

# Algoritmos I

$$NF = (P_1 + 2P_2 + 3P_3 + T_1 + T_2 + L) / 8$$

→ MOJ

→ Linux → GCC

- www.brunoribas.com.br
- brunoribas@utfpr.edu.br

processador  
→ pipeline  
→ nops  
→ hiperthread

→ ULA

→ clock máquina

→ memória / cache

→ Problema SAT

$$(a \vee b) \wedge (a \vee \neg b) \wedge (\neg a \vee \neg b)$$

$2^n$

## Tipos Abstratos Dados

int 32 bits - 4 bytes

$2^{32}$  possibilidades

bit ligado 1

desligado 0  
[-32767, +32767]

char 8 bits - 1 byte

$2^8 = 256$  possibilidades

[-128, +127]

long

long 64 bits - 8 bytes  $2^{64}$

unsigned int

float 32 bits - 4 bytes

char

double 64 bits - 8 bytes

...

short 16 bits

## Terceiro

<tipo> <id>[<#>];

int meuvector [10];  $\Rightarrow$  consome 40 bytes

double flutuantes [10];

int meuvector2 [10][10];  $\Rightarrow$  consome 400 bytes

```

int a [10];
double d [10];
char c [10];
char n [10][10];

```

↓ matriz com 10  
 nomes com 10  
 caracteres cada  
 (o último é '\0')

int c [10];  
 posso declarar vetor  
 una struct também

```

minhavar [<#>].a
minhavar [<#>].b
minhavar [<#>].c

```

pl formar  
 → vetor

⇒ man memset

manual  
 pega uma  
 região de memória  
 e coloca 0

## struct

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

```
#define MAX 10
```

```
struct nome
```

```
{
```

int a; // 4 bytes

double b; // 8 bytes

char c; // 1 byte

char p[10], // 10 bytes

```
}
```

nome;

→ tipo novo

```
int main (void)
```

```
{
```

logique

nome minhavar;

```
minhavar.a = 16;
```

```
minhavar.b = 15.83;
```

```
minhavar.c = 'A';
```

```
int a [MAX];
```

```
typedef unsigned int
```

```
#warning "Sei nao hein";
```

07/03

## Pointeiros

```
int a;
```

```
int *b;
```

```
a = 50;
```

```
b = &a;
```

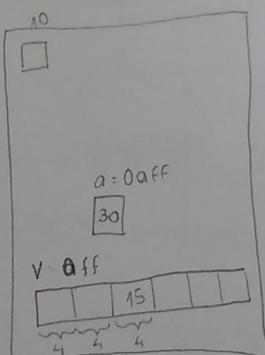
manipulando endereço

$*b = 30$

manipulando valor

$b = 10$

estou dizendo que a



```
int v [10];
```

```
int *c;
```

```
c = &v[2];
```

```
*c = 15;
```

```
c[0] = 15;
```

```
c[1] = 30;
```

$c++$ ; } incrementa ou  
 $c--$ ; } diminui um  
 endereço;

```

int a;
char *b = &a;
b[0] = 'A';
b[1] = 'B';
b[2] = 'C';
b[3] = '\0';
printf ("%s\n", b);

```

⇒ Big engine  
 ⇒ Little engine  
 ⇒ SSE  
 ⇒ MMX

32 bits, 4 bytes  
 ↗ int a, b, g;  
 short int c, d, e, f;

```

a = &a, d = &c[3];
e = &b, f = &e[3];

```

$*c = 10;$

$*d = 20;$

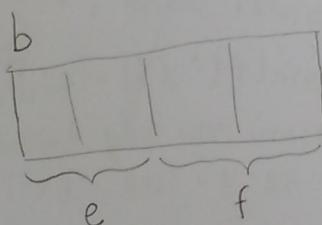
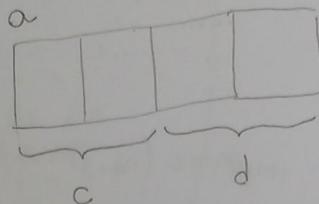
$*e = 22;$

$*f = 32;$

$g = \&g;$

$d = \&c[3];$

$g = a + b;$



$\frac{8 \text{ bits}}{8}$   
 short int a, b;  
 char \*c, \*d, \*e, \*f, \*h, \*i;

$c = \&a;$

$d = \&c[3];$

$e = \&b;$

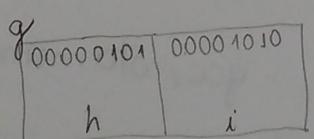
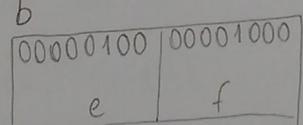
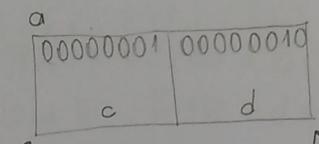
$f = \&e[3];$

$*c = 1; // 00000001$

$*d = 2; // 00000010$

$*e = 4; // 00000100$

$*f = 8; // 00001000$



$g = a + b;$

$h = \&g;$

$i = \&h[3];$

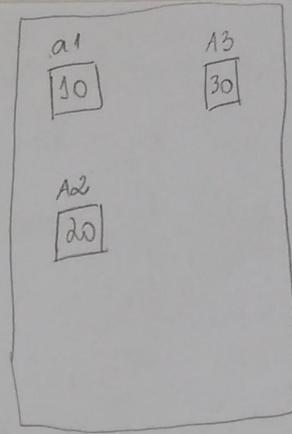
use forse  $d = \&a[3]$   
 ele apontaria aqui

↓ 8 bits      ↓ 8 bits

```

void troca (int *a, int *b)
{
    int c = *a;
    *a = *b;
    *b = c;
}

```



```

int main (void)
{
    int a1, a2, a3;
    a1 = 10;
    a2 = 20;
    troca (&a1, &a2);
}

```

$$a[0] = 10 \Leftrightarrow *a = 10$$

```

typedef a_st
{
    int a;
    double b;
} a_st;

```

```

int main (void)
{
    a_st a;
    a.a = 20;
    a.b = 30.546228;
    return 0;
}

