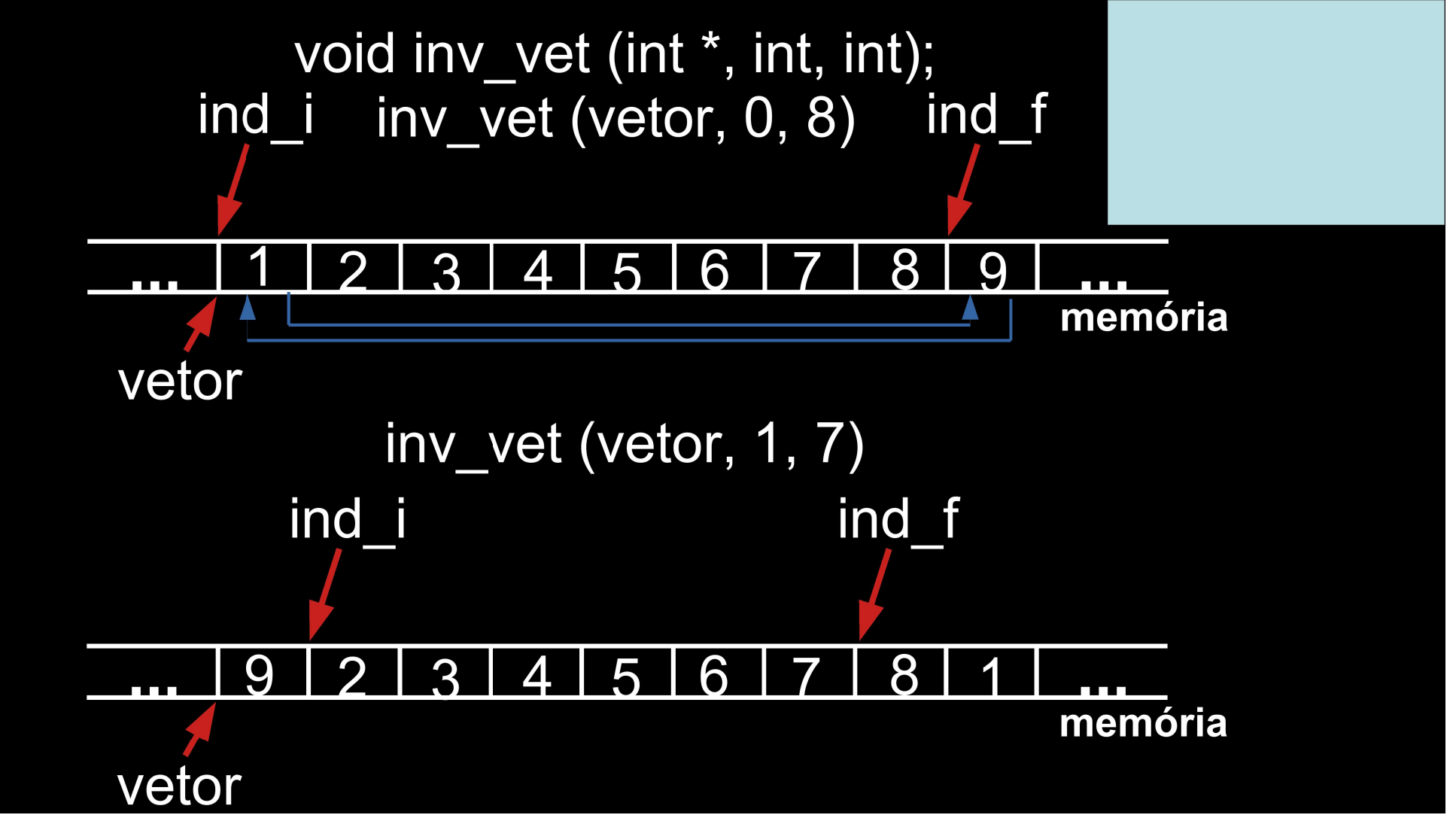
Recursividade

Exercício:

Para uma melhor compreensão do conceito de recursividade construa, na linguagem de programação C, uma função recursiva que receba como parâmetro dados referentes a um vetor com elementos inteiros e inverta a ordem de seus elementos.

