retorne um inteiro de 32 bits sem sinal em que cada bit contem o valor correspondente a uma posição da matriz. Represente apenas os primeiros 32 bits dos 33 que são usados para representar o tabuleiro.



3)(2.5 pontos) Faça uma função que receba uma matriz M[4][4] de inteiros iguais a 0 ou 1, e retorne um inteiro de 16 bits sem sinal em que cada bit contem o valor correspondente a uma posição da matriz. O bit 15 deve conter o valor da posição m[0][0] e o bit 0 deve conter o valor da posição [3][3]. Ex: Se a matriz M contem:

0	1	0	1
1	1	1	1
0	1	0	1
0	0	0	0

A função deve retornar o valor inteiro correspondente ao binário 0101 1111 0101 0000.

4) Diz-se que um valor está no formato *big endian*, quando as partes mais significativas do valor aparecem nas primeiras posições. Assim, números decimais estão no formato *big endian* porque os dígitos de maior peso aparecem antes dos dígitos de menor peso. Da mesma forma, em valores representados em little endian, a parte de menor peso aparece no início. Ex: datas em formato americano (AAAA/MM/DD).

- Processadores da linha Intel x86 armazenam valores inteiros no formato little endian, ou seja, os bytes menos significativos estão nas posições inferiores de memória.
- O que é escrito pelo programa a seguir?

```
#include <stdio.h>
#include <stdlib.h>
int main()
{
unsigned int a=0xABCDEF12;
unsigned char *p=(unsigned char *)&a;
for (int i=0;i<4;i++)
      printf("p[%d]%x\n",i,p[i]);
system("pause");
}
```

- Faça uma função que receba um inteiro de 4 bytes em formato *little endian* e retorne, como um segundo parâmetro, o valor convertido para o formato *big endian*.

Codificação de Huffman

- A codificação de Huffman é um método de compressão que usa as probabilidades de ocorrência dos símbolos no conjunto de dados a ser comprimido para determinar códigos de tamanho variável para cada símbolo.
- Ele foi desenvolvido em 1952 por David A. Huffman que era, na época, estudante de doutorado no MIT, e foi publicado no artigo "A Method for the Construction of Minimum-Redundancy Codes".

- Uma árvore binária completa, chamada de árvore de Huffman é construída a partir da junção dos dois símbolos de menor probabilidade, que são então somados em símbolos auxiliares e estes símbolos auxiliares recolocados no conjunto de símbolos.
- O processo termina quando todos os símbolos foram unidos em símbolos auxiliares, formando uma árvore binária.
- A árvore é então percorrida, atribuindo-se valores binários de 1 ou 0 para cada aresta, e os códigos são gerados a partir desse percurso.

Exemplo de Distribuição de Probabilidade

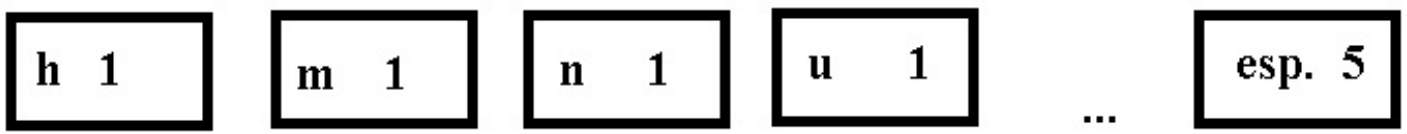
- Considere a frase "Eu odeio a codificação de huffman!!!"
- O número de ocorrências de cada símbolo na frase é:

símbolo	n. de ocorr.	símbolo	n. de ocorr.
a	3	c	2
d	3	e	2
f	3	h	1
i	3	m	1
n	1	o	4
u	1	ã	1
ç	1	!	3
espaço	5		

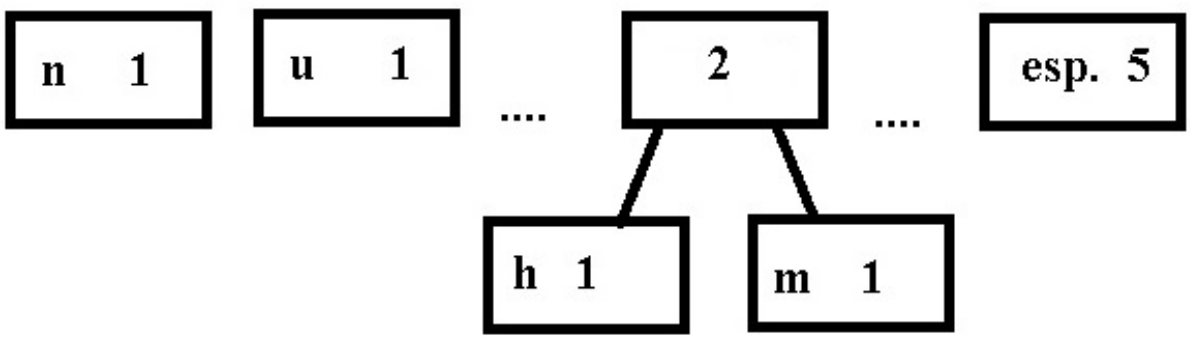
- A codificação de Huffman tem desempenho ótimo quando as probabilidades dos símbolos são potências negativas de dois (2^{-1}, 2^{-2}, ...).
- A codificação gerada tem também a garantia de ser não ambígua, pois nenhum código pode ser o prefixo de outro código.

Criação da Árvore de Huffman

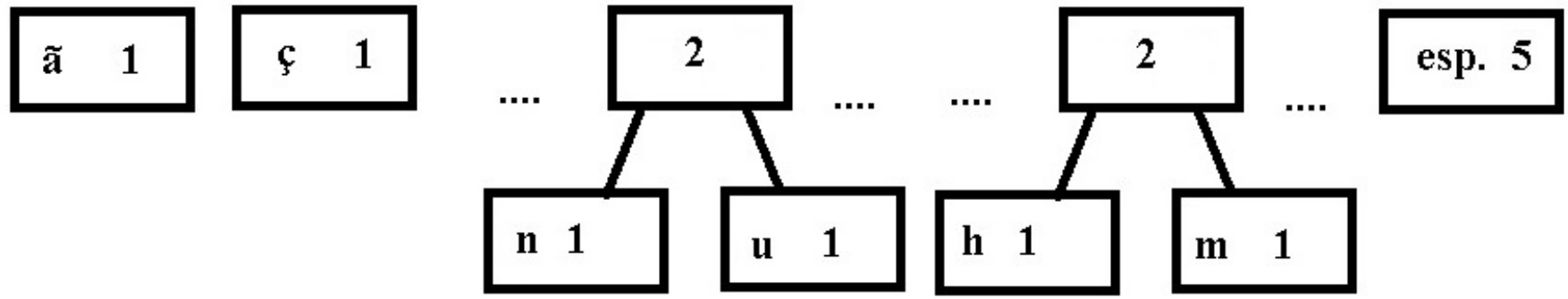
1. Determinação da freqüência de ocorrência de cada símbolo;
2. Criação de uma Fila de Prioridades por freqüência de ocorrência de cada símbolo, na qual cada nó será uma raiz de árvore binária contendo um símbolo e a freqüência de ocorrência correspondente;

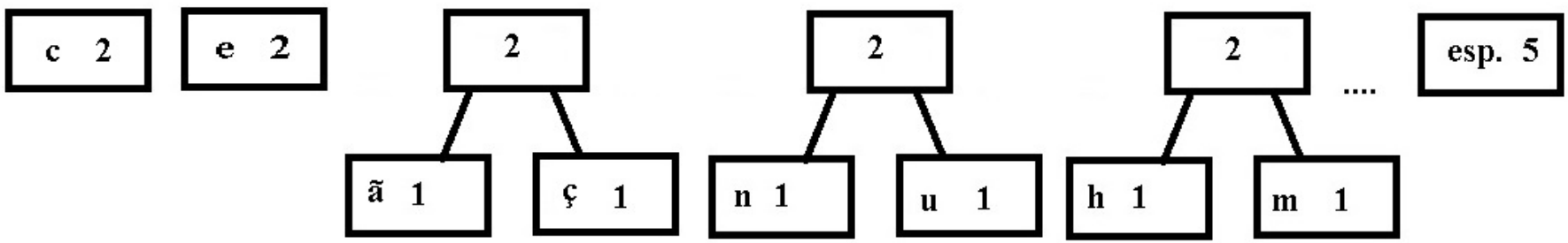


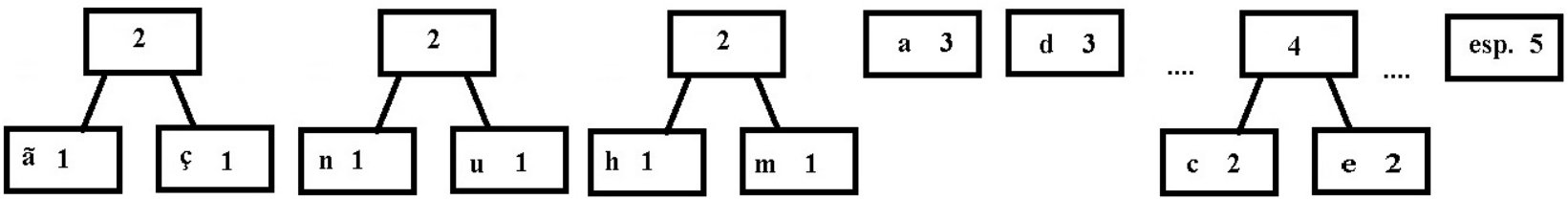
1. Retirada da Lista de Prioridades os dois primeiros nós e sua inclusão como filhos esquerdo e direito de um nó de agregação que não tenha símbolo mas cuja freqüência de ocorrência seja igual à soma das freqüências de ocorrência dos símbolos contidos em seus filhos;
2. Inclusão na Lista de Prioridades do nó de agregação assim criado;