```

```

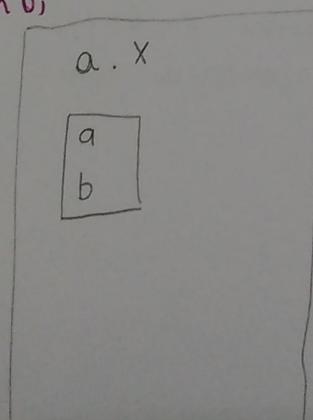
void imprime (a_st x)
{
    printf ("%d\n", x.a);
    printf ("%lf---%f", x.b)
}

```

```

void f (a_st *x)
{
    x->a = 65; // x[0].a = 65;
    x->b = 127.2; // x[0].b = 127.2;
}

```



ls

gcc olamamae.c -o olamamae

ls

./olamamae

./Ex-B < entrada\_B

11/03/16

→ printf retorna o número de bytes que ele conseguiu imprimir com sucesso

```
int main (void)
{
    int N;
    scanf ("%d", &N);
    while (N != 0)
    {
        calculo_jogada (N);
        scanf ("%d", &N);
    }
    return 0;
}
```

→ scanf retorna a qtd de %s que ele leu com sucesso

```
void calculo_jogada (int N)
{
    int a, x, alic = 0, beto = 0;
    for (i = 0; i < N; i++)
    {
        scanf ("%d", &x);
        if (x == 0)
        {
            alic++;
        }
        else
        {
            beto++;
        }
    }
}
```

printf ("Alice ganhou %d e Beto ganhou %d \n", alic, beto);

End Of File

^d

Posso substituir os dois scanf por um único  
while (scanf ("%d", &N) = 1 && N != 0)

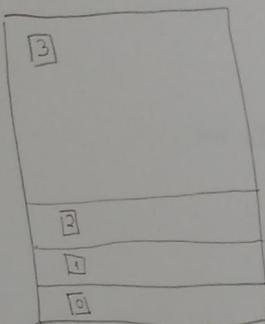
## Recursividade

fat {  
 se x=0 então fat é 1  
 se m fat é  $(x)^*((x-1)!)$

```

int fat(int x)
{
    int f = 1, i;
    for (i = x; i > f; i--)
    {
        f = f + i;
    }
    return f;
}

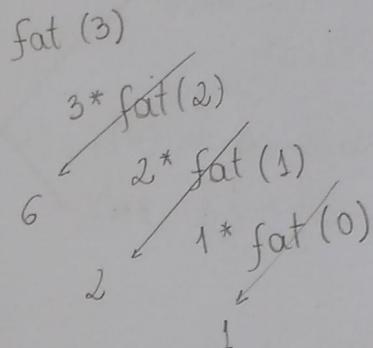
```



```

int fat(int x)
{
    if (x == 0) return 1;
    return x * fat(x - 1);
}

```



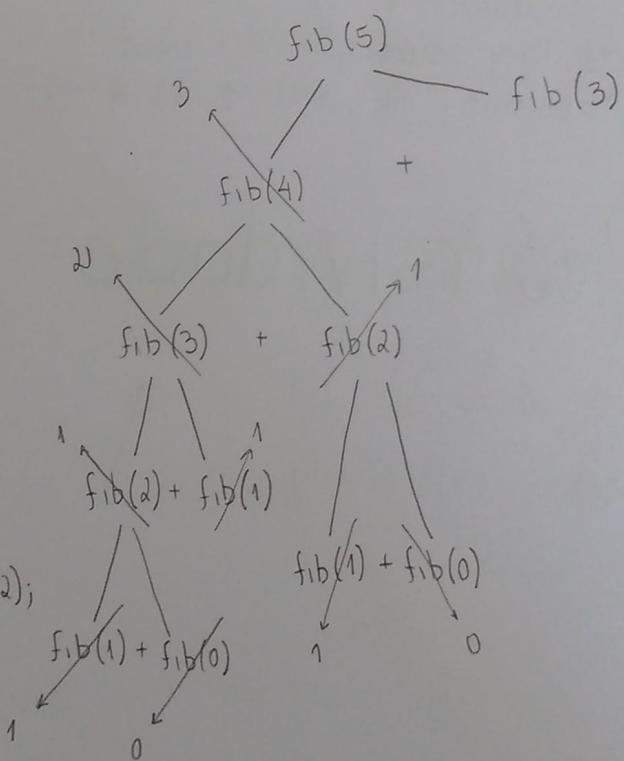
## Fibonacci

$$\begin{aligned} f(0) &= 0 \\ f(1) &= 1 \\ f(n) &= f(n-1) + f(n-2) \end{aligned}$$

```

int Fib(int n)
{
    if (n == 0) return 0;
    if (n == 1) return 1;
    return Fib(n-1) + Fib(n-2);
}

```



int memoria[10],  
 $\{ 0, 1, -1, -1, -1, -1, -1, -1, -1, -1 \}$   
0 1 2 3 4 5 6 7 8 9

memoria[0] = 0;

memoria[1] = 1,

int fib(int x)

{

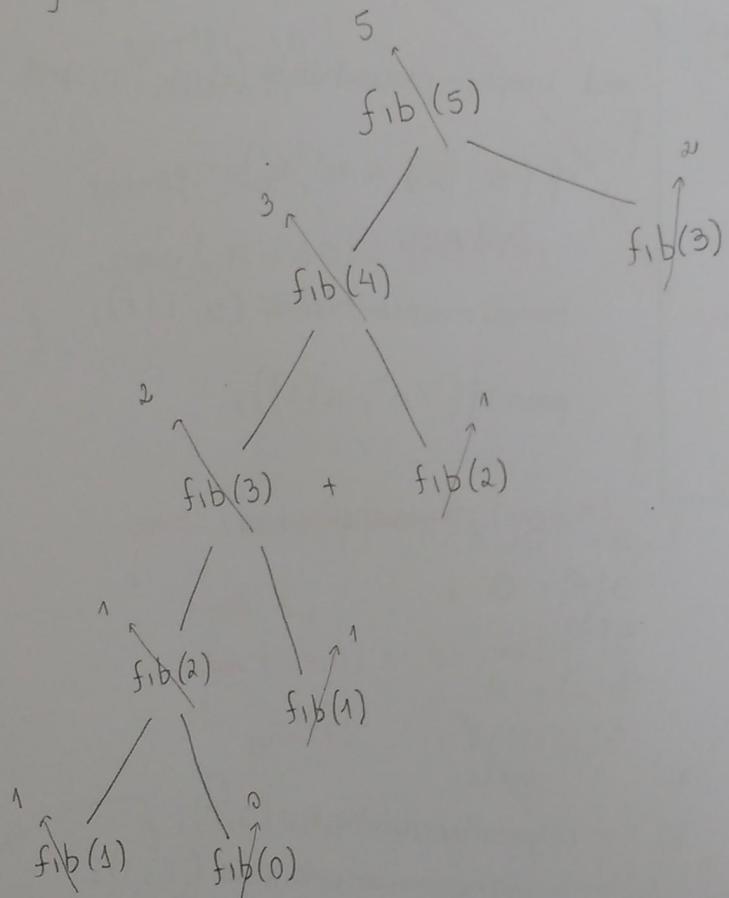
if (memoria[x] != -1)

return memoria[x];

memoria[x] = fib(x-1) + fib(x-2);

return memoria[x];

