Curso de Programação C/C++

Alexandre, Minasi
Mário, Eduardo, Charles e Fábio
http://s2i.das.ufsc.br/

"A caminhada é feita passo a passo, com calma e perseverança."

Sistemas Industrias Inteligentes - S2i
Depto. de Automação e Sistemas - DAS
Universidade Federal de Santa Catarina - UFSC

Conteúdo do Curso

- **Programação C:**
  - Histórico e Motivação;
  - Conceitos Básicos;
  - Entrada e Saída Console;
  - Operadores;
  - Laços;
  - Comandos para a Tomada de Decisão;
  - Funções;
  - Diretivas do Pré-Processador;
  - Matrizes e Strings;
  - Tipos Especiais de Dados;
  - Ponteiro e Alocação Dinâmica de Memória;
  - Manipulação de Arquivos em C;

Conteúdo do Curso

- **Programação C++:**
  - Introdução a Programação em C++;
  - Classes e Objetos em C++;
  - Sobrecarga de Operadores;
  - Herança;
  - Funções Virtuais e Amigas;
  - Operações com Arquivos Iostream;
  - Namespaces (Definição de Escopo);
  - Templates (Tipo Genéricos);
  - Conteineres;
  - Tratamento de Exceções;
  - String;
- **Metodologia de Desenvolvimento de Software;**
- **Apresentação do Visual C++ (Aplicativo Demo):**
  - Porta IO;

