

**Universidade Federal de Viçosa**  
**Campus de Florestal**

# **Algoritmos e Estruturas de Dados I (CCF 211)**

Introdução (Cap01 – Seções 1.1 e 1.2 - Ziviani)

*Profa. Thais R. M. Braga Silva*  
*<thais.braga@ufv.br>*



# Revisão Básica - Linguagem C

- **Algoritmo básico em linguagem C**
- **Variáveis, Tipos primitivos, Operadores, E/S**
- **Comandos de decisão**
- **Comandos de repetição**
- **Vetores, Matrizes e Strings**
- **Funções**
- **Ponteiros**

# Algoritmo Básico Linguagem C

```
#include <stdio.h>

int main(void) {
    printf("Hello, World!\n");
    return 0;
}
```

Hello, World!

# Comandos - Linguagem C

- **Definição de variáveis**
  - `<tipo> <nome>;`
- **Tipos primitivos**
  - `int, char, float, double, void`
  - Valores lógicos: `0` → falso / diferente de `0` → verdadeiro
  - modificadores `signed, unsigned` (`int` e `char`)
  - modificadores `long` (`int` e `double`), `short` (`int`)

# Comandos - Linguagem C

- **Operadores**

- Aritméticos (+, -, \*, /, %)
- Relacionais (>, >=, <, <=, ==, !=)
- Lógicos (&&, ||, !)
- Lógicos bit a bit (&, |, ^, ~, >>, <<)

- **Entrada e saída (stdio.h)**

- printf( "*string com marcadores*", lista\_de\_valores)
- scanf( "*lista de marcadores*", lista\_de\_variaveis)
  - Usar & antes de cada variável
  - As variáveis já tem que ter sido previamente declaradas

# Comandos - Linguagem C

```
1      #include <stdio.h>
2      #include <conio.h>
3
4      void main()
5      {
6          int num1, num2, num;
7          printf("Enter 1st number\n");
8          scanf("%d", &num1);
9          printf("Enter 2nd number\n");
10         scanf("%d", &num2);
11         num=num1+num2;
12         printf("Sum of 2 number is %d", num);
13         getch();
14     }
15
```

# Tipos Primitivos

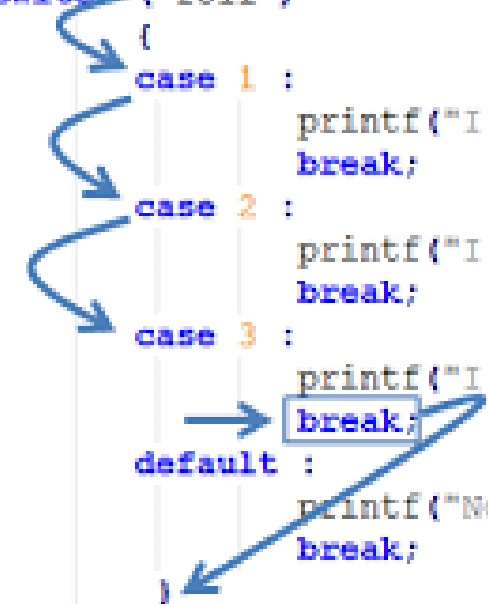
Tipo	Num de bits	Formato para leitura com scanf	Intervalo	
			Inicio	Fim
char	8	%c	-128	127
unsigned char	8	%c	0	255
signed char	8	%c	-128	127
int	16	%i	-32.768	32.767
unsigned int	16	%u	0	65.535
signed int	16	%i	-32.768	32.767
short int	16	%hi	-32.768	32.767
unsigned short int	16	%hu	0	65.535
signed short int	16	%hi	-32.768	32.767
long int	32	%li	-2.147.483.648	2.147.483.647
signed long int	32	%li	-2.147.483.648	2.147.483.647
unsigned long int	32	%lu	0	4.294.967.295
float	32	%f	3,4E-38	3.4E+38
double	64	%lf	1,7E-308	1,7E+308
long double	80	%Lf	3,4E-4932	3,4E+4932