}



14/03/16

→ Entrada: uma string s [len 1000]

→ Saída: string ao contrário

```
#include <stdio.h>
#include <string.h>

int main (void)
{
    int tam, i;
    char string[1001];

    scanf ("%s", string);
    tam = strlen(string);

    for (i = (tam - 1); i >= 0; i++)
    {
        printf ("%c", string[i]);
    }
    printf ("\n");
    return 0;
}
```

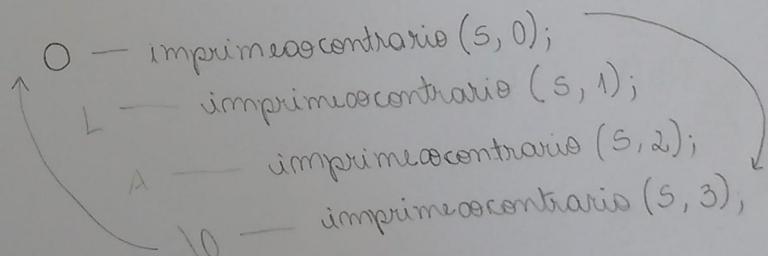
```
void imprimeocontrario (char *s)
{
    if (*s == '\0')
        return;
    imprimeocontrario (s + 1);
    printf ("%c", *s);
}
```

O L A 10  
s[0] s[1] s[2] s[3]

```
int main (void)
{
    char s[1001];
    scanf ("%s", s);
    imprimeocontrario (s, 0);
    return 0;
}
```

```
void imprimeocontrario (char *s, int pos)
{
    if (s[pos] == '\0')
        return;
    imprimeocontrario (s, pos + 1);
    printf ("%c", s[pos]);
}
```

s = "OLA\0"  
s[0] = 'O'  
s[1] = 'L'  
s[2] = 'A'  
s[3] = '\0'



```
int main (void)
```

```
{
```

```
    int N;
```

```
    scanf ("%d", &N);
```

```
    iaoe (N);
```

```
    return 0;
```

```
}
```

forma iterativa

```
void iaoe (int N)
```

```
{
```

```
    while (N != 0)
```

```
{
```

```
    printf ("%d", N % 10);
```

```
    N = N / 10;
```

```
}
```

```
    printf ("\n");
```

```
}
```

4 3 2 1

forma recursiva

```
void iaoe (int N), int prof)
```

```
{
```

```
    if (N == 0){
```

```
        printf ("\n");
```

```
        return;
```

```
}
```

```
    printf ("%d", N % 10);
```

```
    iaoe (N / 10), prof + 2);
```

```
}
```

```
iaoe (1234)
```

```
iaoe (123)
```

```
iaoe (12)
```

```
iaoe (1)
```

```
iaoe (0)
```

```
imprimeespacos (prof);
```

```
void imprimeespacos (int N)
```

```
{
```

```
    int i;
```

```
    for (i = 0; i < N; i++)
```

```
{
```

```
    printf (" ");
```

```
}
```

```
}
```

# MDC

$$\text{MDC}(A, B) = \text{MDC}(B, u)$$

$$u = A \% B$$

int MDC (int a, int b)

```
{ if (b == 0)
    return a;
    return mdc(b, a % b);
}
```

MDC(314159, 271828)

MDC(271828, 42331)

MDC(42331, 14824)

MDC(14824, 6644)

MDC(6644, 4548)

MDC(4548, 2099)

MDC(2099, 350)

MDC(350, 349)

MDC(349, 1)

MDC(1, 0)

int achamaior(int \*v, int tam)

```
{ int maior = v[0];
    int i;
    for (i=1; i < tam; i++)
    { if (v[i] > maior)
        maior = v[i];
    }
    return maior;
}
```

18/03/2016

\* Ler o manual das funções

strcmp(char \*str1, char \*str2)

→ compara 2 strings

- Man

- Man stdio.h

Ex 1 - Faixa 2

```
#include <stdio.h>
int main(void)
{
    int n, cont = 0;
    while (scanf("%d", &n) == 1) // EOF
    {
        cont++;
        printf("%d\n", cont);
    }
    return 0;
}
```

Ex 2 - Faixa 2

```
int main(void)
{
    char resp[4];
    int i, simS = 0, triagem = 0;
    while (scanf("%s", resp) == 1)
    {
        if (resp[0] == 's') simS++;
        for (i=0; i < 9; i++)
        {
            scanf("%s", resp);
            if (resp[0] == 's') simS++;
        }
        if (simS >= 2) triagem++;
        simS = 0;
    }
    printf("%d\n", triagem);
    return 0;
}
```

EOF

## Kilos I

```

int n; cont=0, ret;
ret = scanf("%d", &n);
while (ret == 1)
{
    cont++;
    ret = scanf("%d", &n);
}

```

## ESTRUTURA BÁSICA DE LAGO

int

```

scanf("%d", &n);
while (n != 0)
{
    ...
    scanf("%d", &n);
}

```

⇒ no bloco 3 eu poderia ter feito

ret = scanf("%d : %d : %d", &h, &m, &s);

while (ret == 3)