1. Repetição dos passos 4 e 5 até que a Fila de Prioridades contenha um só nó, que é a raiz da árvore de Huffman;






2
ã 1
ç 1
2
n 1
u 1
2
h 1
m 1
a 3
d 3
....
4
c 2
e 2
....
esp. 5

1. Atribuição do código correspondente a cada símbolo identificado pela trajetória obtida da raiz até a folha da árvore de Huffman que contém o símbolo.

Codificação de Huffman

1. Identificação na tabela de códigos de Huffman do código correspondente ao símbolo a codificar;
2. Substituição do símbolo a codificar pelo código correspondente.

Decodificação de Huffman

1. Escolha do primeiro bit do código como bit corrente;
2. Desça um nível na árvore de Huffman de acordo com o valor do bit corrente (passar ao filho mais velho se o bit for 0 ou passar ao filho mais novo se o bit for 1) e atribuição ao bit corrente do próximo bit do ?string? de bits do código a decodificar;
3. No caso de ser atingida uma folha da árvore de Huffman deve-se fazer a substituição dos bits correspondentes à trajetória até a folha pelo caractere contido nessa folha;
4. Repetição dos passos 2 e 3 até que termine o "string"de bits do código a decodificar.

Campos de Bits

- C tem um método intrínseco para acessar um único bit dentro de um byte. Isso pode ser útil por diversas razões:
  - Se o armazenamento é limitado, você pode armazenar diversas variáveis Booleanas (verdadeiro/falso) em um byte.
  - Certos dispositivos transmitem informações codificadas nos bits dentro de um byte.
  - Certas rotinas de criptografia precisam acessar os bits dentro de um byte. Para acessar os bits, C usa um método baseado na estrutura. Um campo de bits é, na verdade, apenas um tipo de elemento de estrutura que define o comprimento, em bits, do campo. A forma geral de uma definição de campo de bit é:

- Para declarar um campo de bits, define-se o tipo do campo e a lista de sub-campos, com o número de bits em cada um:

```
struct nome {
tipo <lista de campo:comprimento>;
} lista de variaveis;
```

Ex:

```
struct pixel {
unsigned short int r : 5, g : 6, b : 5;
} p1,p2,p3;
```

- Um campo de bit deve ser declarado como int, unsigned ou signed. Campos de bit de comprimento 1 devem ser declarados como unsigned, porque um único bit não pode ter sinal. (Alguns compiladores só permitem campos do tipo unsigned).
- Um exemplo de campos de bits é a comunicação via serial que devolve um byte de estado organizado como mostra a tabela a seguir:
- Estado da comunicação serial.

Bit	Significado quando ligado
0	alteração na linha clear-to-send
1	alteração em data-set-ready
2	borda de subida da portadora detectada
3	alteração na linha de recepção
4	clear-to-send
5	data-set-ready
6	chamada do telefone
7	sinal recebido

- Pode-se representar a informação em um byte de estado utilizando o seguinte campo de bits:
- Exemplo

```
struct status_type {
unsigned delta_cts : 1;
unsigned delta_dsr : 1;
unsigned tr_edge : 1;
unsigned delta_rec : 1;
unsigned cts : 1;
unsigned dsr : 1;
unsigned ring : 1;
unsigned rec_line : 1;
} status;
```

- Para atribuir um valor a um campo de bit, simplesmente utiliza-se a forma para atribuição de outro tipo de elemento de estrutura.
- status.ring = 0;
- Não é necessário dar um nome a todo campo de bit. Isto torna fácil alcançar o bit que se deseja acessar, contornando os não usados. Por exemplo, se apenas cts e dtr importam, pode-se declarar a estrutura status_type desta forma:

```
struct status_type {
unsigned : 4;
unsigned cts : 1;
unsigned dsr : 1;
} status;
```

- Além disso, nota-se que os bits após dsr não precisam ser especificados se não são usados.
- Variáveis de campo de bit têm certas restrições:
  - Não pode obter o endereço de uma variável de campo de bit.
  - Variáveis de campo de bit não podem ser organizadas em matrizes.
  - Não pode ultrapassar os limites de um inteiro.
  - Não pode saber, de máquina para máquina, se os campos estarão dispostos da esquerda para a direita ou da direita para a esquerda.
  - Em outras palavras, qualquer código que use campos de bits pode ter algumas dependências da máquina.

- Finalmente, é válido misturar elementos normais de estrutura com elementos de campos de bit. O Exemplo abaixo define um registro de um empregado que usa apenas um byte para conter três informações: o estado do empregado, se o empregado é assalariado e o número de deduções. Sem o campo de bits, essa variável ocuparia três bytes.
- Exemplo

```
struct emp {
struct addr endereco;
float salario;
unsigned ativo : 1; /* ocioso ou ativo */
unsigned horas : 1; /* pagamento por horas */
unsigned deducao : 3; /* deduções de imposto */
};
```

Manipulação de Arquivos Binários

- Estrutura FILE: Para a manipulação de arquivos é utilizado a declaração de ponteiro (ponteiro de arquivo). Isto é, um ponteiro para informações que definem vários dados sobre o arquivo, como o seu nome, status, e a posição atual do arquivo. Um ponteiro de arquivo é uma variável ponteiro do tipo FILE .
- Todas as funções são realizadas utilizando o ponteiro. Para a declaração de um ponteiro de arquivo utiliza-se a seguinte sintaxe:

  FILE *<var>
- Exemplo

  FILE *fp;

ABERTURA DE ARQUIVOS

- A função fopen() abre arquivo e o associa ao descritor do arquivo. Ela retorna o ponteiro de arquivo associado a esse arquivo. Sua sintaxe é:

  FILE *fopen(const char * <nome_arquivo>, const char * <modo_abertura>);

- O modo de abertura define a forma como é feito o acesso aos dados (somente leitura, leitura e escrita, etc). As formas principais são apresentadas na tabela a seguir.

Modo	Significado
rb	Abre um arquivo binário para leitura
wb	Cria um arquivo binário para escrita
ab	Anexa a um arquivo binário
r+b ou rb+	Abre um arquivo binário para leitura/escrita
w+b ou wb+	Cria um arquivo binário para leitura/escrita
a+b ou ab+	Anexa a um arquivo binário para leitura/escrita

- Exemplo

  FILE *arq; /* ponteiro de arquivo */

  arq = fopen("dados.dat","wb");

- Se ao abrir um arquivo para leitura o mesmo não existir a função fopen retorna um ponteiro nulo (NULL).

  arq = fopen("dados.dat","rb");

  if (arq == NULL)

  printf("Erro na abertura do arquivo");

Escrita de um Bloco de Dados

- Para se gravar um bloco de dados é utilizada a função fwrite(). Seu protótipo está definido a seguir:
- Sintaxe:

  size_t fwrite(void *buffer, size_t num_bytes, size_t count, FILE *fp);
- onde buffer é um ponteiro para uma região de memória que contém as informações que serão escritas no arquivo. O número de bytes de cada item a gravar é especificado por num_bytes. O argumento count determina quantos itens serão gravados. E, finalmente, fp é um ponteiro de arquivo para um arqyuvi aberto anteriormente.
- Esta função devolve o número de itens escritos. O número retornado pode ser menor que count quando o final de arquivo for atingido ou ocorrer um erro de gravação.

