

SSC0101 - ICC1 – Teórica

---

Introdução à Ciência da Computação I

**Estruturas Dinâmicas - Ponteiros**  
**Parte IV**

Prof. Vanderlei Bonato: [vbonato@icmc.usp.br](mailto:vbonato@icmc.usp.br)

Prof. Claudio Fabiano Motta Toledo: [claudio@icmc.usp.br](mailto:claudio@icmc.usp.br)

---

# Sumário

---

- Typedef
- Macros
- Exemplos
- Vetor de ponteiros para funções
- const & volatile

# Typedef

---

- Utilizado para criar novos nomes de tipos de dado.

```
#define N 3
typedef double scalar;
typedef scalar vector[N];
typedef scalar matrix[N][N];
```

- Também poderíamos ter utilizado:

```
typedef vector matrix[N];
```

- O uso de typedef permite ao programador pensar em termos da aplicação desenvolvida.

# Typedef

---

```
void soma(vector x, vector y, vector z)
{
    int i;
    for (i=0; i<N; ++i)
        x[i] = y[i] + z[i];
}

scalar produto_scalar(vector x, vector y, vector z)
{
    int i;
    scalar sum = 0.0;
    for (i=0; i<N; ++i)
        sum += x[i] * y[i];
    return sum;
}
```

# Typedef

---

```
void produto_matriz(matrix a, matrix b, matrix c)
{
    int i, j, k;
    for(i=0; i<N; ++i){
        for(j=0; j<N; ++j){
            a[i][j]=0.0;
            for(k=0; k<N; ++k){
                a[i][j] += b[i][k]*c[k][j];
            }
        }
    }
}
```


# Macro

---

- Substitui um identificador da macro por um texto substitutivo.

```
#define max(A,B) ((A)>(B)) ? (A):(B))
```

- Apesar de parecer uma função, o uso de max torna-se um código. Cada ocorrência de max é trocada pelo argumento real correspondente.

```
x = max(p+q, r+s);  x=((p+q)>(r+s)?(p+q):(r+s));
```

- Cuidado com as armadilhas:

```
#include <stdio.h>
#include <stdlib.h>
#include <conio.h>
#define max(A,B) ((A)>(B) ? (A):(B))
#define quad(X) X*X
int main()
{
    int p=1,q=2,r=3,s=4,
        i=5, j=10, z=3;
    printf("\nmax=%d", max(p+q,r+s));
    printf("\nmax=%d", max(i++, j++));
    printf("\ni=%d j=%d",i,j);
    printf("\nquad=%d", quad(z+1));
    return 0;
}
```

```
max=7
max=11
i=6 j=12
quad=7
```

# Macro - Assert

---

- A macro `assert()` é utilizada para assegurar que o valor de uma expressão é o valor esperado.
- O arquivo de cabeçalho `assert.h` contém essa macro.
- Se o valor esperado não for retornado, o sistema irá imprimir uma mensagem e o programa será encerrado.