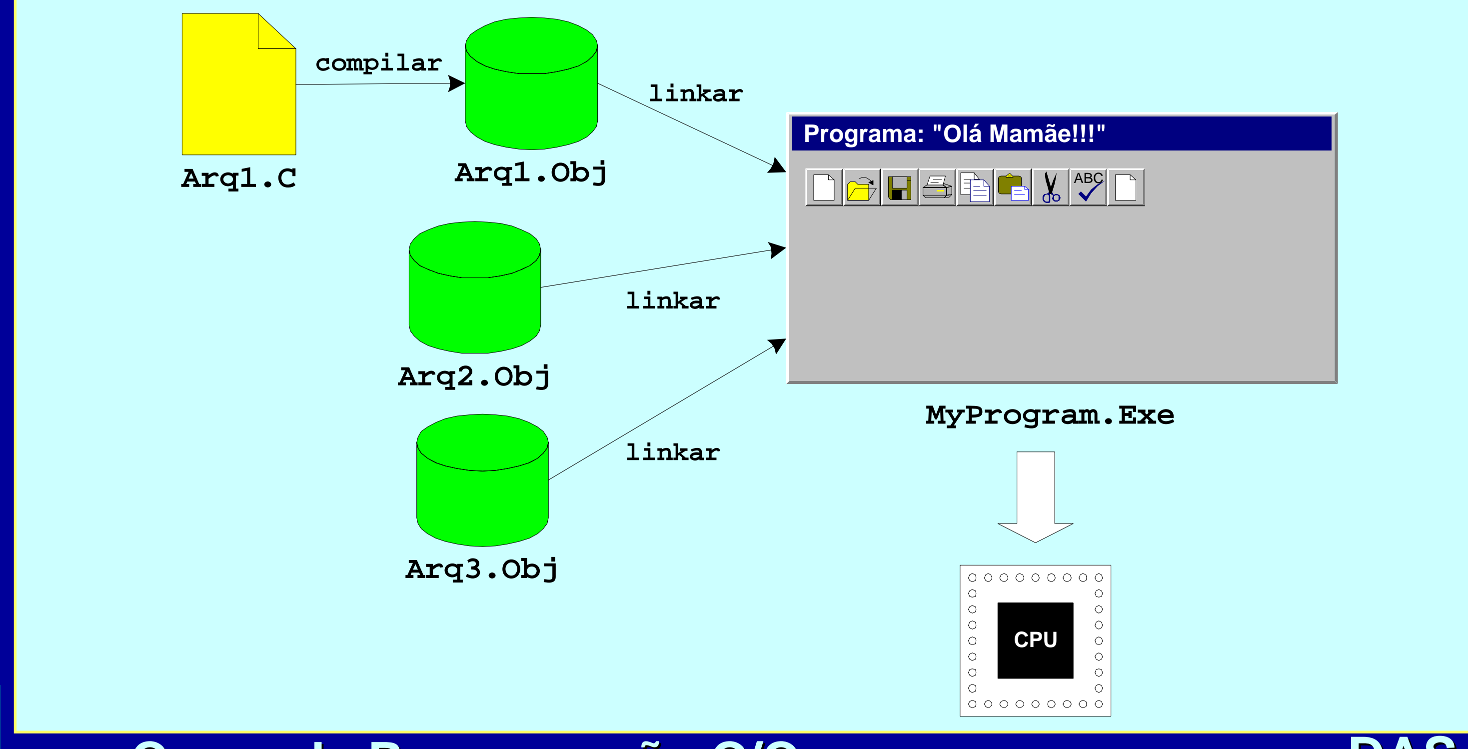
Histórico da Programação C/ C++

- Linguagem amplamente difundida na indústria;
- Originou-se da Linguagem BCPL e Bem 1970;
- Desenvolvida para o UNIX na Bell Laboratories para o DEC (Digital Equipment Corporation) PDP11;
- Criadores: Dennis M. Ritchie e Ken Thompson;
- Uma linguagem de alto nível, devido a estrutura fornecida;
- Um linguagem de baixo nível pois pode gerar códigos tão próximos da linguagem de máquina e otimizados como os códigos em Assembly;
- Velocidade de processamento e programas pequenos;
- Baseado em um Núcleo pequeno de funções -> Portabilidade;
- Exemplos de Sistemas Desenvolvidos em UNIX: UNIX, MS-DOS, TPW, compiladores, editores de texto, etc.

Quadro Comparativo

Criando um Programa Executável

- Escrever o programa em um editor de texto e salvar em um arquivo ASCII terminando em .C; Chamado Código-Fonte;
- Compile o arquivo gerando um arquivo .OBJ;
- Linkeditar os arquivos para gerar o aplicativo .EXE;



Estrutura Básica de um Programa C

```
< diretivas do pré-processador >
< declarações globais >;
main()
{
    < declarações locais >;     /* comentário */
    < instruções >;
}
< outras funções >
```

```
/* Programa : Bom Dia! */
#include <stdio.h>

void main()

{   printf("Bom Dia!!!!");
}
```

- Main( ) é única, determina o início do programa!!!

Variáveis e Tipos de Dados

```
/* Programa : Exemplo de variáveis! */
#include <stdio.h>
void main()
{
   int num; /* declaracao */
   num = 2; /* atribui um valor */
   printf("Este é o número dois: %d", num); /*acesso num*
}
```

Identificador	Categoria
char	Caracter
int	Inteiro
float	Real de ponto flutuante
double	Real de ponto flutuante de dupla precisão
void	Sem valor.

Variáveis & Tipos de Dados

Modificadores	Efeito
signed	Bit mais significante é usado como sinal
unsigned	Bit mais significante é parte do número (só +)
long	Estende precisão
short	Reduz precisão

Tipo	Tamanho	Valores possíveis
(signed) char	1 Byte	-128 a +127
unsigned char	1 Byte	0 a 255
(short) (signed) int	2 Bytes	-32.768 a +32.767
(short) unsigned int	2 Bytes	0 a 65.535
(signed) long int	4 Bytes	-2.147.483.648 a +..647
unsigned long int	4 Bytes	0 a 4.294.967.295
float	4 Bytes	±3,4E-38 a ±3,4E+38
long float	8 Bytes	±1,7E-308 a ±1,7E+308
double	8 Bytes	±1,7E-308 a ±1,7E+308