Exemplo

```
int var_int;
FILE *arq;
arq = fopen("dados.dat","wb");
var_int = 5;
fwrite(&var_int,sizeof(var_int),10,arq);
```

Leitura de um bloco de dados

- Para se ler um bloco de dados é utilizada a função fread(). Seu protótipo está definido a seguir:
- Sintaxe:

  size_t fread(void *buffer, size_t num_bytes, size_t count, FILE *fp);
- onde buffer é um ponteiro para uma região de memória que receberá os dados do arquivo. O número de bytes de cada item a ler é especificado por num_bytes. O argumento count determina quantos itens serão lidos. E, finalmente, fp é um ponteiro de arquivo para um arquivo aberto anteriormente.
- Esta função devolve o número de itens lidos. O número retornado pode ser menor que count quando o final de arquivo for atingido ou ocorrer um erro de leitura.

Exemplo

```
int var_int;
FILE *arq;
arq = fopen("dados.dat","rb");
fread(&var_int,sizeof(var_int),1,arq);
```

Acesso Aleatório : FSEEK()

- Operações de leitura e escrita aleatórias podem ser executadas utilizando o sistema de E/S bufferizado com a ajuda de fseek(), que modifica o indicador de posição de arquivo. Seu protótipo é mostrado aqui:
- Sintaxe:

  int fseek (FILE *fp, long numbytes, int origem);
- Onde, numbytes, um inteiro longo, é o número de bytes a partir de oriem, que se tornará a nova posição corrente, e origem é uma das seguintes macros definidas em STDIO.H.

Origem	Nome da Macro
Início do arquivo	SEEK_SET
Posição atual	SEEK_CUR
Final do arquivo	SEEK_END

- A função fseek() devolve 0 se a operação for bem-sucedida e um valor diferente de zero se ocorre um erro.

```
if (fseek(fp, atol(argv[2]), SEEK_SET)!=0)
printf("Erro!!!");
```

- 1) Faça um programa que leia 10 valores inteiros do teclado e grave-os em um arquivo BINÁRIO dados.bin.
- 2) Faça um programa que leia o arquivo dados.bin, contendo valores inteiros, e escreva na tela os valores lidos.
- 3) Faça um programa que leia o arquivo dados.bin e troque, no mesmo arquivo, os valores pares por zero.
- 4) Utilize o programa do exercício 2 para verificar se a alteração no arquivo pedida no exercício 3 foi feita corretamente.

Função FTELL

- Esta função retorna a posição do ponteiro de um arquivo binário em relação ao seu começo.
- Esta função aceita um único argumento, que é o ponteiro para a estrutura FILE do arquivo.
- Seu protótipo é mostrado aqui:
- Sintaxe:

  long ftell (FILE *fp);
- Retorna um valor do tipo long, que representa o número de bytes do começo do arquivo até a posição atual.
- A função ftell() pode não retornar o número exato de bytes se for usada com arquivos em modo texto, devido à combinação CR/LF que é representada por um único caractere em C.

Fechamento de Arquivo

- A função fclose() fecha um arquivo que foi aberto através de uma chamada à fopen(). Esta função tem a seguinte sintaxe:
- Sintaxe:

  int fclose(FILE *fp);
- onde fp é o ponteiro de arquivo devolvido pela chamada à fopen(). O valor de retorno 0 significa uma operação de fechamento bem?sucedida.

Rewind( )

- Esta função reposiciona o indicador de posição de arquivo no início do arquivo especificado como seu argumento. Seu protótipo é:
- Sintaxe:
- void rewind(FILE *fp);

O Exemplo a seguir reposiciona o indicador de posição do arquivo do programa anterior e mostra o conteúdo do mesmo.

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
int main() {
char str[80];
FILE *fp;
if((fp=fopen("frase.dat","w"))==NULL) {
printf("Arquivo nao pode ser aberto\n");
exit(1);
}
do {
printf("entre uma string (CR para sair): \n");
gets(str);
strcat(str,"\n");
fputs(str,fp);
} while (*str != '\n');
rewind(fp); /* reinicializa o file pointer */
while(!feof(fp)) {
fgets(str, 79, fp);
printf(str);
}
fclose(fp);
return 0;
}
```

Condições de Erro

- Para determinar se um erro ocorreu utiliza-se a função ferror(), mas esta informação não basta para a solução por parte do usuário.
- Para isso utiliza-se a função perror() em conjunto com a função ferror(). O argumento de perror() é uma string fornecida pelo programa que normalmente é uma mensagem de erro que indica em que parte do programa ocorreu erro.
- Sintaxe:
- void perror (const char *str);
- Se for detectado um erro de disco, ferror() retornará um valor verdadeiro (não zero) e perror() imprimirá a seguinte mensagem
- Erro de Busca: Bad data
- A primeira parte da mensagem é fornecida pelo programa, e a segunda parte, pelo sistemaoperacional

- Exercícios:
  1. 1) Faça um programa que leia o arquivo arvore_binaria_16 (disponível no ucs virtual -> acervo da turma), com uma imagem BMP de 16 bits e a converta para tons de cinza. Use o mesmo cabeçalho do arquivo de origem, em que cada uma das 3 componentes de cada pixel é substituída pela média das cores. Desenvolva o código a partir do código gray_scale.cpp, disponível no acervo da turma.
  2. 2) Faça um programa que leia o arquivo anterior gerando uma imagem com apenas a componente vermelha (zerando as componentes verde e azul).
  3. 3) Idem, trocando a componente verde com a componente vermelha.
  4. 4) Repita os exercícios anteriores com o arquivo enterprise.bmp (BMP 24)
  5. 5) Faça um programa que leia o enterprise.bmp (BMP24 bits) e converta-o para um BMP16 bits.

- A imagem arvore_binaria_16 está em um padrão BMP565, em que as componentes R e B tem 5 bits e a componente G tem 6 bits. Para calcular a média deve-se considerar somente os 5 bits mais significativos da componente G.
- Supondo que se queira a média das componentes de um pixel px.