    } porque ele  
    vai ler 3

~~fflush(stdin); ⇒ não é POSIX e é  
imutável~~

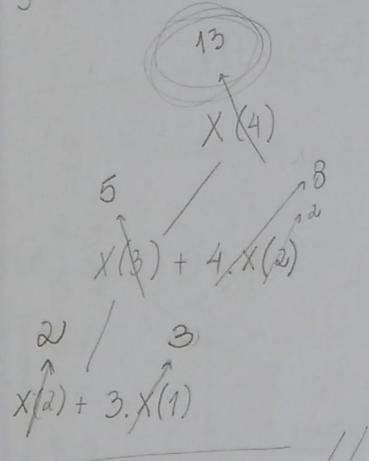
~~fflush(stdout); ⇒ faz sentido~~

scanf(" %c", &c);  
 ↓ consome tudo  
 p/ min

```

int X (int n)
{
    if (n == 1 || n == 2) return n;
    return X(n-1) + n * X(n-2);
}

```

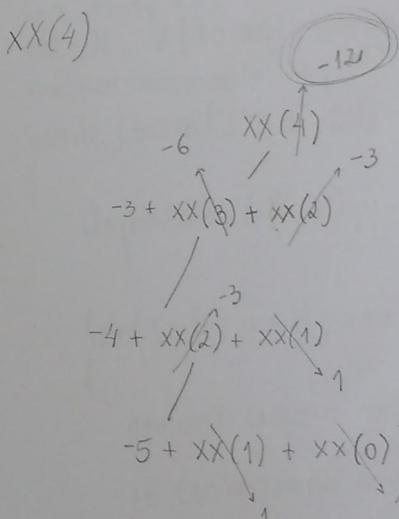


int XX(int i)

```

{
    if (i > 1)
        return (XX(i-1) + XX(i-2));
    return 1;
}

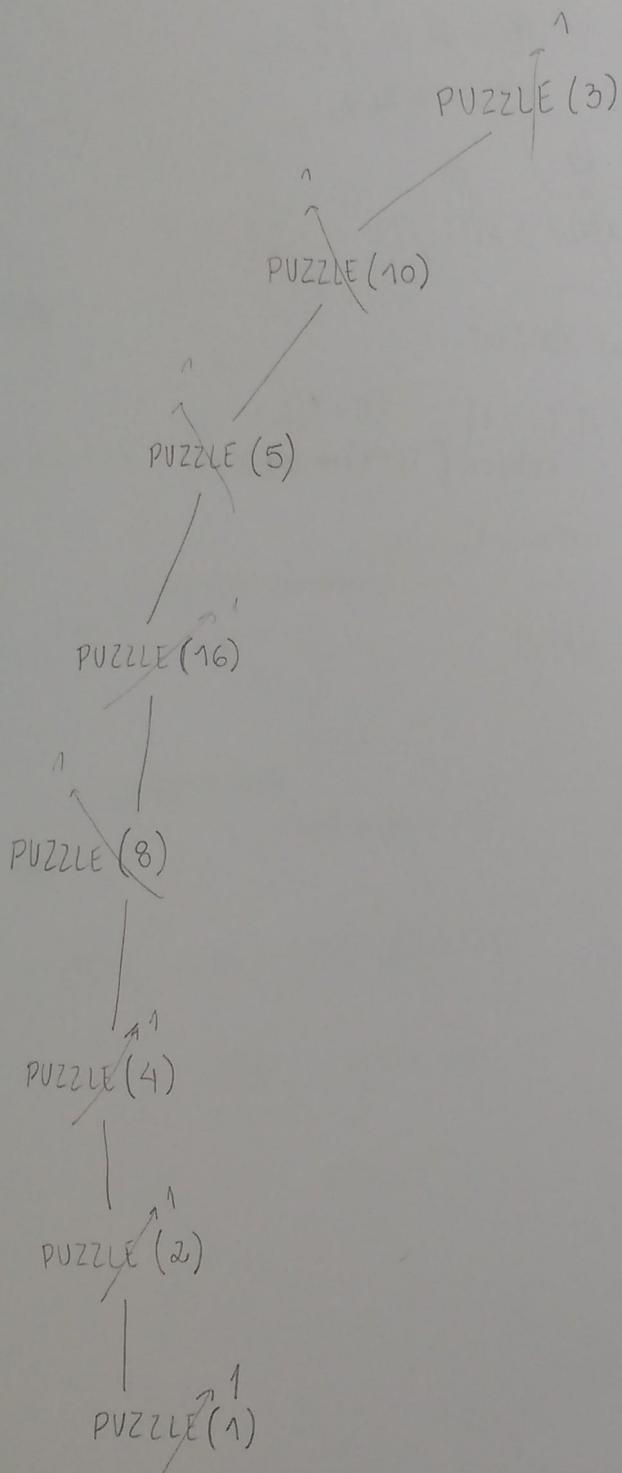
```



```

int puzzle (int N)
{
    if (N == 1) return 1;
    if (N % 2 == 0)
        return puzzle (N/2);
    return puzzle ([3 * N] + 1);
}

```



21/03

maior elemento do vetor

```

int maximo I (int N, int * v)
{
    int i;
    int maior = v[0];
    for(i=1; i < N; i++)
    {
        if(v[i] > maior)
        {
            maior = v[i];
        }
    }
    return maior;
}

```

int maximo R (int N, int \* v)
{
 if (N == 1)
 return v[0];
 maior = v[N-1];
 maiorR = maximo R (N-1, v);
 if (maior > maiorR)
 return maior;
 return maiorR;
}

deve tudo  
até o final  
pl/ depois  
comparar e  
descobrir o  
maior

N=5  
 $v = \{14, 1, 8, 0, 3\}$

$\maximo R(5, v) \rightarrow 14$

$\maximo R(5, v)$   
 $maior = 3 \rightarrow 14$

$\maximo R(4, v) \rightarrow 14$

$\maximo R(4, v)$   
 $maior = 0 \rightarrow 14$

$\maximo R(3, v) \rightarrow 14$

$\maximo R(3, v)$   
 $maior = 8 \rightarrow 14$

$\maximo R(2, v) \rightarrow 14$

$\maximo R(2, v)$   
 $maior = 1 \rightarrow 14$

$\maximo R(1, v) \rightarrow 14$

$\maximo R(1, v)$   
 $maior = 14$

uint removenulos (int N, int \*v)

{

    int i, contn = 0;

    for(i=0; i < N; i++)

}

    if(v[i] == 0)

{

        int j = i + 1;

        while(j < N && v[j] == 0)

{

            j++;

}

        if(j < N){

            v[i] = v[j];

            v[j] = 0;

}

    if(v[i] != 0)

        contn++;