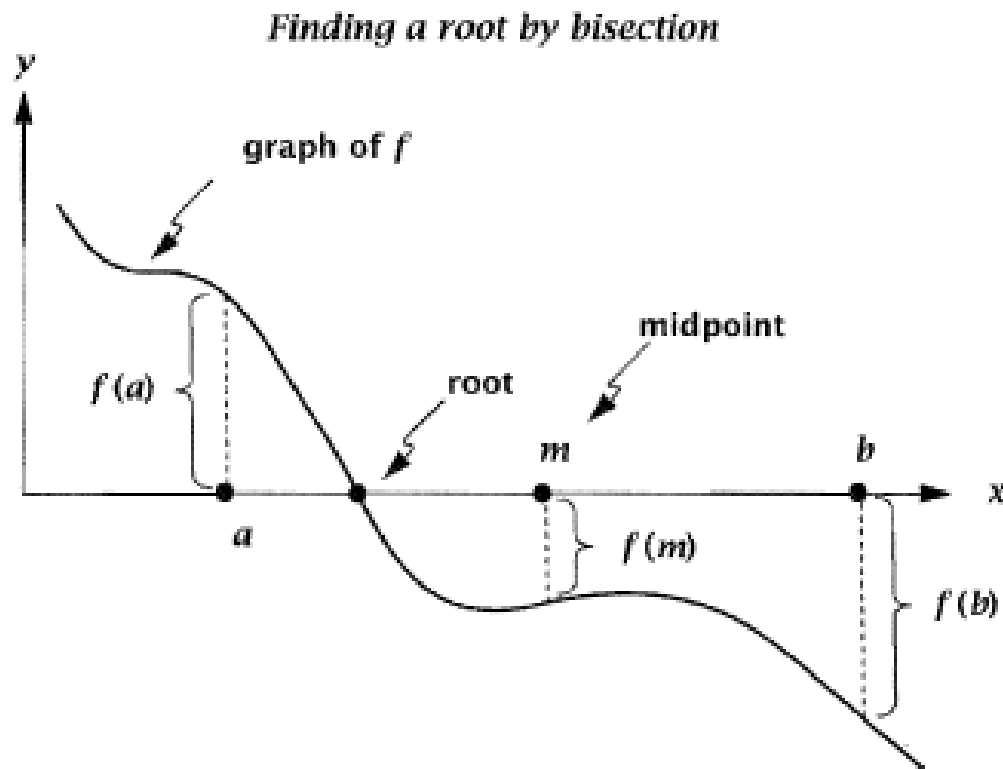
# Macro - Assert

---

```
#include <assert.h>
#include <stdio.h>
int f(int a, int b);
int g(int c);
int main(void)
{
    int a, b, c;
    ....
    scanf("%d%d", &a, &b);
    c=f(a,b);
    assert(c>0); \\se condição for falsa sai do programa e mostra mensagem
                \\tipicamente com nome do arq. e num. da linha
    b=g(c);
    ...
}
```

# Exemplo 1

- Encontrar a raiz do polinômio  $x^5 - 7x - 3.0$



## main.c

```
#include "find_root.h"
int cnt=0;
const dbl eps=1e-13;

int main()
{
    dbl a = -10.0;
    dbl b = 10.0;
    dbl root;

    assert((f(a)*f(b)<=0.0));
    root = bisection(f,a,b);
    printf("%s%d\n%s%.15f\n%s%.15f\n",
        "Num. de chamadas de fct: ",cnt,
        "Raiz aproximada: ", root,
        " Valor da funcao: ", f(root));
    return 0;
}
```

## bisection.c

```
#include "find_root.h"

dbl bisection(dbl f(dbl x), dbl a, dbl b)
{
    dbl m = (a+b)/2.0;

    ++cnt;
    if(f(m)==0.0 || b-a < eps)
        return m;
    else if (f(a)*f(m)<0.0)
        return bisection (f,a,m);
    else
        return bisection(f,m,b);
}
```

Num. de chamadas fct: 49  
Raiz aproximada:  
1.719628091484431  
Valor da função: -  
0.00000000000000979

## find\_root.h

```
#include <assert.h>
#include <stdio.h>
typedef double dbl;
extern int cnt;
extern const dbl eps;
dbl bisection(dbl f(dbl x), dbl a, dbl b);
dbl f(dbl x);
```

## fct.c

```
#include "find_root.h"

dbl f(dbl x)
{
    return (x*x*x*x*x - 7.0*x - 3.0);
}
```

## Exemplo 2

- Solucionando a equação de Kepler

$$x - e \sin(x) - m \text{ com } m=2.2, e=0.5.$$

```
#include <assert.h>    kepler.h
#include <math.h>
#include <stdio.h>
typedef double dbl;
extern int    cnt;
extern const dbl eps;
extern const dbl e;
extern const dbl m;
dbl  bisection(dbl f(dbl x), dbl a, dbl b);
dbl  kepler(dbl x);
```

kepler.c

```
#include "kepler.h"

dbl kepler(dbl x)
{
    return (x- e*sin(x)-m);
}
```

main.c

```
#include "kepler.h"
int cnt=0;
const dbl eps=1e-15;
const dbl e=0.5;
const dbl m=2.2;

int main()
{
    dbl a = -100.0;
    dbl b = 100.0;
    dbl root;

    assert((kepler(a)*kepler(b)<=0.0));
    root = bisection(kepler,a,b);
    printf("%s%d\n%s%.15f\n%s%.15f\n",
        "Num. de chamadas de fct: ",cnt,
        "Raiz aproximada: ", root,
        " Valor da funcao: ", f(root));
    return 0;
}
```