Constantes

- **#define** <nome> <valor>
- **const** <tipo> <nome> = <valor>**;**

```
/* Programa: Exemplo do uso de Constantes */
#define Size 4
#define true 1
#define false 0

void main()
{
   const char c = 'c';
   const int num = 10;
   const int terminou = true;
   int val = Size;
}
```

Ponteiros

- Armazenam o valor do endereço de uma variável;
- Tipo de Dado especial em C.
- <tipo> * <nome> = <endereço>; int *pNum = 0;
- Ponteiros ocupam 2 Bytes (tamanho de um endereço);
- Operador & fornece o endereço de uma variável;

```
/* Exemplo com Ponteiros */
void main()
{
   int num, valor;
   int *ptr = 0;
   ptr = &num;
   num = 10;
   valor = *ptr;
}
```

E/ S Console : Printf()

- Usa bibliotecas: <conio.h> e <stdio.h>
- Sintaxe: printf("string-formatação", <parm1>, <parm2>, ...);
- String-formatação é texto composto por:
- "%[Flags] [largura] [.precisão] [FNlh] < tipo > [\Escape Sequence]"

Flags	**Efeito**
-	justifica saída a esquerda
+	apresenta sinal (+ ou -) do valor da variável
em branco	apresenta branco se valor positivo, sinal de - se valor negativo
#	apresenta zero no início p/ octais, Ox para hexa e ponto decimal para reais

largura = número máximo de caracteres a mostrar
precisão = número de casas após a vírgula a mostrar
F = em ponteiros, apresentar como "Far" => base : offset (xxxx : xxxx)
N = em ponteiros, apresentar como "Near" => offset
h = apresentar como "short"
l = apresentar como "long"

E/ S Console : Printf()

Escape Sequence	Efeito
\\	Barra
\”	Aspas
\0	Nulo
\a	Tocar Sino (Bell)
\b	Backspace
\f	Salta Página
\n	Nova Linha
\o	Valor em Octal
\r	Retorna Cursor
\t	Tabulação
\x	Valor em hexadecimal

Tipo	Formato
%c	Caracter
%d,%i	Inteiro decimal (signed int)
%e,%E	Formato científico
%f	Real (float)
%l,%ld	Decimal longo
%lf	Real longo (double)
%o	Octal (unsigned int)
%p	Pointer (offset): Near, (base: offset): Far
%s	Apresenta uma string com final 00H
%u	Inteiro decimal sem sinal (unsigned int)
%x	Hexadecimal

```
#include <stdio.h>
void main() {
    printf("\n\t\tBom dia\n");
    printf("O valor de x é %7.3f\n", 3.141523);
    printf("Os valores de i e y são: %d %lf", 12, -3.12);
}
```

E/ S Console

- cprintf() imprime na tela na posição e cor definidas;
- getche(), getch() : lêem um caracter com ou sem imprimir na tela;
- putch(), putchar() : escreve um único caracter na tela;
- scanf(); inverso da printf(), use a mesma string-formatação mas recebe o endereço da variável como parâmetro;

```
#include <stdio.h>
#include <conio.h>
void main() {
    float x;
    char ch;
    printf("Entre com o valor de x : ");
    scanf("%f", &x);
    printf("Entre com um caracter : ");
    ch = getche();
}
```

Operadores

- Aritméticos: **+ - * /** e o sinal negativo: **- ;**
- Relacionais: **&&** (e), **| |** (ou), **!** (não), **<** (menor), **<** (menor igual), **>** (maior), **>=** (maior igual), **==** (igual), **!=** (diferente), **?:** (ternário);
- Binários, operam nos bits das variáveis, são eles: **&** (e), **|** (ou), **^** (ou exclusivo), **~** (não), **<<** (desloca a esquerda), **>>** (desloca direita);
- Operadores de Ponteiros: * (acessa o conteúdo do ponteiro), & (obtém o endereço de uma variável);
- Incrementais: ++ (incrementa) ; -- (decrementa);

  ++a; => incrementa primeiro a e depois a usa (pré-fixado)

  a++; => primeiro usa a variável e depois a incrementa (pós-fixado)
- Atribuição, combinação dos outros operadores, são eles: = , += , -= , *= , /= , %= , >>= , <<= , &= , |= , ^= ;