}

    return contn;

}

28/03/16

% i    v[5]    % d



Posta Especial 3 - J

2

saldo

2 3 (-1)

4 1 (6)

9 saldo

2 2 φ

0 5 -5

6 2 4  
1 4 -3  
0 0 0  
5 1 4  
1 5 -4  
6 2 4  
0 5 -5

typedef struct saldo  
{  
    int inicio, fim, soma;  
} saldo;

int main (void)

{  
    int teste = 1;  
    int N, V[10000], encontrado, saldo  
    melhor.soma = -1;  
    while (scanf ("%d", &N) == 1 && N != 0)

{

    lepartidas (N,V);

    comecaem = 0;

    for(i=0; i < N; i++)

{

        encontrado = mvapd (i,N,V);

        if (encontrado.soma > melhor.soma)

{

            melhor = encontrado;

        } comecaem = i+1;

}

    imprimirposta (teste++, melhor);

}

```

int mvapd (int inicio, int fim, int * difs)
{
    saldo ret; ret.inicio = inicio + 1; saldo
    int soma = difs [inicio], maiorSoma = soma;
    inicio++;
    while (soma >= 0 && inicio < fim)
    {
        soma = soma + difs [inicio];
        inicio++;
    }
    if (maiorSoma <= 0)
    {
        ret.soma = -1
        return ret;
    }
    ret.soma = maiorSoma;
    ret.fim = inicio;
    return ret;
}

```

$i=0 \Rightarrow soma = -1$   
 $i=1 \Rightarrow soma = 6$

$(0, 2, \{-1, 6\})$

$$ret \begin{cases} inicio = 1 \\ fim = \\ soma = -1 \end{cases}$$

$f (soma >= maiorSoma)$   
 $\{ maiorSoma = soma;$   
 $ret.fim = inicio;$

imprimir resposta

$<=$

nenhum

01/04/2016

## Pilha

→ F I L O

I	N	A	U
R	S	T	
S	T		
T			

↳ empilha (push)

↳ desempilha (pop)

↳ está vazia (isempty)

↳ inicializa-pilha ( )

#define MAXP 100

```
typedef struct pilha
{
    s_type vet[MAXP];
    int cont;
} pilha;
```

define true 1

define false 0

define s\_type char

```
void inicializa_pilha(pilha * p)
{
    p->cont = 0
}
```

```
int empilha(pilha * p, int item)
{
    if (p->cont >= MAXP)
        return false;
    p->vet[p->cont++] = item;
    return true;
}
```

```
s_type desempilha(pilha * p)
{
    p->cont--;
    return p->vet[p->cont];
}
```

```
int esta_vazia(pilha * p)
{
    if (p->cont == 0)
        return true;
    return false;
}
```

ultimo que entrou em pilha

ou    p->vet[p->cont] = item;  
      p->cont++

```
s_type desempilha(pilha * p)
{
    return p->vet[--p->cont];
}
```

if(esta\_vazia(&p) == true)

preciso sair

```

int main (void)
{
    int i, tam → pilha p;
    char s [50];
    scanf ("%s", s) → inicializa_pilha (&p);
    tam = strlen (s);

    for (i=0; i < tam; i++)
    {
        if (s[i] == '(' || s[i] == '[')
            empilha (&p, s[i]);
        else if (s[i] == ')')
        {
            if (desempilha (&p) != '(')
                saida_desonrosa ();
        }
        else if (s[i] == ']')
        {
            if (desempilha (&p) != ']')
                saida -
        }
    }
    if (esta_vazia (&p))
        printf ("OK, legal\n");
    else
        saida_desonrosa (i);
}

```

```

int main (void)
{
    pilha p;
    inicializa_pilha(&p);
    int lido;
    while (scanf ("%d", &lido) == 1)
        empilha (&p, lido);
    printf ("%d", desempilha (&p));
    while (!esta_vazia (&p))
        printf ("%d", desempilha (&p));
    printf ("\n");
    return 0;
}

```

Entrada : 1 2 3 4

Saída : 4 3 2 1

```

int main (void)
{
    pilha p, p2;
    inicializa_pilha(&p);
    inicializa_pilha(&p2);
    int lido;
    while (scanf ("%d", &lido) == 1)
        empilha (&p, lido);
    while (!esta_vazia (&p))
        empilha (&p2, desempilha (&p));
    printf ("%d", desempilha (&p2));
    while (!esta_vazia (&p2))
        printf ("%d", desempilha (&p2));
    printf ("\n");
    return 0;
}

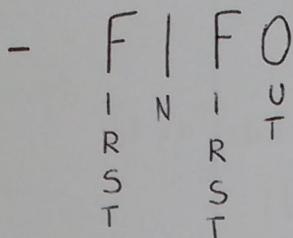
```

Entrada : 1 2 3 4

Saída : 1 2 3 4

stype aux = desempilha (&p);  
 empilha (&p2, aux);

# Fila



- Inicializa - fila
- enfila
- desenfila
- esta - vazia

```
#define f-type int
typedef struct file
{
    f-type vet[MAXF];
    int cont, inicio, fim;
} fila;
```

```
void inicializafila (fila *f)
{
    f->cont = 0; f->inicio = 0; f->fim = 0;
}
```

```
int esta_vazia (fila *f)
{
    if (f->cont == 0) return true;
    return false;
}
```

vet [fila.fim]

FILA(F) → copia ini  
 F → vet[fim]

if (f->cont > MAXF)  
 return false;

f->vet[f->fim] = elem;  
 cont++; f->fim = ((f+1)%MAXF);  
 return true;

f-type desenfila (fila \*f)

f-type ret = f->vet[f->inicio];  
 f->inicio = ((f+1)%MAXF);  
 int i;  
 for (i=1; i < cont; i++)  
 {  
 f->vet[i-1] = f->vet[i];  
 }  
 f->cont --;  
 return ret;

↓ tem custo linear (N)

↓ f->inicio ++;

```

int main (void)
{
    fila f;
    int num;
    inicializafila (&f);
    while (scanf ("%d", &num) == 1)
    {
        enfila (&f, num);
    }
    impcont (&f);
}

```

```

void impcont (fila *f)
{
    int n;
    if (esta_vazia (f)) return;
    n = desenfila (f);
    impcont (f);
    printf ("%d\n", n);
}