```
media=(px.r+px.b+px.g/2)/3;
px.r=media;
px.g=media*2; // volta a 6 bits
px.b=media;
```

- Utilizando o conjunto de caracteres ascii disponível em http://paulbourke.net/dataformats/asciiart/, altere o programa anterior para gerar um arquivo texto com a imagem do arquivo bmp lido.

- O programa huffman_arquivo.c, disponível no acervo da disciplina, compacta um arquivo utilizando a codificação de huffman. O programa grava no arquivo inicialmente a tabela de 256 inteiros contendo o número de ocorrências no arquivo original de cada caracter ascii de 0 a 255. A seguir é gravado um inteiro contendo o número de caracteres do arquivo original e a partir daí são gravados os dados codificados. Faça um programa para descompactar o arquivo gerado.

12/07/13 - Prova 1

1) (3.5 pontos) Escreva uma função que receba os apontadores de início de duas listas simplesmente encadeadas em que cada nodo contem um valor inteiro e verifica se as duas listas são exatamente iguais, ou seja, contem os mesmos valores na mesma ordem. A função deve retornar:

1 - Se as listas são exatamente iguais

0 - Se elas não são exatamente iguais

3) (3.5 pontos) Faça uma função que receba um apontador para uma lista de inteiros, e verifique se a lista está ordenada, retornando:

0 - Se a lista não está ordenada

1 - Se a lista está ordenada

Prova 1 - 06/05/14

2) (3.5 pontos) Escreva uma função que receba um apontador para uma lista simplesmente encadeada em que cada nodo contem um valor inteiro, e remova todos os nodos que contem um valor par.
3) (3 pontos) Escreva uma função que receba um apontador para o início de uma lista simplesmente encadeada com inteiros, e escreva todos os valores da lista na ordem contrária à que aparecem na lista, ou seja, primeiro o último elemento, depois o penúltimo e assim por diante.

Prova 1a - 2015-2

1) (2.5 pontos) Escreva uma função que receba os apontadores de início de duas listas simplesmente encadeadas em que cada nodo contem um valor inteiro e gere uma terceira lista com os elementos que aparecem somente em uma das listas, retornando, como parâmetro, o apontador da nova lista criada.
2) (2.5 pontos) Escreva uma função que receba o apontador de início de uma lista simplesmente encadeada em que cada nodo contem um valor inteiro e retorne, como parâmetro, o valor que ocorre mais vezes na lista.

3) (2.5 pontos) Escreva uma função que receba um apontador para o início de uma lista simplesmente encadeada com inteiros, e remova da lista todas as repetições, de modo que a lista resultante tenha apenas uma ocorrência de cada valor da lista original. Os valores na lista resultante devem estar na mesma ordem dos valores na lista original.
4) (2.5 pontos) Escreva uma função que receba um apontador para o início de uma lista simplesmente encadeada com inteiros, e ordene todos os valores da lista em ordem crescente.

Prova 1b - 2015-2

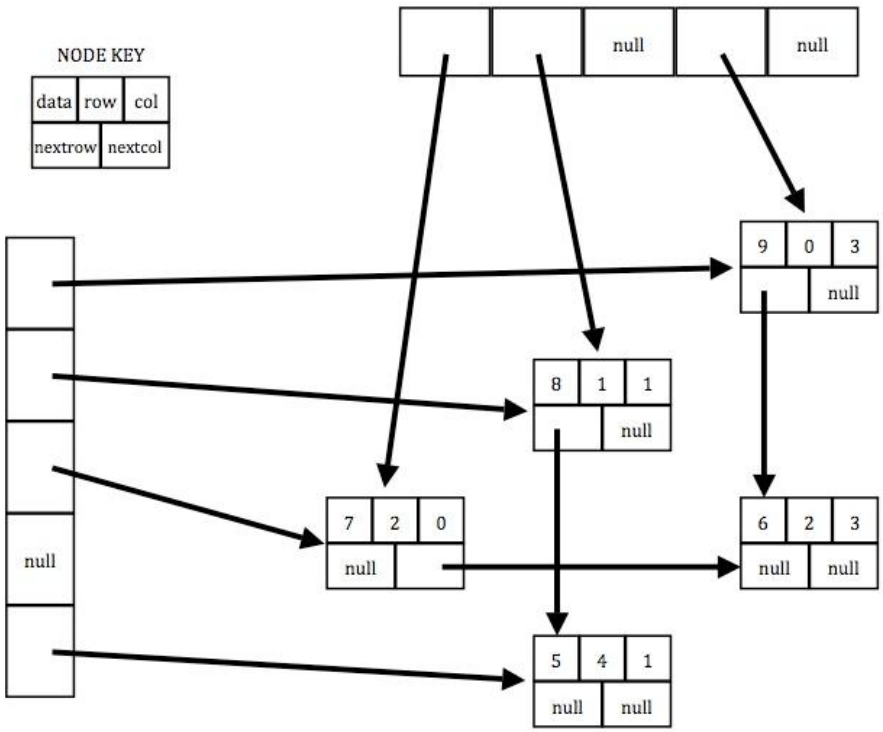
1)(2.5 pontos) Escreva uma função que receba o apontador de início de uma lista duplamente encadeada em que cada nodo contem um valor inteiro, o apontador para o próximo nodo e um apontador (inicialmente todos os nodos possuem NULL nesse campo) para o nodo anterior e, acerte em todos os nodos o apontador para o nodo anterior.

3)(2.5 pontos) Escreva uma função que receba um apontador para o início de uma lista simplesmente encadeada com inteiros, e escreva todos os valores da lista na ordem contrária à que aparecem na lista, ou seja, primeiro o último elemento, depois o penúltimo e assim por diante.
4)(2.5 pontos) Escreva uma função que receba um apontador para o início de uma lista simplesmente encadeada com inteiros e receba um inteiro N, e remova da lista todas as ocorrências de N.

Recuperação da Prova 1 - 2015-2

1)(2.5 pontos) Uma matriz é dita **esparsa** quando possui uma grande quantidade de elementos que valem zero. Matrizes esparsas têm aplicações em problemas de engenharia (por exemplo, o método das malhas para resolução de circuitos elétricos ou sistemas de equações lineares). Também têm aplicação em computação: armazenamento de dados (e.g., planilhas eletrônicas). A matriz esparsa pode ser implementada através de um conjunto de listas ligadas que apontam para elementos diferentes de zero como mostrado na figua abaixo. Desta forma os elementos que possuem valor zero não são armazenados.

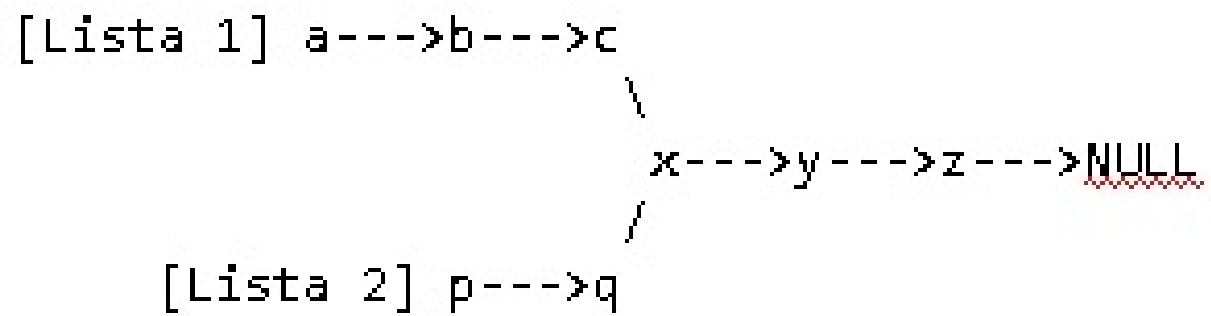
Nesta estrutura cada elemento possui o valor do elemento da matriz, o número da linha e da coluna do elemento, um apontador para o próximo elemento da mesma coluna, e um apontador para o próximo elemento da mesma linha.



Faça uma função que receba a posição de um elemento (linha e coluna) e remova-o de uma matriz esparsa 10x10 de floats, utilizando a estrutura representada na figura acima.

Prova1 - 2016_2 - Turma 38-39

1)(2.5 pontos) Faça uma função que receba apontadores para duas listas simplesmente encadeadas, em que uma parte é comum às duas listas (veja figura abaixo), e retorne o pointer para o nodo em que as duas se encontram. No desenho abaixo, o nodo a ser retornado é o nodo X. Considere que as duas listas não são vazias, garantidamente possuem uma parte em comum e podem ocorrer repetições de valores em cada uma das listas.



```
Node *FindMergeNode(Node *headA, Node *headB)
   {
   while (headA!=NULL)
      {
      Node *ptaux=headB;
      while (ptaux!=NULL)
         {
         if (ptaux==headA) return headA;
         ptaux=ptaux->next;
         }
      headA=headA->next;
      }
   return NULL; // caso não tenham sublista em comum
   }
```

3)(2.5 pontos) Escreva uma função que receba um apontador para o início de uma lista simplesmente encadeada com inteiros, e ordene a lista em ordem crescente.

- Solução: Usei um bubblesort com flag para verificar se já está ordenado.

```
void ordena(struct nodo *inicio)
   {
   if (inicio==NULL) return;
   int troca=1;
   while (troca)
      {
      troca=0;
      struct nodo *ptaux=inicio;
      while (ptaux->prox!=NULL)
         {
         if (ptaux->valor>ptaux->prox->valor)
            {
            int aux=ptaux->valor;
            ptaux->valor=ptaux->prox->valor;
            ptaux->prox->valor=aux;
            troca=1;
            }
         ptaux=ptaux->prox;
         }
      }
   }
```

4)(2.5 pontos) Escreva uma função que receba um apontador para o início de uma lista **duplamente encadeada** com inteiros e receba um inteiro N, e remova da lista todas as ocorrências de N.

- Solução: Remoção de elemento em lista duplamente encadeada está nas lâminas

Prova1 - 2016_2 - Turma 58-59

1)(2.5 pontos) Suponha dada uma lista encadeada primária cada um de cujos nós aponta para uma outra lista encadeada cujos nós contêm caracteres. Exemplo: o primeiro nó da lista primária aponta para uma lista que contém 'A' e 'B', nessa ordem (a lista termina com NULL); o segundo nó da lista primária aponta para uma lista que contém 'C'; o terceiro e último nó aponta para uma lista que contém 'D' e 'E', nessa ordem. Agora escreva uma função que receba uma lista do tipo primário e amarre todas as listas secundárias, uma após a outra, em um única lista. No exemplo acima, o resultado seria uma lista que contém 'A', 'B', 'C', 'D' e 'E', nessa ordem. A função não deve criar novos nodos, e sim usar os nodos já criados, alterando seus apontadores.

Solução:

```
void questao1(struct nodopri *ini, struct nodos **res)
   {
   *res=NULL;
   struct nodos *fim=NULL;
   while (ini)
      {
      if (*res==NULL)
         {
         *res=ini->lis;
         fim=ini->lis;
         }
      else fim->prox=ini->lis;
      while (fim->prox!=NULL) fim=fim->prox;
      ini=ini->prox;
      }
   }
```

2)(2.5 pontos) Uma forma de encontrar o nodo do meio de uma lista simplesmente encadeada é percorrê-la a partir do início com dois pointers p1 e p2 ao mesmo tempo, p1 avançando um nodo a cada iteração, e p2 avançando dois nodos a cada iteração. Quando p2 chegar ao final da lista, p1 estará no meio da lista. Utilizando o algoritmo descrito, implemente uma função que receba o apontador do início da lista e retorne um pointer para o nodo mais próximo do meio da lista. Não serão aceitas soluções que utilizem outro algoritmo (como, por exemplo, encontrar o tamanho da lista e percorrê-la uma segunda vez até a metade do tamanho)

Solução:

```
struct nodos *questao2(struct nodos *inic)
   {
   if (inic==NULL) return NULL;
   struct nodos *ptaux=inic;
   while (ptaux!=NULL && ptaux->prox!=NULL)
      {
      inic=inic->prox;
      ptaux=ptaux->prox->prox;
      }
   return inic;
   }
```

3)(2.5 pontos) Escreva uma função que receba os apontadores de duas listas simplesmente encadeadas L1 e L2, e gere uma terceira lista com todos os valores que estão em L1 mas não estão em L2. Retorne como parâmetro o apontador da lista criada.