Operadores

<table>
<tr><th>Operadores</th><th>Tipos</th></tr>
<tr><td>! - ++ --</td><td>Unários; não lógicos e menos aritméticos</td></tr>
<tr><td>* / %</td><td>Aritméticos</td></tr>
<tr><td>+ -</td><td>Aritméticos</td></tr>
<tr><td>< > <= >=</td><td>Relacionais</td></tr>
<tr><td>== !=</td><td>Relacionais</td></tr>
<tr><td>&&</td><td>Lógico &</td></tr>
<tr><td>||</td><td>Lógico OU</td></tr>
<tr><td>= += -= *= /= %=</td><td>Atribuição</td></tr>
</table>

```
/*Operadores*/
void main(){
    int i = 10;
    int *ptr = 0;
    i = i + 1;
    i++;
    i = i - 1;
    i--;
    ptr = &i;
    (*ptr)++;
    ptr++; /*incrementa 2 Bytes */
}
```

Laços

- **For**:

for(<inicialização>; <teste>; <incremento>)

instrução;

```
int x, y, temp, conta;
        for(conta = 0; conta < 10; conta += 3)
                printf("conta=%d \n", conta);
        for( x=0, y=0; x+y < 100; x = x+1, y++)
        {
                temp = x + y;
                printf("%d\n", x+y);
        }
```

- **While:**

while(<expressão de teste>)

instrução;

```
int conta = 0;
while(conta < 10)
{
    printf("conta=%d\n",conta);
    conta++;
}
```

Do-While

- **Do-While:**

```
do
{
    instrução;
} while(expressão de teste);
```

- **Break:** interrompe o laço;
- **Continue:** volta a execução do laço;
- **Goto:** interrompe a sequência de execução e faz o processador pular para instrução após o label passado como parâmetro;

```
goto parte1;
parte1:
    printf("Análise dos Resultados. \n");
```

Comandos para Tomada de Decisão

IF:
```
if(expressão de teste)
        instrução;
else
        instrução;
```

```
if(ch == 'p') /* Forma basica de um comando if */
            printf("\nVoce pressionou a tecla 'p'!");
      else
            printf("\nErrou!");
```

SWITCH:
```
switch(variável ou constante)
{
        case constante1:
                instrução;
                break;
        default:
                instrução;
}
```

Switch( )

```
switch(F)
{
    case 0:
            printf("\nDomingo");
            break;
    case 1:
            printf("\nSegunda-Feira");
            break;
    case 2:
            printf("\nTerca-Feira");
            break;
    case 3:
            printf("\nQuarta-Feira");
            break;
    case default:
            printf("\nError”);
            break;
}
```

Funções

Unidade de código de programa autônoma para cumprir uma tarefa dada;

```
<tipo retorno> <nome func> (<tipo> <nome>, <...> <...>)
{
    <declarações locais>;
    <comandos>;
    return <expressão ou valor de retorno>;
}
```

```
#include <stdio.h>
doisbeep()
{
   int k;
   printf("\x7"); /*Beep*/
   for(k=1; k<10000; k++)
         ;
   printf("\x7"); /*Beep*/
}
```

```
void main()
{
   doisbeep();
   printf("Digite carac:");
   getche();
   doisbeep();
}
```

Funções

Funções Recursivas: Funções que chamam a si própria;

```
long int Fatorial(long int i)
{
    if(i > 1)
            return i*Fatorial(i-1);
    else
            return 1;
}
```

Prototipagem de funções:

<tipo> <nome>(<tipo1>, <tipo2>)

```
long int Fatora(long int);
void main()
{
    Fatora(15);
}
```

Funções

Classes Armazenamento:

- **auto:** locais;
- **extern:** globais;
- **static:** locais & estáticas;
- **extern static:** mesmo fonte;
- **register:** registradores;

```
#include <stdio.h>
int k = 10;
void soma();
void main()
{
   int register p = 0;
   for(p=0; p<5; p++)
         soma();
}
```

```
void soma()
{
   static int i = 0;
   int j = 0;
   i++;
   j += k;
   printf("i = %d\n", i);
   printf(”j = %d\n", j);
}
```

Diretivas do Pré-Processador

- **Comandos para o pré-processador do computador;**
- **#define** : usada para definir constantes e macros

```
#define  PI         3.14159
#define  ERRO       printf("\n Erro.\n");
#define  PRN(n)     printf("%.2f\n",n);
void main()
{
   float num1 = 27.25;
   float num2;
   num2 = 1.0/3.0;
   if(num2 == 0.0)
         ERRO;
   PRN(num2);
}
```

- **#undef** : remove a mais recente definição feita com #define.

```
#undef   ERRO
```

Diretivas do Pré-Processador

- **#include** : causa a inclusão de uma código fonte em outro, headers (.h)

```
#include <stdio.h> -> arquivos fonte do compilador
#include “mylib.h” -> arquivos locais
```

- **#if, #ifdef, #ifndef, #elif, #else e #endif :** compilação condicional **->** portabilidade .

```
#define DEBUG 1
#if DEBUG == 1
   printf(“\nERRO = ”, erro1);
#elif DEBUG == 2
   printf(“\nERRO = ”, erro2);
#endif
#ifndef WINDOWS
#define VERSAO “\nVersão DOS”
#else
#define VERSAO “\nVersão Windows”
#endif
```

Diretivas do Pré-Processador

- **defined** : Um alternativa ao #ifdef e #ifndef é o operador defined.

```
#if defined(UNIX) && !defined(INTEL_486)
...
#endif
```

- **#error :** provoca uma mensagem de erro de compilação.

```
#if TAMANHO > TAMANHO1
#error “Tamanho incompatível”
#endif
```

- **#pragma :** notificação ao compilador.

```
#pragma inline -> indica a presença de código Assembly no
arquivo.
#pragma warn -> avisa para ignorar "warnings" (avisos).
```

Matrizes & Strings

- Usada para representar um conjunto de dados de um tipo;
- **Declaração:** `<Tipo><nome>[<dimensão1>][<dimensão2>]...;`

```
int notas[5];
unsigned float tabela[3][2];
char cubo[3][4][6];
```

- **Acesso:** `<nome>[i][j][k][...]`

```
notas[3] = 4; -> acessa o quarto elemento
char ch = cubo[1][0][5];
```

- **Inicialização:** `<nome> = { a0, a1, ..., an-1};`

```
static int tab[] = {50, 25, 10, 5, 1};
char tabela[3][5] =  { {0, 0, 0, 0, 0},
                       {0, 1, 1, 1, 0},
                       {1, 1, 0, 0, 1} };
tabela[2] == {1, 1, 0, 0, 1};
```

Matrizes & Strings

- **Em Funções:** declara comum parâmetro ou tipo de retorno (pointer).

```
#define  TAMAX        30
void ordena(int vector[], int size);
main()
{
   int list[TAMAX];
   ordena(list, TAMAX);
}
```

OBS.: `list == &list[0];`

- **String:** matriz de caracteres terminada com '\0'.

```
char name[] = "Marcos";
name[6] == '\0';
```

- **Declaração & Inicialização:**

```
char name[15];          char name[] = "Bom Dia!!!";
char text[]={'B','o','m',' ','d','i','a','!','\0'};
```

Matrizes & Strings

- **Funções de Manipulação de Strings: ;**

- **`Strlen():`** retorna o tamanho ocupado por uma string sem contar o delimitador '\0'.

```
strlen(string);
```

- **`Strcat():`** concatena duas strings

```
strcat(string1, string2);
```

- **`Strcmp():`** compara duas strings

```
strcmp(string1, string2);
```

- **`Strcpy():`** copia uma string em outra

```
strcpy(string1, string2);
```

Tipos Especiais de Dados

- **Como definir novos tipos de dados;**
- **`Typedef:`** criação de novos tipos de variáveis

```
typedef <tipo> <definição>
typedef double real_array [10];   /*novo tipo*/
real_array x, y;
```