```

forma  
recursiva

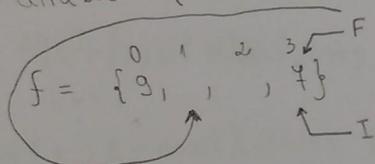
void imprime cont(fila \*f)

```

pilha p;
inicializa_pilha (&p);
while (! esta_vazia (f))
{
    empilha (&p, desenfila (f));
}
while (! estavazia (&p))
{
    printf ("%d\n", desempilha (&p));
}

```

pra fazer ele voltar  
de fim pro  
início (Fila circular)



08/04/2016

# Lista

- Inicializa (pilha, fila)

- esta\_vazia (p/ não continuar empilhando ou enfileirando)

→ Inicializa

→ Remove

→ Insere

↳ Remove no fim

desempilha

↳ Insere no fim

↳ Remove no inicio

desenfila

↳ Insere no inicio

↳ Remove em K

↳ Insere em K

→ Busca

```
typedef struct lista
{
    l-type vet[MAXL];
    int cont;
    int inicio, fim;
} lista;
```

```
void inicializa_lista(lista *L)
{
    inicializa_fila(L);
}

int insere_fim(lista *L, l-type elem)
{
    return enfileira(L, elem);
}
```

```
int insere_inicio(lista *L, l-type elem)
{
    if (L->cont > MAXL)
        return false;
    L->inicio = (MAXL + (L->inicio - 1) % MAXL);
    L->vet[L->inicio] = elem;
    L->cont++;
    return true;
}

int insere_emk(lista *L, l-type elem, int k)
{
    if (L->cont > MAXL)
        return false;
    pilha aux;
    inicializa_pilha(&aux);
    int i;
    for(i=0; i < (k-1); i++)
        empilha(&aux, remove_inicio(L));
    insere_inicio(L, elem);
    for(i=0, i < (k-1); i++)
        insere_inicio(L, desempilha(&aux));
    L->cont++;
    return true;
}
```

```
l-type remove_K(lista *L, int K)
{
    l-type ret;
    pilha aux;
    inicializa_pilha(&aux);
    int i;
    for(i=0; i<(K-1); i++)
        empilha(&aux, remove_inicio(l));
    ret = remove_inicio(L);
    for(i=0; i<(K-1); i++)
        insere_inicio(L, desempilha(&aux));
    l->cont--;
    return ret;
}
```

```
int busca(lista *L, l-type elem)
{
    int i = l->inicio, k = 1;
    while(k <= l->cont && L->vet[i] != elem)
        i = (i+1) % MAXL;
    if(k > l->cont)
        return -1;
    return k;
}
```

25/04/2016

## Garbage Collector

```
#include <stdlib.h>
```

```
void *malloc (size_t size);  
free (void *ptr);
```

```
void *realloc (void *ptr, size_t size);
```

```
int main (void)  
{  
    char *v, *a;           NULL  
    v = (char *) malloc (1000 * sizeof (char));  
    if (v == NULL)  
        printf ("socorro\n");  
    a = (char *) malloc (500 * sizeof (char));  
    exit (1);  
}
```

```
int main (void)      i * j = 500 * 300  
{  
    int **m;  
    m = (int **) malloc (500 * sizeof (int));  
    for (i = 0; i < 500; i++)  
        m[i] = (int *) malloc (300 * sizeof (int));  
  
    v = alocar_vetor (300);  
}  
  
int *alocar_vetor (int tam)  
{  
    int *r;  
    r = (int *) malloc (tam * sizeof (int));  
    if (r == null)  
        saida_com_erro  
    return r;
```

Lista Simplemente Enc. 29/04/16

\* malloc retorna um ponteiro do tipo void, por isso faço um typecasting

```
typedef struct lista
{
    no *inicio;
    int cont;
} lista;
```

```
typedef struct no
{
    l-type elem;
    no *prox;
} no;
```

```
void inicializa_lista(lista *L)
{
    L->inicio = NULL;
    L->cont = 0;
}
```

```
int insere_inicio (lista *L, l-type elem)
{
    no *novono = (no*) malloc (sizeof (no));
    if (novono == NULL)
        return false;
```

```
novono->elem = elem;
novono->prox = L->inicio;
L->inicio = novono;
L->cont++;
return true;
```

}

```
l-type remove_inicio (lista *L)
{
    l-type ret = L->inicio->elem;
    no *arm = L->inicio;
    L->inicio = L->inicio->prox;
    free (arm);
    L->cont--;
    return ret;
}
```

```

int insere_fim (lista *L, l-type elem)
{
    no *tmp;
    no *novono = (no *) malloc (sizeof (no));
    if (novono == NULL)
        return false;
    novono->elem = elem;
    novono->prox = NULL;
    if (L->inicio == NULL)
        L->inicio = novono;
    else
    {
        tmp = L->inicio;
        while (tmp->prox != NULL)
            tmp = tmp->prox;
        tmp->prox = novono;
    }
    L->cont++;
    return true;
}

```



E SE SÓ TIVER UM NULL?

```

l-type remove_fim (lista *l)
{
    no *tmp = l->inicio; ant = l->inicio;
    while (tmp->prox != NULL)
    {
        ant = tmp;
        tmp = tmp->prox;
    }
    l-type ret = tmp->elem;
    if (l->inicio == tmp)
        l->inicio = NULL;
    ant->prox = NULL;
    free (tmp);
    L->cont--;
    return ret;
}

```

```
void remove_no (Jirafa *L, no *rm)
{
    if (rm != L->inicio)
    {
        no *tmp = L->inicio;
        while (tmp->prox == rm)
            tmp = tmp->prox;
        tmp->prox = rm->prox;
        free(rm);
        L->cont--;
    }
    else
        remove_inicio(L);
}
```

```
int busca (Jirafa *L, L-type elem)
{
    no *tmp = L->inicio;
    int k = 0;
    while (tmp->elem != elem && tmp->prox != NULL)
    {
        tmp = tmp->prox;
        k++;
    }
    if (tmp->elem == elem)
        return k;
    else
        return -1;
}
```

```

int insere_ordenado(lista *L, l_type elem)
{
    if (L->inicio == NULL)
        return insere_inicio(L, elem);
    no *novono = (no*)malloc(sizeof(no));
    novono->elem = elem;
    novono->atual, anterior = L->inicio;
    while (eh_menor(atual->elem, elem) == 1)
    {
        ant = atual;
        atual = atual->prox;
    }
    novono->prox = atual;
    ant->prox = novono;
}