# Controle de Fluxo

- **Comandos de decisão**
  - if else / switch

```
1 #include <stdio.h>
2
3 main(){
4
5     int idade;
6     printf ("Informe a sua idade: ");
7     scanf ("%d",&idade);
8
9     if (idade <= 12){
10         printf ("\n->Voce ainda e uma crianca!\n");
11     }else if (idade <= 20){
12         printf ("\nVoce ainda e um adolescente!\n");
13     }else if (idade <= 59){
14         printf ("\nVoce e um adulto!\n");
15     }else{
16         printf ("\nTerceira idade!\n");
17     }
18     system ("pause");
19 }
```

```
void main()
{
    int roll = 3 ;
    switch ( roll )
    {
        case 1 :
            printf("I am Pankaj");
            break;
        case 2 :
            printf("I am Nikhil");
            break;
        case 3 :
            printf("I am John");
            break;
        default :
            printf("No student found");
            break;
    }
}
```





# Controle de Fluxo

- **Comandos de repetição**
  - for / while / do while

Exemplo do laço: for

```
1  #include<stdio.h>
2
3  main(){
4      int contador = 0;
5      for (contador=0; contador <=10; contador++) {
6          printf ("%d \n", contador);
7      }
8  }
```

Exemplo do laço: while

```
1  #include<stdio.h>
2
3  main(){
4
5      int contador = 0;
6
7      while(contador <= 10) {
8          printf ("%d \n", contador);
9          contador++;
10     }
11 }
```

```
1  #include <stdio.h>
2  int main() {
3      int n;
4      n=5;
5      do {
6          n++;
7      }
8      while ( n<5 );
9      printf(" %d ", n );
10     return 0;
11 }
```

# Tipos Estruturados

- **Vetores**

- `<tipo> <nome>[<tamanho>];`

- **Matrizes**

- `<tipo> <nome>[<tamanho>]...[<tamanho>];`

- **Strings**

- `char <nome>[<tamanho>];`

- `gets(), strcpy(), strcat(), strcmp(), strlen()`  
`(<string.h>)`

# Tipos Estruturados

```
#include <stdio.h>

int main()
{
    int i;
    int a[10];
    printf("Enter student's scores: \n");
    for(i = 0; i < 10; i++) {
        scanf("%d", &a[i]);
    }
    printf("Your student's scores are: \n\n");
    for(i = 0; i < 10; i++) {
        printf("%d\n", a[i]);
    }
    return 0;
}
```

# Manipulação de Arquivos

- Cabeçalho <stdio.h>
- FILE \*<var\_arq>; // FILE: estrutura que descreve arq.
- <var\_arq> = fopen("nome-arquivo", "r/w/rb/wb");
- fclose(<var\_arq>);
- NULL / EOF
- Leitura/escrita de caracteres: fputc()/fgetc()
- Leitura/escrita de strings: fputs()/fgets()
- Leitura/escrita de tipos: fscanf()/fprintf() (especif. formato)
- Leitura/escrita de blocos de bytes: fwrite()/fread()

# Manipulação de Arquivos

```
93  void txt_ler_numero()  
94  {  
95      FILE *arq;  
96      char ch = ' '  
97      system("cls");  
98  
99      arq = fopen("arquivo.txt", "r");  
100  
101      printf("\n\n Lendo o arquivo caracter a caracter: \n\n");  
102      do  
103      {  
104          ch = fgetc(arq);  
105          printf("%c - %d\n", ch, ch);  
106      } while (!feof(arq));  
107  
108      fclose(arq);  
109      printf("\n\n");  
110      system("pause");  
111  }
```

# Subprogramas

- **Configuração geral:**

- `<tipo_retorno> <nome> (<parâmetros>){ ...  
return <valor>;}`
- void: função sem retorno ou sem parâmetros