```
void questao3(struct nodos *L1, struct nodos *L2, struct nodos **LS)
   {
   *LS=NULL;
   while (L1)
      {
      struct nodos *paux=L2;
      int tem=0;
      while (paux!=NULL){
         if (L1->valor==paux->valor) tem=1;
         paux=paux->prox;
         }
      if (tem==0)
         {
         struct nodos *ptaux=(struct nodos *)malloc(sizeof(struct nodos));
         ptaux->prox=*LS;
         *LS=ptaux;
         }
      L1=L1->prox;
      }
   }
```

4)(2.5 pontos) Escreva uma função que receba um apontador para o início de uma lista **duplamente encadeada** com inteiros e receba um inteiro N, e remova da lista todos os nodos entre a primeira e a segunda ocorrência de N. Considere que a lista não é vazia e contem ao menos duas ocorrências de N.

```
void questao4(struct nodod **L1, char valor)
   {
   struct nodod *ptaux=*L1;
   while (ptaux->valor!=valor) ptaux=ptaux->prox;
   struct nodod *ant=ptaux->ant;
   struct nodod *ptaux2=ptaux;
   ptaux=ptaux->prox;
   while (ptaux->valor!=valor)
      {
      free(ptaux2);
      ptaux2=ptaux;
      ptaux=ptaux->prox;
      }
   if (ant==NULL)
      *L1=ptaux->prox;
   else ant->prox=ptaux->prox;
   free(ptaux);
   }
```

Prova 2 - 05/07/13

- Considere a declaração do nodo de árvore a seguir:

```
struct nodo{    int valor;
                struct nodo *fesq,*fdir;}
```

- 1) (3 pontos) Faça uma função que receba o apontador raiz de uma árvore e crie uma cópia da árvore, retornando como um segundo parâmetro a raiz da árvore criada.
- 2) (3.5 pontos) Faça uma função que receba o apontador raiz de uma árvore em que cada e remova da árvore todos os nodos de valor par.
- 3) (3.5 pontos) Uma árvore é dita balanceada se em nenhum nodo a diferença de altura entre as sub-árvores do filho à esquerda e do filho à direita é maior do que 1. Escreva uma função que receba o apontador da raiz de uma árvore e verifica se ela é balanceada, retornando:
  - 1 - Se a árvore é balanceada
  - 0 - Se ela não é balanceada

Prova 2 - 08/07/14

2) (2.5 pontos) Considere a declaração do nodo de árvore a seguir:

```
struct nodo{int valor;
struct nodo *fesq,*fdir;}
```

Faça uma função que receba o apontador raiz de uma árvore e crie uma cópia da árvore, retornando como um segundo parâmetro a raiz da árvore criada.

3)(2.5 pontos) Uma forma de representar um conjunto é através de um vetor de bits. Por exemplo, um conjunto de valores entre 0 e 7 poderia ser representado como um unsigned char, onde cada bit i de 0 a 7 representaria a presença ou não do elemento i no conjunto. Assim, o conjunto 1,4,5 seria representado pelo char 00110010. Da mesma forma, um conjunto de valores entre 0 e 31 poderia ser representado por um unsigned int. Utilizando operadores binários, faça as três funções a seguir:

1. void insere (unsigned int conjunto, int valor) que insere um valor entre 0 e 31 em um conjunto.
2. int testa (unsigned int conjunto, int valor) que verifica se um dado valor está em um conjunto, retornando 0 ou 1
3. void remove (unsigned int conjunto, int valor) que remove um valor entre 0 e 31 de um conjunto

Recuperação Prova 2 - 20/07/14

2) (2.5 pontos) Considere a declaração do nodo de árvore a seguir:

```
struct nodo{int valor;
struct nodo *fesq,*fdir;}
```

Faça uma função que receba o apontador raiz de uma árvore de busca e crie uma árvore de busca que contenha somente os elementos de valor par da árvore original, retornando como um segundo parâmetro a raiz da árvore criada.

3)(2.5 pontos) Uma operação comum em algoritmos genéticos é a troca de bits entre duas variáveis, operação chamada de crossover. Faça uma função que receba dois unsigned int e o número de um bit (entre 0 e 31) e troque o bit correspondente entre as duas variáveis, retornando as variáveis alteradas.

4)(2.5 pontos) Um filtro de média móvel é obtido calculando-se a média de um conjunto de valores, sempre se adicionando um novo valor ao conjunto e se descartando o mais velho. Assim a aplicação de um filtro de média móvel considerando 3 elementos para o cálculo da média, aplicado ao conjunto de valores a seguir:
3 5 7 4 7 9 2 6 4 8 5 2 7 6 2
resultaria no conjunto de valores
5 5 6 6 6 5 4 6 5 5 4 5 5
onde o primeiro 5 = ((3+5+7)/3), o segundo 5 é ((5+7+4)/3) e assim por diante. Faça um programa que leia um arquivo binário de inteiros teste.dat e gere um arquivo saida.dat resultante da aplicação de um filtro de média móvel de janela igual a 10 sobre o arquivo lido.

Prova 2 - 2015-2

1) (2.5 pontos) Considere a declaração do nodo de árvore a seguir:

```
struct nodo{int valor;
struct nodo *fesq,*fdir;}
```

Duas árvores binárias são ditas **similares** se suas estruturas são idênticas, mesmo que os valores nos nodos não sejam necessariamente idênticos. Faça uma função que receba os apontadores das raízes de duas árvores e verifique se as duas árvores são similares, retornando:

- 1 - Se as duas árvores são similares
- 0 - Se as duas árvores não são similares

```
int similar (struct nodo *r1, struct nodo *r2)
    {
    if ((r1==NULL)&&(r2==NULL)) return 1;
    if ((r1==NULL)||(r2==NULL)) return 0;
    if (similar(r1->fesq,r2->fesq) &&
          similar(r1->fdir,r2->fdir)) return 1;
    return 0;
    }
```

2) (2.5 pontos) Usando a mesma declaração de tipo do exercício anterior, faça uma função que receba os apontadores das raízes de duas árvores r1 e r2 e verifique se r1 é sub-árvore de r2 retornando:

- 1 - Se a árvore encabeçada por r1 é sub-árvore da árvore encabeçada por r2 ou
- 0 - em caso contrário.

Se for necessário, desenvolva as funções auxiliares que forem necessárias.

```
int iguais (struct nodo *r1, struct nodo *r2)
   {
   if ((r1==NULL)&&(r2==NULL)) return 1;
   if ((r1==NULL)||(r2==NULL)) return 0;
   if (r1->valor==r2->valor)
      if (iguais(r1->fdir,r2->fdir))
         if (iguais(r1->fesq,r2->fesq))
            return 1;
   return 0;
   }
```

```
int sub (struct nodo *r1, struct nodo *r2)
   {
   if (r1==NULL) return 1; // uma árvore vazia é subárvore de qualquer árvore
   if (r2==NULL) return 0;
   if (iguais(r1,r2)) return 1;
   if (sub(r1,r2->fesq)) return 1;
   if (sub(r1,r2->fdir)) return 1;
   return 0;
   }
```

3)(2.5 pontos) Faça uma função que receba uma matriz M[4][4] de inteiros iguais a 0 ou 1, e retorne um inteiro de 16 bits sem sinal em que cada bit contem o valor correspondente a uma posição da matriz. O bit 15 deve conter o valor da posição m[0][0] e o bit 0 deve conter o valor da posição [3][3]. Ex: Se a matriz M contem:

0	1	0	1
1	1	1	1
0	1	0	1
0	0	0	0

A função deve retornar o valor inteiro correspondente ao binário 0101 1111 0101 0000.