UNIVASF


void inv_vet (int *, int, int);
ind_i
inv_vet (vetor, 0, 8)
ind_f
... 1 2 3 4 5 6 7 8 9 ...
memória
vetor
inv_vet (vetor, 1, 7)
ind_i
ind_f
... 9 2 3 4 5 6 7 8 1 ...
memória
vetor

```c
void inv_vet (int *vet, int ind_i, int ind_f)
{
    int aux;
    if (ind_i<ind_f)
    {
        aux = vet[ind_i];
        vet[ind_i] = vet[ind_f];
        vet[ind_f] = aux;
        inv_vet (vet, ind_i+1, ind_f-1);
    }
}
```

Recursividade

Um outro exemplo muito utilizado de problema que possui uma definição recursiva é a geração da série de Fibonacci:

{0, 1, 1, 2, 3, 5, 8, 13, 21, 34, …}

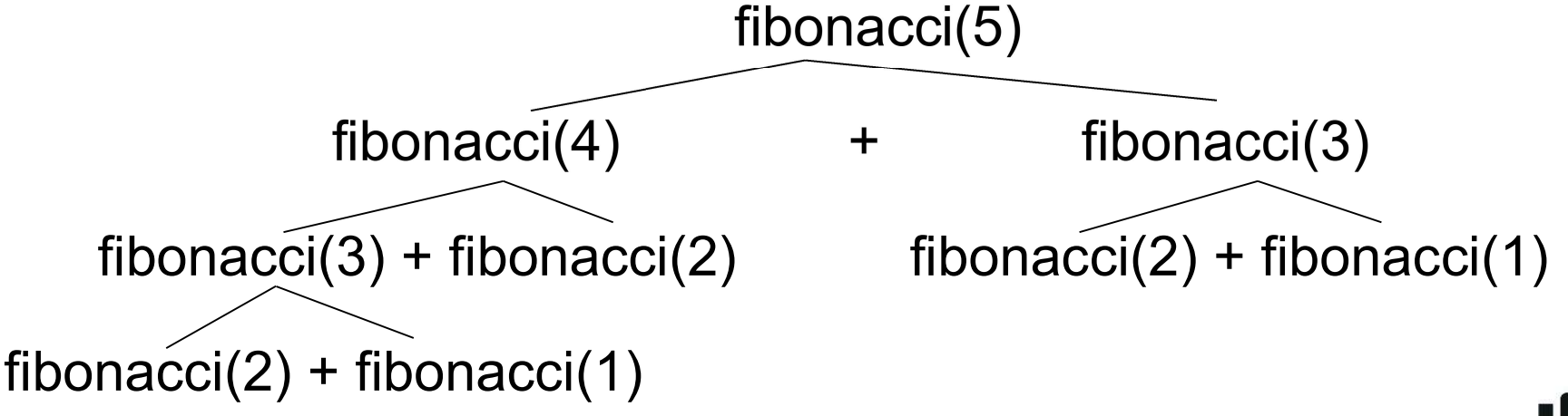
Uma função recursiva que recebe a posição do elemento na série e retorna seu valor é:

```
unsigned int fibonacci(unsigned int i)
{
    if (i==1)
       return 0;
    if (i==2)
       return 1;
    return (fibonacci(i-1) + fibonacci(i-2))
}
```

Recursividade

Fora os problemas mencionados anteriormente, relacionados à troca de contexto na recursão, qual seria outro problema proveniente da recursão evidenciado na função recursiva apresentada para o cálculo do valor de um elemento da série de Fibonacci com base na sua posição?

O cálculo do mesmo valor n vezes.



Recursividade

Como vimos, mesmo problemas que possuem uma definição recursiva podem ser solucionados de forma imperativa. Um exemplo disso é o cálculo imperativo do valor de um elemento da série de Fibonacci com base na sua posição, apresentado abaixo:

```
unsigned int fibonacci(unsigned int i) {
    if (i==1)
        return 0;
    if (i==2)
        return 1;
    else {
        unsigned int a, b;
        for(a=0, b=1; i-2; b+=a,a=b-a,i--);
        return b;
    }
}
```

Recursividade

Assim como a série de Fibonacci existem outras sequências definidas por recorrência, ou seja, onde um valor da sequência é definido em termos de um ou mais valores anteriores, o que é denominado de relação de recorrência.