- **`Enum`** **:** atribuir valores inteiros seqüenciais a constantes. Funciona como um label (nomeação).

```
enum cores
{
   verde        = 1,
   azul         = 2,
   preto,        /*recebe valor anterior + 1 = 3 */
   branco       = 7
};
int i = verde;
```

Tipos Especiais de Dados

- **Estruturas:** criação de novos tipos de dados complexos, composto por um conjunto de dados com diferentes tipos.

```
struct <nome>
{
    <tipo> <nome campo>;
}[<variáveis   deste   tipo>];
/*Declaração   de   variáveis*/
struct <nome> <nome var>;
```

```
struct Dados_Pessoais
{
    char  Nome[81];
    int   Idade;
    float Peso;
} P1;
struct Dados_Pessoais P2;
```

```
typedef struct <nome>
{
    <tipo> <nome-campo>;
} <nome tipo>;
/* Declaração de variáveis/
<nome-tipo><nome-variável>;
```

```
typedef struct Dados_Pessoais
{
    char Nome [81];
    int Idade;
} Pessoa;
Pessoa P1, P2 = {“Karen”, 17};
P1.Idade = 20;
```

Tipos Especiais de Dados

- **Uniões:** similar a estruturas, entretanto todos os dados são armazenados no mesmo espaço de memória, que possui a dimensão do maior dado.

```
union <nome>
{
      <tipo> <nome campo>;
}[<variáveis deste tipo>]; /* Declaração de variáveis/ union <nome> <nome var>;
```

```
union Oito-bytes
{
        double x;
        int    i[4];
} unionvar;
unionvar.x = 2.7;
unionvar.i[1] = 3;
```

- **Bitfields:** tipo especial de dado para representação de bits.

```
struct <nome>
{
    <tipo><campo>:<numbits>;
};
```

```
struct Registro_Flags
{
        unsigend int Bit_1 : 1;
        unsigend int Bit_2 : 1;
        unsigend int Bit_3 : 1;
        unsigend int Bit_4 : 1;
        unsigend int Bit_5 : 1;
}
```

Ponteiros & Alocação Dinâmica

- **declaração**: declarado com o operador (*) antes do nome da variável.

```
<tipo> *<nome da variável> = <valor inicial>;
```

```
int num = 15;
int * ptr = &num;
*ptr = 10;
```

- **operações:** podem sofre operações como os tipos básicos.

```
ptr++; ptr--; ptr = ptr + 3; (ptr >= &num); (ptr != &num)
```

- **funções :** recebem ponteiros como parâmetro ou retornam ponteiros.

Velocidade, trabalhar com tipos de dados complexos ou grandes, permite a modificação do valor dos parâmetros (múltiplos dados de retorno), flexibilidade (ponteiros tipo void).

```
void altera(int *, int *);
int x = 0, y = 0;
altera(&x, &y);
```

Ponteiros & Alocação Dinâmica

- **matrizes**: matrizes são ponteiros para uma lista de dados do mesmo tipo.

```
*(matriz + índice) == matriz[índice]
float notas[10];                int i = 0;
float *ptr = 0;                 ptr = notas;
(notas + i++) == (ptr++)
```

- **strings**: ponteiro para uma lista de caracteres terminada em '\0'.

```
char salute1[] = char *salute2 = "Saudacoes";
char * * ListaAlunos = {"Alexandre", "Paulo"};
```

- **ponteiros para ponteiros**: o dado armazenado em um ponteiro é na verdade um ponteiro para outra variável.

```
int tabela[4][5];
tabela[2][4] == *(ptr + 2*5 + 4) == *(ptr + 14);
int linha[5] = tabela[2] == *(tabela + 2);
tabela[2][4]==linha[4]== *(linha+4)== *(*(tabela+2)+4);
cubo[i][j][k] = *(*(*(cubo + i) + j) + k);
```