```
unsigned short int bits (int m[4][4])
   {
   unsigned short int valret=0,i,j;
   for (i=0;i<4;i++)
      for (j=0;j<4;j++)
         valret = (valret << 1) | m[i][j];
   return valret;
}
```

4)(2.5 pontos) Faça uma função que receba os nomes de dois arquivos texto arq1 e arq2 e dois strings st1 e st2, substituindo todas as ocorrências de st1 no primeiro arquivo por st2, e colocando o resultado em arq2.
void substitui (char arq1[ ], char arq2[ ], char st1[ ], char st2[ ])

```
int proxocorr(int i, char linha[], char st1[])
// procura a próxima ocorrência de st1 na linha
   {
   for (;i<strlen(linha);i++)
      {
      int cont=0;
      for (int j=0;j<strlen(st1);j++)
         if (linha[i+j]==st1[j]) cont++;
      if (cont==strlen(st1)) return i;
      }
   return -1;
   }
```

```
void strsubstitui(char linha[], char linhasai[], char st1[], char st2[])
   {
   int isai=0;
   int i=0, pos;
   strcpy(linhasai,"");
   while ((pos=proxocorr(i,linha,st1))!=-1)
      {
      for (;i<pos;) linhasai[isai++]=linha[i++];
      strcat(&(linhasai[isai]),st2);
      isai+=strlen(st2);
      i=pos+ strlen(st1);
      }
   while (linha[i]!='\0') linhasai[isai++]=linha[i++];
   }
```

```
void substitui(char arq1[], char arq2[], char st1[], char st2[])
   {
   char linha[500],linhasai[500];
   FILE *farq1=fopen(arq1,"rt");
   FILE *farq2=fopen(arq2,"wt");
   while (fgets(linha,500,farq1))
      {
      strsubstitui(linha,linhasai,st1,st2);
      fprintf(farq2,"%s",linhasai);
      }
   fclose(farq1);
   fclose(farq2);
   }
```

```
void escbits(unsigned short int v)
   {
   int i;
   for (i=15;i>=0;i--) printf("%d",v>>i&1);
   printf("\n");
   }
```

Recuperação da Prova 2 - 2015-2

1) (2.5 pontos) Árvore AVL (ou árvore balanceada pela altura), em Ciência da Computação, é uma árvore de busca binária autobalanceada. Em tal árvore, as alturas das duas sub-árvores a partir de cada nó diferem no máximo em uma unidade. Faça uma função que receba o apontador para a raiz de uma árvore binária e verifique se ela é uma árvore AVL retornando:

1. Se é uma AVL
2. Se não é

- Resposta:

```
int altura (struct nodo *raiz)
      {
      if (raiz==NULL) return -1;
      int esq=altura (raiz->fesq);
      int dir=altura (raiz->fdir);
      if (esq>dir) return esq+1;
      else return dir+1;
      }
int AVL(struct nodo *raiz)
      {
      if (raiz==NULL) return 1;
      if (fabs(altura(raiz->fesq)-altura(raiz->fdir))>1) return 0;
      if (AVL(raiz->fesq) && AVL(raiz->fdir)) return 1;
      return 0;
      }
```

2) (2.5 pontos) Faça uma função que receba um inteiro sem sinal de 16 bits e inverta a posição de seus bits (troque o bit 0 com o 15, o 1 com o 14 e assim por diante) retornando um inteiro sem sinal de 16 bits com o valor resultante das trocas.

```
unsigned short int inverte(unsigned short int v)
{
      unsigned short int x=0,i;
      for (i=0;i<=15;i++)
            x=(x<<1)|((v>>i)&1);
      return x;
}
```

3) (2.5 pontos) Faça uma função que receba o apontador para a raiz de uma árvore binária e escreva os valores da árvore por níveis (primeiro a raiz, depois todos os nodos do nível 1, depois todos os nodos do nível 2 e assim por diante).
4) (2.5 pontos) Faça um programa que leia o arquivo "sonzao.dat" em que cada byte representa uma amostra de áudio de 8 bits e gere um arquivo de saída em que cada byte é o resultado da média do byte correspondente no arquivo original com os dois bytes seguintes a ele. Ex: se o arquivo original contem:

7 9 6 8 5 6 - o arquivo de saída conterá

$\frac{7+9+6}{3}$ $\frac{9+6+8}{3}$ $\frac{6+8+5}{3}$ $\frac{8+5+6}{3}$ - ou seja

7.33 7.66 6.66 6.66... (só a parte inteira, pois cada amostra é um inteiro de 8 bits)

Recuperação da Prova 2b - 2015-2

1) (2.5 pontos) Considere a declaração do nodo de árvore a seguir:

```
struct nodo{int valor;
struct nodo *fesq,*fdir;}
```

Uma árvore é dita **estritamente binária** caso todos os seus nós possuam 2 filhos ou sejam folhas. Faça uma função que receba o apontador da raiz de uma árvore binária e retorne:

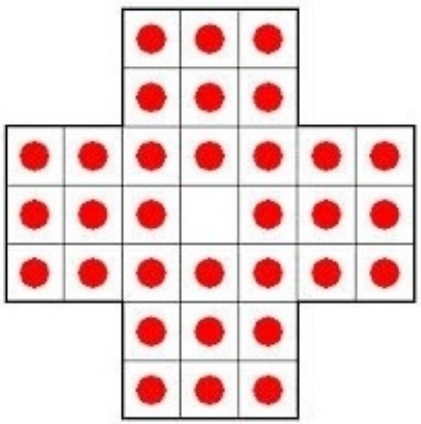
- 1 - Se a árvore é estritamente binária
- 0 - Se a árvore não é estritamente binária

```
int estrit(struct nodo *raiz)
   {
   if (raiz==NULL) return 1;
   if ((raiz->fesq==NULL && raiz->fdir!=NULL)||
       (raiz->fesq!=NULL && raiz->fdir==NULL)) return 0;
   return estrit(raiz->fesq) && estrit(raiz->fesq);
   }
```

2) (2.5 pontos) Usando a mesma declaração de tipo do exercício anterior, faça uma função que receba o apontador para a raiz de uma árvore binária e crie uma cópia da árvore, retornando a raiz da árvore criada. Se for necessário, desenvolva as funções auxiliares que forem necessárias.