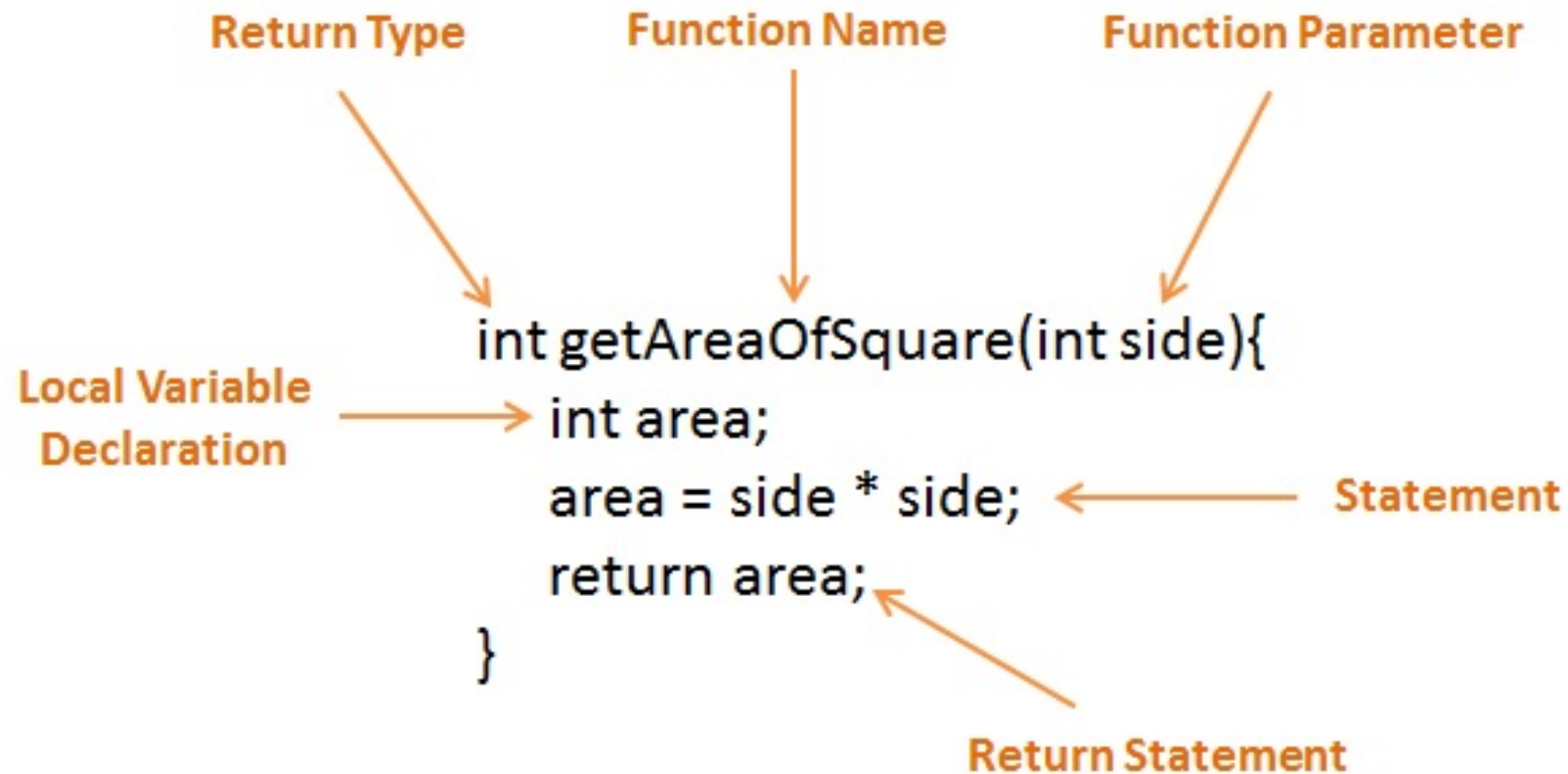
- Devem ser declaradas (totalmente ou protótipo) antes da função principal (*main*)
- Separam as funcionalidades do programa em blocos. O fluxo de execução desvia para o bloco da função mediante sua chamada/invocação
- Arquivos cabeçalhos (extensão .h): contém definições (variáveis, macros, tipos, protótipos..)