# Vetor de ponteiros para funções

---

- Suponha que o método da biseccção fosse aplicado a um conjunto de funções.
- Um vetor de ponteiros para funções poderia ser utilizado.
- O exemplo a seguir apresenta o uso de ponteiros para funções.

```

#include <assert.h>
#include <math.h>
#include <stdio.h>

#define N 4;

typedef double dbl;

typedef dbl (*pfdd)(dbl);

extern int cnt;
extern const dbl eps;

dbl bisection(pfdd f, dbl a, dbl b);
dbl f1(dbl x);
dbl f2(dbl x);
dbl f3(dbl x);

```

find\_roots.h

bisection.c

```

#include "find_roots.h"

dbl bisection(pfdd f, dbl a, dbl b)
{
    dbl m = (a+b)/2.0;

    ++cnt;
    if(f(m)==0.0 || b-a < eps)
        return m;
    else if (f(a)*f(m)<0.0)
        return bisection (f,a,m);
    else
        return bisection(f,m,b);
}

```



# Vetor de ponteiros para funções

---

- `typedef dbl (*pfdd) (dbl);`
  - Cria um tipo `pfdd` que representa ponteiro para função com um único argumento do tipo `double` e que retorna um `double`.
- `dbl bisection (pfdd f, dbl a, dbl b);`
  - Utiliza o tipo `pfdd`
  - Os protótipos abaixo são equivalentes:
    - `dbl bisection (dbl f(dbl), dbl a, dbl b);`
    - `dbl bisection (dbl f(dbl x), dbl a, dbl b);`

```

#include "find_roots.h"
int cnt=0;
const dbl eps=1e-13;
int main(void)
{
    int begin_cnt, nfct_calls, i;
    dbl a = -100.0;
    dbl b = 100.0;
    dbl root, val;
    dbl f [ N ] = {NULL, f1, f2, f3};

    for(i=1; i<N; ++i){
        assert(f [ i ] (a) * f [ i ] (b) <=0.0);
        begin_cnt = cnt;
        root = bisection(f [ i ],a,b);
        nfct_calls = cnt - begin_cnt;
        val= f [ i ] (root);
        printf("%s%d\n%s%.15f\n%s%.15f\n",
            "Para f[" , i , "](x) a raiz aproximada x0 é: ", root,
            " Valor de f [ ",i,"](x0)=" , val,
            " Num, de chamadas do método = ", nfct_calls);
    }
    return 0;
}

```

main.c

# Vetor de ponteiros para funções

---

- `pfdd f[N] = {NULL, f1, f2, f3};`
  - Declara um vetor de n elementos do tipo `pfdd`.
  - Teremos `f [0] = NULL`.
  - As declarações abaixo são equivalentes
    - `pfdd f[N] = {NULL, &f1, &f2, &f3};`
    - `dbl (*f[N])(dbl) = {NULL, f1, f2, f3};`
    - `dbl (*f[N])(dbl) = {NULL, &f1, &f2, &f3};`
- `root = bisection(f[ i], a, b) ⇔ root = bisection(*f[i],a,b);`

fct.c

```
#include "find_roots.h"
```

```
dbl f1(dbl x)
```

```
{
```

```
    return (x*x*x - x*x + 2.0*x - 2.0);
```

```
}
```

```
dbl f2(dbl x)
```

```
{
```

```
    return (sin(x) - 0,7*x*x*x +3.0);
```

```
}
```

```
dbl f3(dbl x)
```

```
{
```

```
    return (exp(0.13*x) - x*x*x);
```

```
}
```

# Const & Volatile

---

- O padrão ANSI C introduziu os qualificadores de tipo `const` e `volatile`.
- `const` e `volatile` são qualificadores de tipo por restringirem ou qualificarem a forma como um identificador de determinado tipo pode ser usado.
- A declaração `const` costuma aparecer após uma classe de armazenagem
  - `static const int k=3;`
  - `k` pode ser iniciado e, depois disso, não pode ser atribuído, incrementado, decrementado ou modificado de alguma outra forma já que é `const`.