```
struct nodo *copia (struct nodo *raiz)
   {
   if (raiz==NULL) return NULL;
   struct nodo *raux=(struct nodo *)malloc(sizeof(struct nodo));
   raux->fesq=copia (raiz->fesq);
   raux->fdir=copia (raiz->fdir);
   return raux;
   }
```

3)(2.5 pontos) O desafio "resta-um"é jogado em um tabuleiro no formato abaixo, que pode ser representado por uma matriz 7 x 7 na qual são usadas somente as posições representadas na figura. Cada posição da matriz tem 1 para as casas ocupadas e 0 para as casas não ocupadas. Faça uma função que receba uma matriz M[7][7] de inteiros iguais a 0 ou 1, e retorne um inteiro de 32 bits sem sinal em que cada bit contem o valor correspondente a uma posição da matriz. Represente apenas os primeiros 32 bits dos 33 que são usados para representar o tabuleiro.



4)(2.5 pontos) Faça um programa que leia um arquivo binário "Imagem24b.bmp" com uma imagem com pixels em 24 bits, um byte para cada componente de cor RGB no formato RRRRRRRRGGGGGGGGBBBBBBBB, e gere um arquivo binário "Imagem16b.bmp" com pixels em 16 bits com a estrutura RRRRRGGGGGGBBBBB.

Prova 1 - 05/05/16

- Use as seguintes declarações de tipos nas quatro questões:
  - struct nodos {char valor;struct nodos *prox;};
  - struct nodod {char valor;struct nodod *prox,*ant;};
  - struct nodopri {struct nodos *lis; struct nodopri *prox;};

- 1)(2.5 pontos) Suponha dada uma lista simplesmente encadeada primária cada um de cujos nós aponta para uma outra lista simplesmente encadeada cujos nós contêm caracteres. Exemplo: o primeiro nó da lista primária aponta para uma lista que contém 'A' e 'B', nessa ordem (a lista termina com NULL); o segundo nó da lista primária aponta para uma lista que contém 'C'; o terceiro e último nó aponta para uma lista que contém 'D' e 'E', nessa ordem. Agora escreva uma função que receba uma lista do tipo primário e amarre todas as listas secundárias, uma após a outra, em um única lista. No exemplo acima, o resultado seria uma lista que contém 'A', 'B', 'C', 'D' e 'E', nessa ordem. A função não deve criar novos nodos, e sim usar os nodos já existentes na lista primária, alterando seus apontadores. Use o cabeçalho a seguir para a função:
  - void questao1(struct nodopri *ini, struct nodos **res)

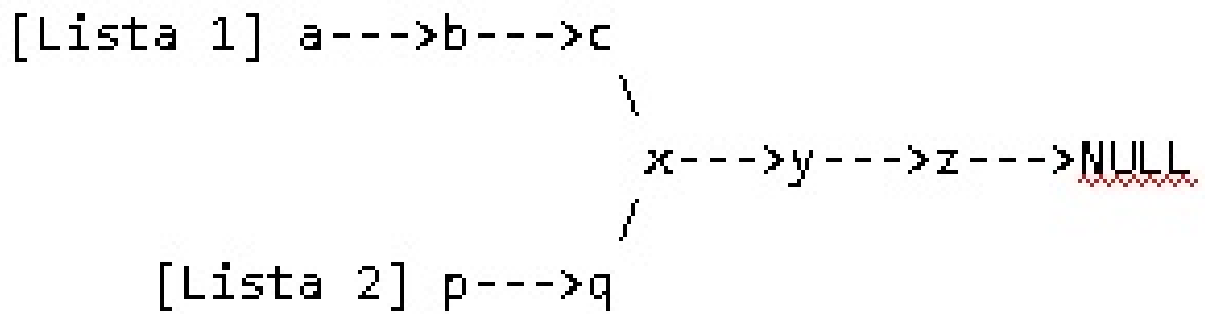
- 2)(2.5 pontos) Uma forma de encontrar o nodo do meio de uma lista simplesmente encadeada é percorrê-la a partir do início com dois pointers p1 e p2 ao mesmo tempo, p1 avançando um nodo a cada iteração, e p2 avançando dois nodos a cada iteração. Quando p2 chegar ao final da lista, p1 estará no meio da lista. Utilizando o algoritmo descrito, implemente uma função que receba o apontador do início da lista e retorne um pointer para o nodo mais próximo do meio da lista. Não serão aceitas soluções que utilizem outro algoritmo (como, por exemplo, encontrar o tamanho da lista e percorrê-la uma segunda vez até a metade do tamanho). Use o cabeçalho a seguir para a função:
  - struct nodos *questao2(struct nodos *inic)

- 3)(2.5 pontos) Escreva uma função que receba os apontadores de duas listas simplesmente encadeadas L1 e L2, e gere uma terceira lista com todos os valores que estão em L1 mas não estão em L2. Suponha que nenhuma das listas é vazia, e que não há repetição de valores em nenhuma das listas. Retorne como parâmetro o apontador da lista criada. Os valores podem aparecer em qualquer ordem na lista resultante. As listas L1 e L2 devem permanecer inalteradas. Use o cabeçalho a seguir para a função:
  - void questao3(struct nodos *L1, struct nodos *L2, struct nodos **LS)

- 4)(2.5 pontos) Escreva uma função que receba um apontador para o início de uma lista **duplamente encadeada** com inteiros e receba um inteiro N, e remova da lista todos os nodos entre a primeira e a segunda ocorrência de N. Considere que a lista não é vazia e contem ao menos duas ocorrências de N.Use o cabeçalho a seguir para a função:
  - void questao4(struct nodod **L1, char valor)

Prova 1 - 03/05/16

1)(2.5 pontos) Faça uma função que receba apontadores para duas listas simplesmente encadeadas, em que uma parte é comum às duas listas (veja figura abaixo), e retorne o pointer para o nodo em que as duas se encontram. No desenho abaixo, o nodo a ser retornado é o nodo X. Considere que as duas listas não são vazias e garantidamente possuem uma parte em comum.



2)(2.5 pontos) Escreva uma função que receba os apontadores de início de duas listas **duplamente encadeadas** em que cada nodo contem um valor inteiro e crie uma lista duplamente encadeada com os elementos que aparecem nas duas listas (interseção das listas), retornando como um terceiro parâmetro da função o apontador do início da lista criada. Considere que as listas não estão ordenadas e que não há repetição de valores em cada lista. E considere que as listas originais não podem ser destruídas.

3)(2.5 pontos) Escreva uma função que receba um apontador para o início de uma lista simplesmente encadeada com inteiros, e ordene a lista em ordem crescente.

4)(2.5 pontos) Escreva uma função que receba um apontador para o início de uma lista **duplamente encadeada** com inteiros e receba um inteiro N, e remova da lista todas as ocorrências de N.

Prova 2 - 05/07/16

- Considere a declaração do nodo de árvore a seguir:

  struct nodo {unsigned char c;

  struct nodo *fesq,*fdir;}
- 1) (3.5 pontos) Faça uma função buscachar com o protótipo a seguir:

  unsigned char buscachar (struct nodo raiz, unsigned int caminho);
- onde Raiz aponta para a raiz de uma árvore que contem em cada folha um caracter ascii. Os nodos internos contem o caracter '\0' no campo c. "caminho"contem um int de 32 bits com o caminho da raiz até o caracter buscado. A cada passo a função deve separar o bit mais significativo de "caminho"e descer pela esquerda na árvore (se o bit for 0) ou pela direita (se o bit for 1) até chegar a um nodo que tenha um caracter válido. A função deve retornar o caracter no nodo folha.

- Ex: se caminho tem a configuração a seguir: 0001101100... a busca na árvore deve seguir o caminho esquerda -> esquerda -> esquerda -> direita -> direita... até achar um caracter válido

- 2) (3.5 pontos) Faça uma função que receba dois strings com o nome de dois arquivos binários e verifique se os arquivos são iguais, retornando:

  1 - se são iguais

  0 - se não são

3) (3.5 pontos) Faça uma função que receba o apontador da raiz de uma árvore binária que contem um inteiro em cada nodo e escreva os conteúdos dos nodos por níveis (primeiro a raiz, depois o nível 1, depois o nível 2, etc).

Prova 2 - 07/07/16

- Considere a definição de tipo a seguir, utilizada em uma árvore de huffmann.

  struct nodohuff {unsigned char c;

  unsigned int quant;

  struct nodo *fesq,*fdir;}

- 1) (3.5 pontos) Faça uma função que receba um vetor de pointers para nodos do tipo descrito acima, encontre os dois nodos de menor valor no campo quant e crie um nodo de mesma estrutura, pendurando nele como filhos os dois nodos encontrados. O campo quant do nodo pai deve receber a soma dos campos quant dos dois filhos e o campo c deve receber ' 0'. Considere que algumas posições do vetor contem pointers NULL. A função deve retornar o pointer para o nodo criado e os números das duas entradas da tabela onde estavam pendurados os menores valores.

2) (3.5 pontos) Uma árvore é dita canônica quando a altura da subárvore à direita de qualquer nó nunca é menor que a altura da subárvore à esquerda. Faça uma função que receba a raiz de uma árvore e verifique se é canônica retornando:

- 1 - Se é canônica
- 0 - Se não é

3) (3.0 pontos) Faça uma função que receba um unsigned short int contendo um pixel rgb 16 bits no formato rrrrrgggggggbbbbb e retorne 3 unsigned chars r,g,b, contendo os bits de cada componente no pixel de 16 bits. Os bits devem estar alinhados à direita em cada uma das 3 componentes obtidas.

Recuperação da prova 2b - 2016-2

- 1) (2.5 pontos) Faça uma função que receba o apontador raiz de uma árvore binária e verifique se é uma árvore binária de busca, ou seja, para cada nodo, todos os valores da subárvore à esquerda são menores que ele, e todos os valores da subárvore à direita são maiores do que ele. A função deve retornar:
  - 1 - Se a árvore é uma árvore binária de busca
  - 0 - Se não é uma árvore binária de busca

2) (2.5 pontos) Faça um programa que leia um arquivo texto "entrada.txt"e gere um arquivo "saida.txt"onde todas as ocorrências da palavra "dinamite"foram substituídas pela palavra "azeitona".

- 3) (2.5 pontos) Faça uma função que receba um valor N e o apontador raiz de uma árvore binária de busca, onde cada nodo contem um valor inteiro, e verifica se o valor N existe na árvore, retornando:

  1 - Se o valor N existe na árvore

  0 - Se o valor N não existe na árvore. Nesse caso, a função deve retornar como parâmetro o valor mais próximo a N. Caso hajam dois valores à mesma distância, a função deve retornar o menor deles.