Exercício:

Estabeleça a relação de recorrência presente na sequência abaixo.

S = { 1, 2, 6, 24, ... }

UNIVASF

Recursividade

A sequência S é definida por recorrência

1. S(1) = 1
2. S(n) = n * S(n-1) para n>=2

UNIVASF

Recursividade

Exercício:

Com base na relação de recorrência estabelecida no exercício anterior, considerando o princípio da modularização, construa um programa que receba uma lista de inteiros positivos, representando posições de elementos na sequência e retorne na saída padrão os respectivos valores da sequência. A lista de posições será finalizada pelo valor zero. Não é necessário validar as entradas.

Exemplo de entrada:	Exemplo de saída:
4	24
2	2
0	

```c
#include <stdio.h>
unsigned int func (unsigned int);
int main() {
    unsigned int num;
    do {
        scanf("%u", &num);
        if (num)
            printf("%u\n",func(num));
    }while(num);
    return 0;
}
unsigned int func (unsigned int n) {
    if (n==1)
        return 1;
    return (n*func(n-1));
}
```

Tipos de Dados Definidos pelo Usuário

Tipos de Dados Definidos pelo Usuário

1. Agregados Heterogêneos (Estruturas)

Um agregado heterogêneo agrupa várias variáveis numa só. Sendo utilizados para agrupar um conjunto de dados não similares formando um novo tipo de dado.

UNIVASF

Tipos de Dados Definidos pelo Usuário

1. Agregados Heterogêneos (continuação)

Para se criar um agregado heterogêneo usa-se o comando **struct**. Sua forma geral é:

```
struct nome_do_tipo_da_estrutura
{
    tipo_1   nome_campo1;
    tipo_2   nome_campo2;
    ...
    tipo_n   nome_campon;
} variáveis_do_tipo_da_estrutura;
```

UNIVASF

Tipos de Dados Definidos pelo Usuário

```
struct nome_do_tipo_da_estrutura
{
    tipo_1    nome_campo1;
    tipo_2    nome_campo2;
    ...
    tipo_n    nome_campon;
};
struct
{
    tipo_1    nome_campo1;
    tipo_2    nome_campo2;
    ...
    tipo_n    nome_campon;
} variáveis_estrutura;
```

Tipos de Dados Definidos pelo Usuário

1. Agregados Heterogêneos (continuação)

Vamos criar um agregado heterogêneo para armazenar um endereço:

```
struct tipo_endereco
{
    char rua [50];
    int numero;
    char bairro [20];
    char cidade [30];
    char sigla_estado [3];
    long int CEP;
};
```

Tipos de Dados Definidos pelo Usuário

1. Agregados Heterogêneos (continuação)

Vamos agora criar uma estrutura chamada ficha_pessoal capaz de armazenar os dados pessoais de uma pessoa:

```
struct ficha_pessoal
{
    char nome [50];
    long int telefone;
    struct  tipo_endereco  endereco;
};
```

UNIVASF

Tipos de Dados Definidos pelo Usuário

1. Agregados Heterogêneos (continuação)

Assim como ocorria nos agregados homogêneos (vetores) , nos agregados heterogêneo também se faz necessário acessar individualmente seus elementos.

Para isso será utilizado o operador “.”.

Veremos sua utilização no exemplo a seguir.