# Subprogramas

- **Passagem de parâmetros**
  - Por valor: parâmetros reais são copiados para os parâmetros formais – não há modificação
  - Por referência: endereço de memória deve ser passado aos parâmetros formais, que são ponteiros e não variáveis – pode haver modificação
- **Recursividade: função que possui chamada a ela mesma – cuidado para utilizar:**
  - Critério de parada e consumo de memória

# Subprogramas

## Function Definition



techcrashcourse.com

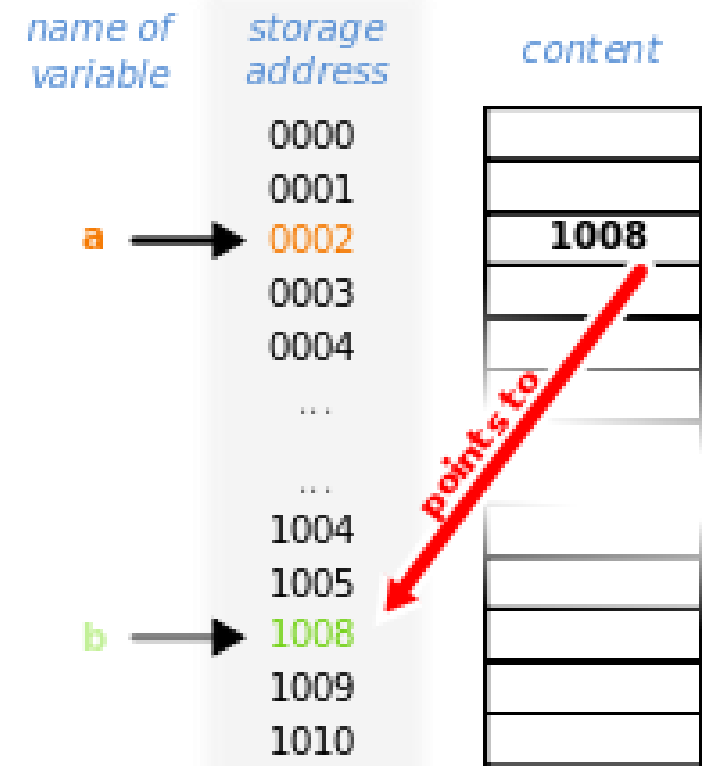
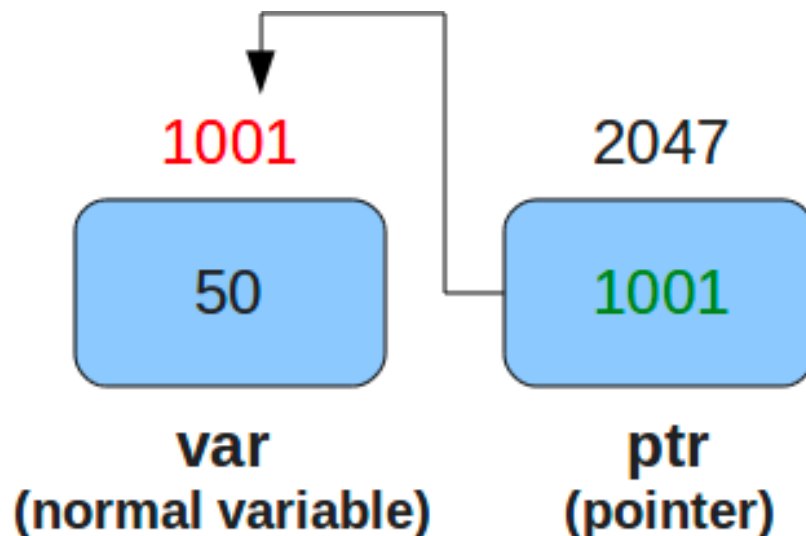


# Subprogramas

```
1  /* C basic structure explained
2     Author: fresh2refresh.com    // Documentation section
3     Date    : 01/01/2012
4  */
5  #include <stdio.h>              // Link section
6  int total;                     // Global declaration section
7  int sum (int, int);            // Function declaration section
8  int main ()                    // Main function
9  {
10     printf ("This is main function \n");
11     total = sum (1, 1);
12     printf ("Total = %d \n", total);
13     return 0;
14 }
15 int sum (int a, int b)
16 {                               // User defined function
17     return a + b;              // definition section
18 }
```

# Ponteiros - Linguagem C

- Variável especial que armazena endereços de memória
- Declaração: `<tipo> *nome;`



# Ponteiros - Linguagem C

- Operadores importantes: & e \*
- & = retorna endereço de memória corrente
- \* utilizado em 3 situações:
  - Para declaração da variável ponteiro (Ex: `int *b;`)
  - \* lado direito da atribuição: acessa conteúdo referenciado pelo ponteiro (dereferenciamento) (Ex: `int a = *b;`)
  - \* lado esquerdo da atribuição: retorna endereço armazenado pelo ponteiro para armazenamento do valor passado do lado direito (Ex: `*b = 10;`)