# Const & Volatile

---

- `const int a=7`  
`int *p = &a; /* o compilador irá reclamar*/`
  - Motivo: o ponteiro não poderá alterar o valor de `a`, pois violaria o conceito de que `a` é constante.
- `const int a = 7;`  
`const int *p = &a;`
  - Agora, `p` é um ponteiro para uma constante inteira e seu valor é o endereço de `a`.
  - O ponteiro `p` não é constante e podemos atribuir a ele algum outro endereço.
  - Não podemos atribuir um valor para `*p`, pois o objeto apontado por `p` não pode ser modificado.

# Const & Volatile

---

- `int a;`  
`int * const p = &a;`
  - Agora, `p` é um ponteiro constante do tipo inteiro e seu valor inicial é o endereço de `a`.
  - Não podemos atribuir um valor para `p`, mas podemos atribuir um valor para `*p`.
- `const int a=7;`
- `const int *const p=&a;`
  - Agora, `p` é um ponteiro constante que aponta para uma constante inteira. Nem `p` e nem `*p` pode ser atribuídos, incrementados ou decrementados.

# Const & Volatile

---

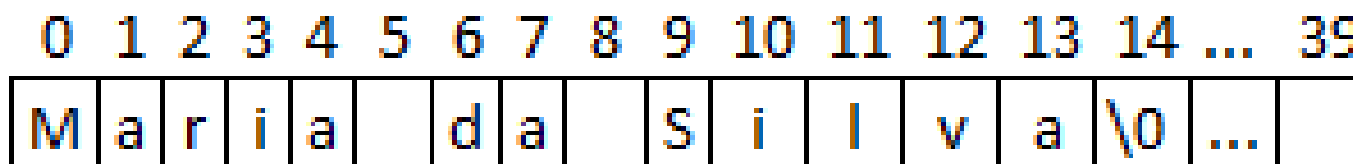
- O tipo qualificador volatile é raramente utilizado.
- Um objeto volatile pode ser modificado de alguma forma pelo hardware.
- `extern const volatile int real_time_clock;`
  - O termo `extern` indica que o compilador deve procurar nesse arquivo ou em algum outro arquivo.
  - O qualificador `volatile` indica que o objeto pode ser alterado pelo hardware.
  - O qualificador `const` indica que o objeto não pode ser atribuído, incrementado, decrementado dentro do programa.
  - O hardware pode mudar o clock, mas o código não.



# Strings

---

- A linguagem C armazena cadeia de caracteres (string) utilizando vetores.
- Cada posição do vetor armazena um caractere.
- Uma cadeia de caracteres (string) em C pode ser vista como uma coleção de bytes terminado pelo caractere NULL ('\0').
- O caractere NULL é atribuído automaticamente ao final da declaração de uma constante de *string* em C.
- O caractere NULL deve ser atribuído pelo usuário ao criar uma *string* via teclado.



nome[40]="Maria da Silva"

# Strings

---

- Diferença entre 'x' e "x".

- char expr = 'x';
- char expr = "x";



- Declare cadeias de caracteres maiores do que o necessário para evitar sobrescrever conteúdos.

char nome[10] = "João Oliveira da Silva Melo e Souza".