```
int mprox(tnode *raiz,int valor,int *prox)
{
if (raiz->val==valor) return 1;
else if (valor<raiz->val)
{
if (raiz->left==NULL){*prox=raiz->val;return 0;}
int aux=mprox(raiz->left,valor,prox);
if (aux==1) return 1;
if (abs(raiz->val-valor)<abs(*prox-valor)) *prox=raiz->val;
return 0;
}
else
{
if (raiz->right==NULL){*prox=raiz->val;return 0;}
int aux=mprox(raiz->right,valor,prox);
if (aux==1) return 1;
if (abs(raiz->val-valor)<abs(*prox-valor)) *prox=raiz->val;
return 0;
}
}
```

4)(2.5 pontos) Faça uma função que receba 3 unsigned char r,g,b, e retorne um unsigned short int com a composição dos 5 bits mais significativos de r e b, e os 6 bits mais significativos de g, no seguinte formato: rrrrrgggggggbbbbb.

1)(2.5 pontos) Escreva uma função que receba os apontadores de início de duas listas **duplamente encadeadas** em que cada nodo contem um valor inteiro e crie uma lista duplamente encadeada com os elementos que aparecem nas duas listas (interseção das listas), retornando como um terceiro parâmetro da função o apontador do início da lista criada. Considere que as listas não estão ordenadas e que não há repetição de valores em cada lista. E considere que as listas originais não podem ser destruídas.

- Solução: Como o enunciado não especifica a posição dos valores na lista de saída, inseri no início que é mais simpes.

```
void intercala(struct nodo *L1,struct nodo *L2,struct nodo **L3)
    {
    *L3=NULL;
    while (L1)
        {
        struct nodo *ptaux=L2;
        while (ptaux)
            {
            if (L1->valor==L2->valor)
                {
                struct nodo *ptaux2=(struct nodo *)malloc(sizeof(struct nodo));
                ptaux2->prox=*L3;
                *L3=ptaux2;
                ptaux2->valor=L1->valor;
                }
            ptaux=ptaux->prox
            }
        L1=L1->prox
        }
    }
```

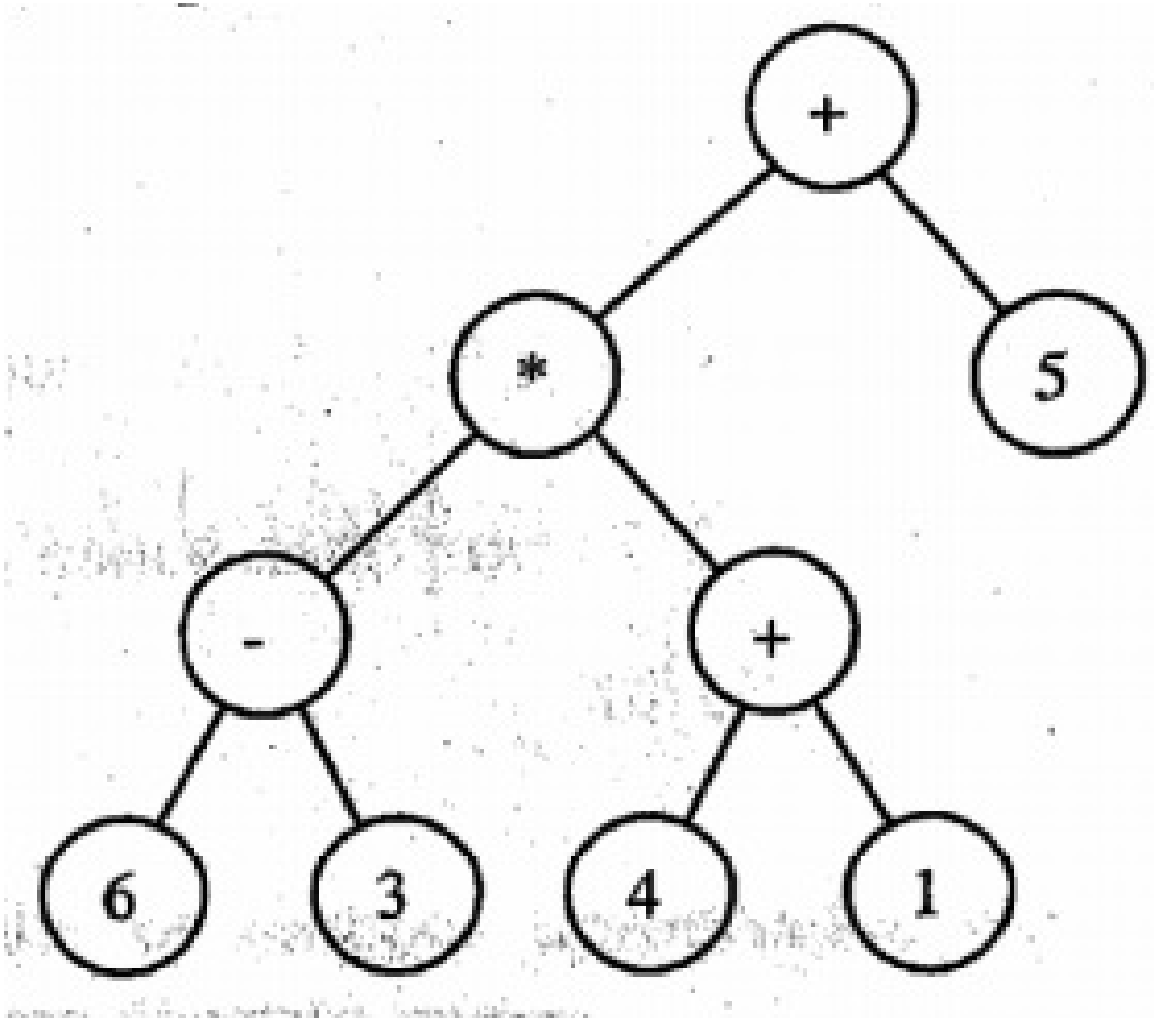
- 2) (2.5 pontos) Escreva uma função que receba o apontador de início de uma lista duplamente encadeada em que cada nodo contem um valor inteiro, o apontador para o próximo nodo e um apontador (inicialmente todos os nodos possuem NULL nesse campo) para o nodo anterior e, acerte em todos os nodos o apontador para o nodo anterior.

```
void acerta_ant(struct nodo *L)
    {
    if (L1==NULL) return;
    while (L1->prox!=NULL)
        {
        L1->prox->ant=L1;
        L1=L1->prox
        }
    }
```

- 3) (2.5 pontos) Escreva uma função que receba um apontador para o início de uma lista simplesmente encadeada com inteiros, e retorne como parâmetros o maior e o menor valor da lista.

- 4) (2.5 pontos) Escreva uma função que receba um apontador para o início de uma lista duplamente encadeada e crie uma cópia da lista duplamente encadeada, retornando como parâmetro o apontador do início da lista criada.

1)(2.5 pontos) Considere uma árvore binária que representa expressões. Por exemplo, a expressão (6-3)*(4+1)+5 é representada pela árvore binária ilustrada na figura abaixo. As folhas da árvore armazenam operando e os nós internos operadores. Se avaliada, essa expressão resulta no valor 20.



- Considere a existência do tipo abaixo usado para representar árvores binárias de expressões:

```
struct arv {
char op;    /* operador: '+','-','*' ou '/' */
float valor; /* valor do operando */
struct arv *esq,*dir;
}
typedef struct arv Arv;
```

- onde o campo *valor* é usado apenas pelas folhas e o campo *op* é usado pelos nós internos. Considere que nos nodos folha (nodos com valores) o campo op possui um caracter em branco.
- Escreva uma função que retorne o valor correspondente à avaliação da expressão. O protótipo da função deve ser:
  - float avalia (Arv* a);

Solução de alguns exercícios

- Ordenar 10 números

```
#include <stdio.h>
int main(){
int i,j,aux,n[10];
for (i=0;i<10;i++)
      scanf("%d",&n[i]);
for (j=0;j<9;j++)
      for (i=0;i<9;i++)
            if (n[i]>n[i+1])
                  {aux=n[i];n[i]=n[i+1];n[i+1]=aux;}
for (i=0;i<10;i++)
      printf("%d\n",n[i]);
}
```

Voltar

- Ordenar 10 datas de aniversário

```
#include <stdio.h>
int main(){
int i,j,aux,d[10],m[10];
for (i=0;i<10;i++)
      scanf("%d%d",&d[i],&m[i]);
for (j=0;j<9;j++)
      for (i=0;i<9;i++)
            if (m[i]>m[i+1] ||m[i]==m[i+1] && d[i]>d[i+1]){
                  aux=m[i];m[i]=m[i+1];m[i+1]=aux;
                  aux=d[i];d[i]=d[i+1];d[i+1]=aux;
                  }
for (i=0;i<10;i++)
      printf("%d/%d\n",d[i],m[i]);
}
```