# Ponteiros - Linguagem C

- Exemplo:
  - `int a = 10;`
  - `int *b = NULL;` *// Sempre iniciar ponteiro com NULL*
  - `b = &a;` *// “b aponta para a”*
  - `printf(“%d\n”,*b);`
- São muito utilizados para alocação dinâmica e passagem de parâmetros por referência

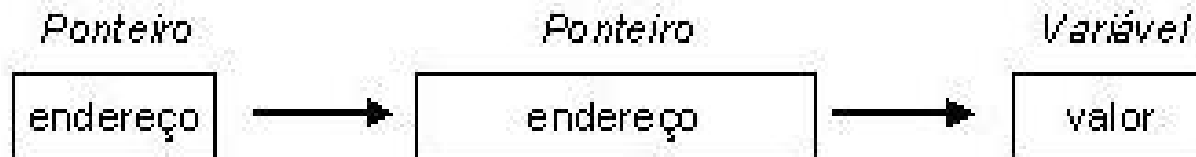
# Ponteiros - Linguagem C

- Ponteiros para ponteiros: ponteiro aponta para uma posição de memória que aponta para outra
  - `<tipo> **<nome>; // uso cuidadoso e restrito`
  - Exemplo: `int **p = NULL;`

## Indireção simples



## Indireção Múltipla



# Ponteiros - Linguagem C

```
*main.c x
1  #include<stdio.h>
2  int main()
3  {
4      int a [] = {10, 20, 30, 40};
5      int *p;
6      p = &a[0];
7      int i;
8      printf("Output via Pointer:\n");
9      printf("Address\t Value");
10     for(i=0; i<4; i++)
11     {
12         printf("\n%u", p);
13         printf("    %d", *p);
14         p++;
15     }
16     return 0;
17 }
```

# Ponteiros - Linguagem C

```
#include <stdio.h>

int main ()

{

    int num,valor;

    int *p;

    num=55;

    p=&num;    /* Pega o endereco de num */

    valor=*p;  /* Valor e igualado a num de uma maneira indireta */

    printf ("\n\n%d\n",valor);

    printf ("Endereco para onde o ponteiro aponta: %p\n",p);

    printf ("Valor da variavel apontada: %d\n",*p);

    return(0);

}
```