- Posições de memória reservadas a outras variáveis poderão ser sobrescritas gerando um erro difícil de ser detectado.
-

# Strings

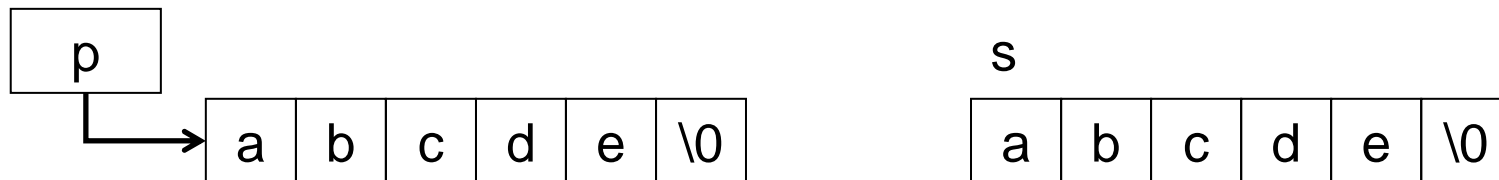
---

- Uma string constante como “abc” é tratada como ponteiro. As expressões a seguir são válidas:

“abc[1] e \*(“abc” + 2)

- `char *p = “abc”;`  
`printf(“%s %s\n”, p, p+1); /* abc bc é exibido */`
- Há diferença entre vetores e ponteiros na manipulação de strings.

`char *p = “abcde”;` e `char s[ ]=“abcde”;`



# Strings – Exemplo

---

```
#include <ctype.h>
```

```
int word_cnt (const char *s)
```

```
{
```

```
    int cnt = 0;
```

```
    while (*s!='\0'){
```

```
        while(isspace(*s))    /*pula espaços em branco*/
```

```
            ++s;
```

```
        if(*s != '\0'){    /*encontra uma palavra*/
```

```
            ++cnt;
```

```
            while(! isspace(*s) && *s != '\0')    /*pula a palavra*/
```

```
                ++s;
```

```
        }
```

```
    }
```

# Strings

---

- Funções que manipulam cadeias de caracteres
    - `char *strcat(char *s1, const char s2);`
      - concatena s1 e s2 colocando o resultado em s1.
    - `int strcmp(const char *s1, const char *s2);`
      - Compara duas cadeias de caracteres, retornando:
      - Valor = 0, se as cadeias forem idênticas.
      - Valor < 0, se string1 for lexicograficamente menor que string2.
      - Valor > 0, se string 1 for lexicograficamente maior que string 2.
    - `char *strcpy(char *s1, const char *s2);`
      - Copia os caracteres em s2 para s1 até \0 ser copiado.
      - Os caracteres em s1, se houver, são sobrescritos.
      - O ponteiro s1 é retornado.
    - `size_t strlen(const char *s);`
      - retorna o número de caracteres de string antes de \0.
-

# String

---

```
size_t strlen(const char *s)
{
    size_t n;
    for(n=0; *s != '\0'; ++s)
        ++n;
    return n;
}
```

```
char *strcpy(char *s1,
             register const char *s2)
{
    register char *p=s1;
    while (*p++ == *s2++)
        ;
    return s1;
}
```

# String

---

```
char *strc1;at(char *s1, register const char *s2)
{
    register char *p=s1;
    while (*p)
        ++p;
    while (*p++ = *s2++)
        ;
    return s1;
}
```

# Referências

---

Ascencio AFG, Campos EAV. Fundamentos de programação de computadores. São Paulo : Pearson Prentice Hall, 2006. 385 p.

Kelley, A.; Pohl, I., *A Book on C: programming in C*. 4ª Edição. Massachusetts: Pearson, 2010, 726p.

Kernighan, B.W.; Ritchie, D.M. C, *A Linguagem de Programação: padrão ANSI*. 2ª Edição. Rio de Janeiro: Campus, 1989, 290p.

Schildt, Herbet, *C Completo e Total*, Pearson, 2006,



# Exercício

---

- Faça um programa que leia uma matriz bidimensional digitada pelo usuário, onde:
    - Cada elemento da matriz armazena um valor de 0-255
    - O tamanho da matriz é informado pelo usuário
  - Realize a operação de convolução dessa matriz com uma outra matriz de tamanho 5x5
-

# Exercício

---

- Kernel do filtro Gaussiano (matrix de 5x5)

$$\frac{1}{273}$$

1	4	7	4	1
4	16	26	16	4
7	26	41	26	7
4	16	26	16	4
1	4	7	4	1

---

---

# FIM Aula 17

---