UNIVASF

```c
#include <stdio.h>
#include <string.h>
/*aqui entrariam as declarações das struct's
vistas anteriormente na sequência correta*/
int main ()
{
    struct ficha_pessoal ficha;
    strcpy (ficha.nome, "Fulano de Tal");
    ficha.telefone=4921234;
    strcpy (ficha.endereco.rua,"Rua das Flores");
    ficha.endereco.numero=10;
    strcpy (ficha.endereco.bairro,"Cidade Velha");
    strcpy (ficha.endereco.cidade,"Belo Horizonte");
    strcpy (ficha.endereco.sigla_estado,"MG");
    ficha.endereco.CEP=31340230;
}
```

Tipos de Dados Definidos pelo Usuário

2. Definição de tipo

O comando typedef é utilizado para definir um novo tipo de dado. Ele é utilizado da seguinte forma

```
typedef  tipo  nome_do_tipo;
```

Tipos de Dados Definidos pelo Usuário

Exemplo:

```
typedef struct
{
    int dia;
    int hora;
    int minuto;
} data;
```

```
data d;
```

Tipos de Dados Definidos pelo Usuário

Exercício:

Construa um programa que manipule um vetor com 5 registros de alunos, onde cada registro possui informações referentes ao nome, data de nascimento, número de matricula, CPF e coeficiente de rendimento do aluno. A manipulação do vetor deve ser feita através das seguintes funções: inicializar vetor, imprimir um determinado registro com base no valor do campo CPF e imprimir um determinado registro com base em sua posição no vetor. O programa não pode possuir variáveis globais, deve se utilizar de forma satisfatória das funções mencionadas e deve definir um novo tipo de dado.

UNIVASF

```c
#include <stdio.h>
#include <string.h>
#include <stdlib.h>
#define num_reg 5
typedef struct
{
    char nom[30];
    char dat[9];
    unsigned long int n_mat;
    char cpf[13];
    float cr;
} registro;
void inicializar_vet (registro *);
void imprimir_reg_pos (registro *, unsigned int);
void imprimir_reg_cpf (registro *, char *);
```

```c
int main()
{
    registro v[num_reg];
    int i;
    char cpf[13];
    inicializar_vet(v);
    printf("\nPosicao do registro a ser impresso: ");
    scanf("%d",&i);
    imprimir_reg_pos (v,i);
    printf("\n\n\nCPF do registro a ser impresso: ");
    scanf("%s",cpf);
    imprimir_reg_cpf (v,cpf);
    return 0;
}
```

```c
void inicializar_vet (registro *v)
{
    int i;
    for (i=0; i<num_reg; i++) {
        printf("\nEntre com as informacoes do %d° registro.",i+1);
        printf("\nNome: ");
        scanf("%29[^\n]", v[i].nom);
        printf("\nData (dd/mm/aa): ");
        scanf("%8[^\n]", v[i].dat);
        printf("\nNumero de matricula: ");
        scanf("%ld",&v[i].n_mat);
        printf("\nCPF: ");
        scanf("%12[^\n]", v[i].cpf);
        printf("\nCoeficinte de rendimento: ");
        scanf("%f",&v[i].cr);
    }
}
```

```c
void imprimir_reg_pos (registro *v, unsigned int pos)
{
    if (pos>num_reg || !pos)
    {
        printf("\nErro!\nPosicao invalida para impressao!\n");
        exit(1);
    }
    else
    {
        printf("\n\n\n%d° registro.\n",pos);
        printf("\nNome: \t\t\t\t%s",v[pos-1].nom);
        printf("\nData: \t\t\t\t%s",v[pos-1].dat);
        printf("\nNumero de matricula: \t\t%ld",v[pos-1].n_mat);
        printf("\nCPF: \t\t\t\t%s",v[pos-1].cpf);
        printf("\nCoeficinte de rendimento: \t%.3f",v[pos-1].cr);
    }
}
```

```c
void imprimir_reg_cpf (registro *v, char *chave)
{
    int i;
    for (i=0; i<num_reg; i++)
        if(!strcmp(v[i].cpf,chave))
        {
            printf("\n\n\n%d° registro.\n",i+1);
            printf("\nNome: \t\t\t\t%s",v[i].nom);
            printf("\nData: \t\t\t\t%s",v[i].dat);
            printf("\nNumero de matricula: \t\t%ld",v[i].n_mat);
            printf("\nCPF: \t\t\t\t%s",v[i].cpf);
            printf("\nCoeficinte de rendimento: \t%.3f",v[i].cr);
            return;
        }
    printf("\n\nNao existe registro com o CPF especificado.\n");
}
```