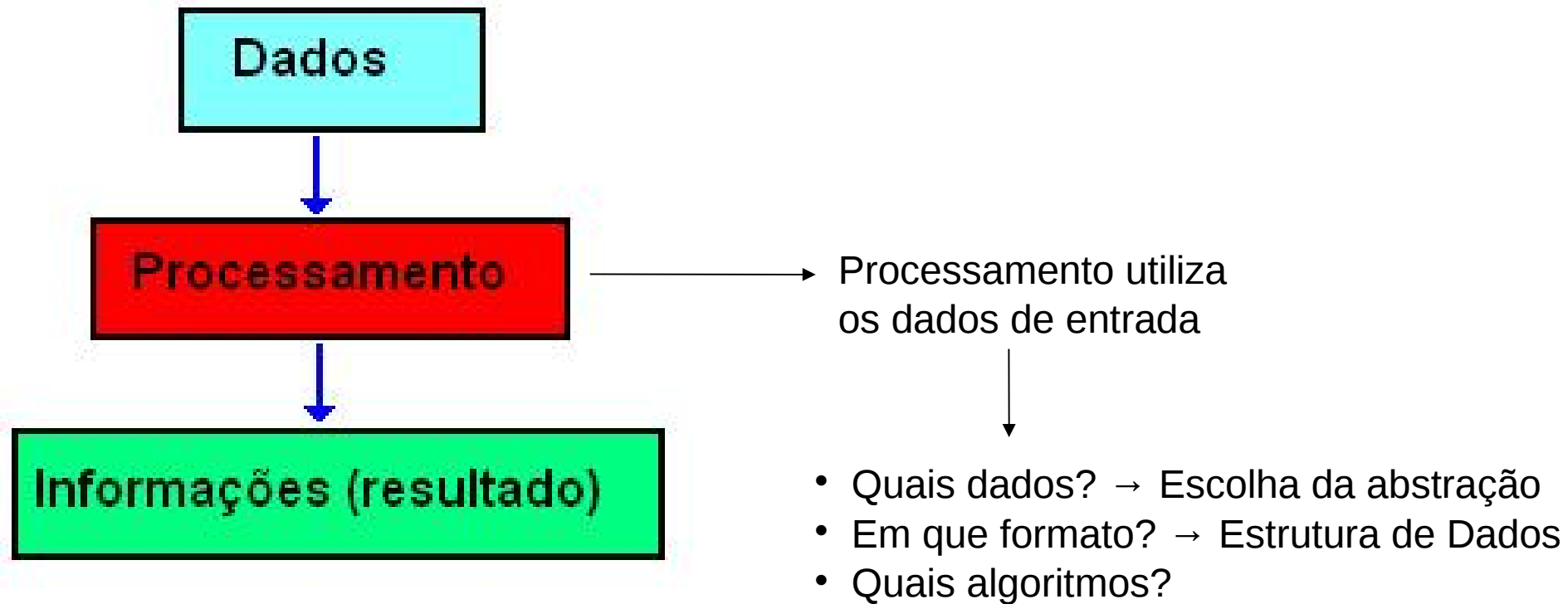
# Exercícios - Linguagem C

- Faça um algoritmo que contenha uma função para calcular as raízes reais de uma equação de segundo grau. Passe os valores das constantes que multiplicam cada termo da equação como parâmetros.
- Faça um algoritmo que contenha uma função que recebe duas variáveis inteiras e zera o valor das mesmas. As variáveis devem permanecer zeradas após o retorno da função.
- Faça um algoritmo que implemente a função `StrEnd(char *s, char *t)`, que retorna 1 (um) se a cadeia de caracteres `t` ocorrer no final da cadeia `s`, e 0 (zero) caso contrário.



# Algoritmos e Estruturas de Dados

Resolução computacional de problemas funciona da seguinte forma:



# Algoritmos e Estruturas de Dados

- **A resolução de um problema envolve escolher uma abstração da realidade**
  - Conjunto de dados que representam a situação real
  - Em seguida, escolher forma de representar esses dados  
= escolher a **Estrutura de Dados (ED)** a ser utilizada
- **Algoritmo**
  - Sequência finita de ações executadas para a resolução de um determinado problema
  - Exemplos: receita culinária, instruções de montagem, forma de uso de medicamentos
  - Algoritmos operam sobre Estruturas de Dados!

# Algoritmos e Estruturas de Dados

- **Exemplo: resolver o problema de alocação de alunos em disciplinas ofertadas**
- **Algumas abstrações necessárias**
  - Aluno – Quais dados podem caracterizá-lo?
    - Matrícula, nome, CRA, lista de disciplinas cursadas, etc..
  - Disciplina – Quais dados podem caracterizá-la?
    - Código, nome, lista de pré-requisitos, etc..
- **Como representar (estruturar) os dados de cada abstração?**
  - Escolha da estrutura de dados

# Representação dos Dados

- **Dados podem estar representados (estruturados) de diferentes maneiras**
  - Existem diferentes possíveis EDs para uma mesma abstração
- **A escolha da representação normalmente é determinada pelas operações que serão utilizadas sobre eles**
- **Exemplo: números inteiros**
  - Representação por palitinhos: operação simples
  - Representação decimal: operação complexa

# Algoritmos e Estruturas de Dados

- **Programar é, basicamente, estruturar dados e construir algoritmos**
  - Escolher Estruturas de Dados e as implementações para as operações necessárias
- **Algoritmos e Estruturas de Dados (EDs) estão intimamente ligados**
  - Selecionar melhor ED dadas as operações (algoritmos) a serem executadas sobre ela
  - Dada uma ED, é possível ou eficiente executar sobre ela determinado algoritmo?