```

//eh\_maior retorna 1 se o primeiro > segundo  
 0 se forem iguais  
 -1 se o segundo > primeiro

02/05/16

## Livata Duplamente Encadeada

```

typedef struct no no;
struct no
{
    l_type elem;
    (struct) no *prox,
    (struct) no *ant;
};

```

```

typedef struct lista
{
    int cont;
    no *inicio;
} lista;

```

```

int insere_inicio(lista *L, l_type elem)
{
    no *novono = (no*)malloc(sizeof(no));
    if (novono == NULL) return false;
    novono->elem = elem;
    novono->prox = L->inicio;
    if (L->inicio != NULL)
        L->inicio->ant = novono;
    novono->ant = NULL;
    L->inicio = novono;
    L->cont++;
    return true;
}

```

```
int _insere_apos(no *anterior, l-type elem)
{
    no *novo = (no *) malloc(sizeof(no));
    if (novo == NULL) return false;
    novo->elem = elem;

    novo->prox = anterior->prox;
    if (anterior->prox != NULL)
        anterior->prox->ant = novo;
    novo->ant = anterior;
    anterior->prox = novo;
}
```

```
l-type _remove (no *rm)
{
    if (rm->ant != NULL)
        rm->ant->prox = rm->prox;
    if (rm->prox != NULL)
        rm->prox->ant = rm->ant;
    l-type ret = rm->elem;
    free(rm);
    return ret;
}
```

```
l-type remove (list &L, no *rm)
{
    l-type ret;
    L->cont--;
    ret = _remove(rm);
    if (L->cont == 0)
        L->inicio = NULL;
    return ret;
}
```

```
void troca2(no* ptr1, no* ptr2)
{
    no* aux = ptr1->prox, *aux2 = ptr2->ant;
    ptr1->prox = ptr2->prox;
    ptr2->prox->ant = ptr1;
    ptr2->prox = aux;
    ptr2->ant = aux2;
}
```

06/05/2016

```
void remove2 (no * ptr1, no * ptr2)
{
    ptr1 → ant → prox = ptr1 → prox;
    ptr1 → prox → ant = ptr1 → ant;
    free (ptr1);
    ptr2 → ant → prox = ptr2 → prox;
    ptr2 → prox → ant = ptr1 → ant;
    free (ptr2);
}
```

```
void removecaminho (no * ptr1, no * ptr1)
{
    if (ptr1 → prox == ptr2) return;
    no * aux = ptr1 → prox;
    ptr2 → ant → prox = NULL;
    ptr1 → prox = ptr2;
    ptr2 → ant = ptr1;
    while (ant → prox != NULL)
    {
        aux = aux → prox;
        free (aux → ant);
    }
    free (aux);
}
```

```
void removecaminho2 (no * ptr1, no * ptr2)
{
    no * aux = ptr1 → prox;
    while (aux != ptr2)
    {
        ptr1 → prox = aux → prox;
        aux = aux → prox;
        free (aux → ant);
        aux → ant = ptr1;
    }
}
```

```
struct aluno  
{  
    char nome [30];  
    int nota;  
}
```

```
struct aluno *a  
a = (struct aluno *) malloc (2 * sizeof (struct aluno));
```

```
typedef struct lista  
{  
    no * inicio;  
    int cont;  
}
```

```
typedef struct no  
{  
    int elem;  
    struct no * ant, prox;  
}
```

```
lista *divide (no * ptr1)  
{  
    lista * ls = (lista *) malloc (2 * sizeof(lista));  
    inicializa_lista (&ls[0]);  
    inicializa_lista (&ls[1]);  
    while (ptr1 != NULL)  
    {  
        aux = ptr1 -> prox;  
        if (ptr1 -> elem / 2 == 0)  
            adiciona (ptr1, &ls[1]),  
        else  
            adiciona (ptr1, &ls[0]);  
        ptr1 = aux;  
    }  
    return ls;  
}
```

```
void adiciona (no * quem, lista * aonde)  
{  
    quem -> ant = NULL;  
    quem -> prox = aonde -> inicio;  
    if (aonde -> inicio != NULL)  
        aonde -> inicio -> ant = quem;  
    aonde -> inicio = quem;  
}
```

09/05/2016

typedef struct no

{  
    int elem;  
    struct no \*ant, \*prox;  
} no;

no \*divide (no \*inicio)

{  
    no \*par<sup>=NULL</sup>, \*impar = NULL, \*PROX;

while (inicio != NULL)

{  
    prox = inicio → prox;

    if (inicio → elem % 2 == 0)

        par = insere\_no\_fim (par, inicio);

    else

        impar = insere\_no\_fim (impar, inicio);

    inicio = prox;

}

no \* insere\_no\_fim (no \*l, no \*add)

{  
    no \*nl = l;

    if (l == NULL)

{  
    add → ant = NULL;

    add → prox = NULL;

    return add;

}

while (l → prox != NULL)

    l = l → prox;

    l → prox = add;

    add → ant = l;

    add → prox = NULL;

    return nl;

}

typedef struct parimpar

{  
    struct no \*par, \*impar

} parimpar

parimpar ret;

ret. par = par;

ret. impar = impar;

return ret;

```

struct no
{
    char tipo; // T, D
    int dist;
    struct no *prox, *desvio;
}

```

```

int eh_posivel (struct no *ptr1, struct no *ptr2)
{
    struct no *atual;

```

```

    pilha p;
    inicializa - pilha (&p);
    atual = ptr1;

```

```

    while (atual != ptr2)
    {

```

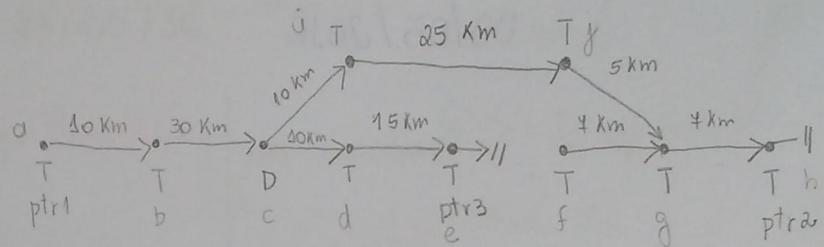
```

        empilha (&p, atual -> prox);
        if (atual -> tipo == 'D')
            empilha (&p, atual -> desvio);

```

```

        atual = desempilha (&p);
    }
}
```



Algoritmos Elementares de Ordenação:

→ Bubble Sort

→ Selection

→ Insertion

Bubble

```
void bolha(int *v, int tam)
{
    int i, j;
    for(i=0; i < tam; i++)
    {
        for(j=0; j < tam-1; j++)
        {
            if(v[j] > v[j+1])
                troca(&v[j], &v[j+1]);
        }
    }
}
```

⇒ É estável;

↳ mantém a ordem relativa dos elementos.