# Programas

- **Programa é uma formulação concreta de um algoritmo abstrato baseado em representações de dados específicas**
- **Representam algoritmos capazes de serem seguidos por computadores**
- **São feitos em alguma linguagem que pode ser entendida e seguida pelo computador**
  - Linguagem de máquina
  - Linguagem de alto nível (uso de compilador)

# Tipos de Dados

- **Tipo de dados define um conjunto de valores e seu respectivo conjunto de operações**
- **Em linguagens são associados à variáveis, constantes, expressões e funções**
- **Idealmente, os programas deveriam ter tipos correspondentes às abstrações de dados necessárias para cada implementação**
- **LPs oferecem tipos simples ou estruturados**
  - Simples: oferecidos pela linguagem, com valores indivisíveis
  - Estruturados: coleção de dados de tipos simples ou estruturados

# Tipo Abstrato de Dado (TAD)

- **Representação completa de uma abstração, criando um novo tipo de dado**
- **Modelo matemático acompanhado das operações definidas sobre ele**
  - Modelo matemático = estrutura de dados
  - Usa tipos de dados e operadores suportados pela linguagem de programação considerada
- **Agrupar uma ED juntamente com as operações que podem ser realizadas sobre os dados que ela representa**



# Tipos Abstratos de Dados (TADs)

- **São generalizações de tipos primitivos de dados**
  - Usados para encapsular tipos de dados, assim como procedimentos encapsulam partes de um algoritmo
  - A definição do tipo e todas as operações definidas sobre ele podem ser localizadas em uma única seção do programa
  - Organiza, facilita e torna mais eficientes a programação de sistemas
  - Exemplo: TADs de um sistema escolar
    - TAD Aluno
    - TAD funcionário
    - TAD SalaDeAula
    - TAD Disciplina

# Tipos Abstratos de Dados (TADs)

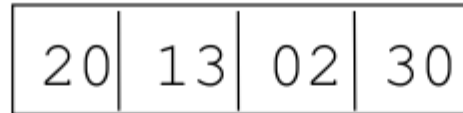
- **O TAD deve encapsular a ED. Código usuário do TAD só deve ter acesso a algumas operações disponibilizadas sobre os dados**
- **Código usuário do TAD x código do TAD**
  - Usuário “enxerga” apenas a interface, não a implementação
- **Dessa forma, usuário pode abstrair da implementação específica**
- **Qualquer modificação nessa implementação fica restrita ao TAD**

# Exemplo: Lista de Números Inteiros

- **Operações**
  - Faz lista vazia
  - Insere número no começo da lista
  - Remove de uma posição  $i$
- **Opções de implementação**
  - Arranjos/Vetores
  - Listas encadeadas

# Exemplo: Lista de Números Inteiros

**Implementação por Vetores:**



```
void Insere(int x, Lista L) {  
    for(i=0;...) {...}  
    L[0] = x;  
}
```

**Implementação por Listas Encadeadas**



```
void Insere(int x, Lista L) {  
    p = CriaNovaCelula(x);  
    L^.primeiro = p;  
    ...  
}
```

**Programa usuário do TAD:**

```
int main() {  
    Lista L;  
    int x;  
  
    x = 20;  
    FazListaVazia(L);  
    Insere(x, L);  
    ...  
}
```

# Implementação de TADs

- **Em linguagens orientadas a objetos (C++, Java) a implementação é feita por meio de classes**
- **Em linguagens estruturadas (C, Pascal) a implementação é feita através da definição de tipos, junto com a definição de funções**
- **Nesta disciplina vamos utilizar os conceitos de linguagens estruturadas**
  - Linguagem C (typedef e structs)
  - Orientação por objetos (classes, etc...) vai ser vista em disciplinas posteriores

# Estruturas

- **Uma estrutura é uma coleção de uma ou mais variáveis, possivelmente de tipos diferentes, colocadas juntas sob um único nome para manipulação conveniente**
- **Exemplo: para representar um aluno são necessárias as informações, nome, matrícula, conceito**
- **Ao invés de criar três variáveis, é possível criar uma única contendo três campos**
- **Em linguagem C: struct**

# Estruturas (Struct) em C

```
struct Aluno {  
    char nome[100];  
    int matricula;  
    char conceito;  
};
```

```
main() {  
    struct Aluno al, aux;  
  
    al.nome = "Luiz"  
    al.matricula = 200712;  
    al.conceito = 'A';  
    aux = al;  
    printf("%s", aux.nome);  
}
```

**al:**

Luiz	
200712	A

**aux:**

Luiz	
200712	A

# Declaração de Tipos (C)

- Para simplificar, uma estrutura ou outros tipos de dados, podem ser definidos como um novo tipo (novo nome)
- Uso da construção typedef

```
typedef struct {  
    char nome[100];  
    int matricula;  
    char conceito;  
} TipoAluno;  
  
typedef int[10] Vetor;
```

```
int main() {  
    TipoAluno al;  
    Vetor v;  
  
    ...  
}
```



# Estruturas e Ponteiros (C)

- **Ponteiros podem apontar para estruturas**
- **Operador “->” facilita acesso ao conteúdo da estrutura referenciada pelo ponteiro**

```
TipoAluno aluno1;  
TipoAluno *p_al = NULL;  
p_al = &aluno1;  
(*p_al).conceito = 'A';  
p_al -> matricula = 1234;
```

# TADs - linguagem C

- **Para implementar um TAD, usa-se a definição de tipos juntamente com a implementação de funções que agem sobre aquele tipo**
  - Em C: typedef struct para criar o novo tipo (TAD)
  - Estrutura de dados escolhida fica dentro da estrutura
- **Como boa regra de programação, evita-se acessar o dado diretamente, fazendo o acesso apenas através das funções**
  - Note que não há uma forma de proibir o acesso direto aos campos da estrutura na linguagem C

# TADs - Linguagem C

- **Uma boa técnica de programação é implementar os TADs em arquivos separados do programa principal**
- **Para isso, em geral, separa-se a declaração e a implementação do TAD em dois arquivos:**
  - NomeDoTAD.h: contendo a declaração
  - NomeDoTAD.c: contendo a implementação
- **O programa, ou outros TADs, que utilizam o seu TAD devem realizar um *#include* do arquivo .h**

# Exemplo

- **Implemente um TAD ContaBancaria, com os campos número e saldo, onde os clientes podem fazer as seguintes operações:**
  - Iniciar uma conta com número e saldo inicial
  - Depositar um valor
  - Sacar um valor
  - Imprimir o saldo
- **Faça um pequeno programa para testar o seu TAD**

# Contabancaria.h

```
// definição do tipo
typedef struct {
    int numero;
    double saldo;
} ContaBancaria;

// cabeçalho das funções
void Inicializa(ContaBancaria* conta, int numero, double saldo);
void Deposito (ContaBancaria* conta, double valor);
void Saque (ContaBancaria* conta, double valor);
void Imprime (ContaBancaria conta);
```

# Contabancaria.c

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

void Inicializa(ContaBancaria* conta, int numero, double saldo)
{
    (*conta).numero = numero;
    (*conta).saldo = saldo;
}

void Deposito (ContaBancaria* conta, double valor)
{
    (*conta).saldo += valor;
}

void Saque (ContaBancaria* conta, double valor)
{
    (*conta).saldo -= valor;
}

void Imprime (ContaBancaria conta)
{
    printf("Numero: %d\n", conta.numero);
    printf("Saldo: %f\n", conta.saldo);
}
```

# Main.c

```
#include <stdio.h>
#include <stdlib.h>
#include "Contabancaria.h"

int main (int argc, char **argv)
{
    ContaBancaria conta1;
    Inicializa(&conta1, 918556, 300.00);
    printf("\nAntes da movimentacao:\n ");
    Imprime(conta1);
    Deposito(&conta1, 50.00);
    Saque(&conta1, 70.00);
    printf("\nDepois da movimentacao:\n ");
    Imprime (conta1);
}
```

# Exercício

- **Implemente um TAD Número Complexo:**
  - Cada número possui os campos real e imaginário
  - Implemente as operações:
    - Atribui: atribui valores para os campos
    - Imprime: imprime o número na forma “ $R+Ci$ ”
    - Cópia: copia o valor de um número para outro
    - Soma: soma dois números complexos
    - EhReal: testa se um número é real
- **Faça uma pequena aplicação para testar o seu TAD**



# Exercício

- **Implemente os seguintes TADs propostos abaixo. Você deverá decidir como estruturar os dados, bem como quais operações estarão definidas. Faça sempre uma pequena aplicação para teste.**
  - TAD Aluno UFV
  - TAD Ponto (no plano cartesiano)
  - TAD Figura geométrica