⇒ Não é adaptativo;

↳ não vai sair mais rápido, não vai percorrer menos ou ter sua complexidade diminuída.

⇒ Complexidade  $n^2$

→ É ruim em todos os casos

```
void selecao(int *v, int tam)
{
    int i, j, min;
    for(i=0; i < tam-1; i++)
    {
        min = i;
        for(j=i+1; j < tam; j++)
        {
            if(v[j] < v[min])
                min = j;
        }
        troca(&v[i], &v[min]);
    }
}
```

⇒ É estável;

⇒ Não é adaptativo;

⇒ Complexidade  $n^2$ ;

30 / 05

## Insertion Sort

```
void insertion(int *vet, int tam)
{
    int i, j, x;
    for(j=1; j < tam; j++)
    {
        x = vet[j];
        for(i=j-1; i >= 0 && vet[i] > x; i--)
        {
            vet[i+1] = vet[i];
        }
        vet[i+1] = x;
    }
}
```

Complexidade  $n^2$  (casos médio e pior)

É adaptativo; (muda menos da ordenação)

É instável;

```
int buscaseq(int *vet; int elem, int tam)
{
    int i=0;
    for(i=0, i < tam; i++)
        if(vet[i] == elem)
            return i;
    return -1;
}
```

```
int buscaSeqOrd(int *vet, int tam, int elem)
{
    int i=0;
    while(i < tam && vet[i] < elem)
        i++;
    if(vet[i] == elem)
        return i;
    return -1;
}
```

```
int bb(int x, int e, int d, int *v)
{
    if(e > d) return -1;
    int m = (e + d) / 2;
    if(v[m] == x) return m;
    if(v[m] < x)
        return bb(x, m+1, d, v);
    else
        return bb(x, e, m-1, v);
}
```

```
int busca_binaria(int x, int *v, int tam)
{
    return bb(x, 0, tam-1, v);
}
```

06/06/2016

## Quick Sort

```
void QuickSort(int *v, int l, int u)
{
    int i;
    if (u <= l) return;
    i = separa(v, l, u);
    QuickSort(v, l, i-1);
    QuickSort(v, i+1, u);
}
```

//

⇒ n é estável

⇒ complexidade  $\approx n \log n$  (médio)

⇒ pior caso  $O(n^2)$

```
int separa(int *v, int p, int u)
{
    int w = v[p], i = p+1, j = u;
    while (i <= j)
    {
        if (v[i] <= w)
            i++;
        else if (v[i] > w)
            j--;
    }
    else
    {
        troca(&v[i], &v[j]);
        i++;
        j--;
    }
}
v[p] = v[j];
v[j] = w;
return j;
```

//  $p \leq u \leq (p+u)/2$

int w = v[p];

if ((c > v[u] && v[u] > v[m]) ||  
(c < v[u] && v[u] < v[m]))

med = u;

else if ((v[u] > c && c > v[m]) ||  
(v[u] < c && c < v[m]))

med = p;

else

med = (p+u)/2;

troca(&v[p], &v[med]);

10/06/2016

## - Merge Sort

```
void MergeSort(int *v, int p, int r)
{
    if (p < r - 1)
    {
        int q = (p + r) / 2;
        MergeSort(v, p, q);
        MergeSort(v, q + 1, r);
        intervala(v, p, q, r);
    }
}
```

```
void intervala(int *v, int p, int q, int r)
{
    int i, j, k, *w;
    w = malloc((r - p) * sizeof(int));
    k = 0;
    i = p;
    j = q + 1;
    while (i <= q && j <= r)
    {
        if (v[i] <= v[j])
            w[k++] = v[i++];
        else
            w[k++] = v[j++];
    }
    while (i <= q)
        w[k++] = v[i++];
    while (j <= r)
        w[k++] = v[j++];
    for (i = p; i <= r; i++)
        v[i] = w[i - p];
    free(w);
}
```

→ mesma complexidade que o quick sort

médio caso =  $n \log n$   
todos

## Matrizes Esparsas

$$\begin{aligned} & (N_{nn} + N_{LCNN})^* \\ & (\text{ptr_da\_lista\_encadeada} * \text{tamanho da lista}) \\ & + (N_{nn} + N_{LCNN})^* \\ & (\text{tamanho da matriz}) \end{aligned}$$

$N = 4$

$$\begin{matrix} 1 & 0 & 0 \\ 2 & 2 & 2 \\ \hline 3 & 2 \end{matrix}$$

14/06

```
typedef struct uno  
{  
    int elem;  
    uno * prox;  
} uno;
```

```
uno * junta (uno * l1, uno * l2)  
{  
    uno * l3;  
    if (l1->elem <= l2->elem)  
    {  
        l3 = l1;  
        l1 = l1->prox;  
    }  
    else  
    {  
        l3 = l2;  
        l2 = l2->prox;  
    }  
    l3->prox = NULL;  
    uno * ult3 = l3;  
    while (l1 != NULL && l2 != NULL)  
    {  
        if (l1->elem <= l2->elem)  
        {  
            ult3->prox = l1;  
            l1 = l1->prox;  
        }  
        else  
        {  
            ult3->prox = l2;  
            l2 = l2->prox;  
        }  
    }  
}
```

~~while (l1 != NULL)  
{  
 ult3->prox = l1;  
 l1 = l1->prox;  
 ult3 = ult3->prox;  
}~~~~while (l2 != NULL)  
{  
 ult3->prox = l2;  
 l2 = l2->prox;  
 ult3 = ult3->prox;  
}~~

```
if (l1 != NULL)  
    ult3->prox = l1;  
if (l2 != NULL)  
    ult3->prox = l2;  
return l3;  
}
```