Ponteiros & Alocação Dinâmica

- **Argumentos do programa**: comandos passados ao programa.

```
int main(int argc, char* argv[]);
argc:número de parâmetros; argv: lista de paramâmetros.
C:> jogo  xadrex.txt       2     3.14159
argv[0] == "jogo";   argv[1] == "xadrex.txt";
argv[2] == "2";      argv[3] == "3.14159";
```

- **Estruturas:** ponteiros são muito úteis se combinados com estruturas.
  - **Lista-ligadas;**

```
struct lista /* declara estrutura */
{
   char titulo[30];  char autor[30];
   int regnum;       double preco;
} livro[2];
struct lista *ptrl;
ptrl = &(livro[0]);
livro[0].preco == (*ptrl).preco == ptrl->preco = 89.95;
```

Ponteiros & Alocação Dinâmica

- **Heap**: área livre de memória, usada para a alocação dinâmica.

```
#include <stdio.h>
struct xx
{
   int num1;
   char ch1;
};
void main()
{
   struct xx * ptr = 0;
   ptr = (struct xx *) malloc(sizeof(struct xx));
   printf("ptr = %x\n", ptr);
   ptr = (struct xx *) calloc(2,sizeof(struct xx));
   if(ptr != 0)
         *(ptr + 1).num1 = 15;
   free(ptr);
}
```

Manipulação de Arquivos em C

- **Tipos**: arquivos podem ser binários ou de texto.
- **Declaração, abertura e fechamento:**

```
FILE *File_ptr;
File_ptr = fopen("Nome do Arquivo","<I/O mode>);
<I/O Mode> = { "r", "w", "b", "a", "t", "+"}
fclose(File_ptr);
exit();
```

- **Leitura e escrita de caracteres:**

```
int mychar;
FILE *fileptr;
fileptr =
fopen("Arq1.TXT","rw");
mychar = getchar(fileptr);
```

```
while(mychar != EOF)
{
   printf("%c", mychar);
   mychar = getchar(fileptr);
}
putchar('@',fileptr);
fclose(fileptr);
```

Manipulação de Arquivos em C

- **Strings**: ler e escrever strings inteiras em um arquivo.

```
FILE *fileptr;
char line[81];
fgets(line, 80, fileptr);
fscanf(fileptr, "%s", line);
fputs(line, fileptr);
fputs("\n", fileptr);
fprintf(fileptr,"%s\n", line);
```

- **Standard I/O:**

```
#define     stdin  (&_streams[0])   /* teclado */
#define     stdout (&_streams[1])   /* monitor */
#define     stderr (&_streams[2])   /* monitor */
#define     stdaux (&_streams[3])   /* porta serial */
#define     stdprn (&_streams[4])   /* impressora padrão*/
fgets(string, 80, stdin);
fputs(string, stdprn);
```

Manipulação de Arquivo em C

- **Dados Binários**:
  - usando as funções anteriores no modo binário, i.e., opção “b”;
  - usando as funções abaixo;

Escrita:
`fileptr = fopen(filename, “wb”);` `fwrite(&dados, sizeof(dados),1 ,fileptr);`
Leitura:
`fileptr = fopen(filename, “rb”);` `fread(&dados,sizeof(dados),1 ,fileptr);`

Programação C++

- **C**: baixa reusabilidade, legilibilidade e facilidade de manutenção;
- **Programa de Computador** é a representação abstrata de um ambiente ou realidade (modelo matemático ou mundo real);
- **Difícil modelagem** da realidade através de linguagens estruturais;
- **Orientação a Objetos:** Todo ambiente pode ser modelado e simulado a partir de uma descrição dos objetos que o compõe e das relações entre eles.
- **Nível de detalhe == nível de abstração** necessário.

Programação C++

- **C++** é uma adaptação de **C** a **Orientação a Objetos**.
- **C++ herda de C** a capacidade de gerar programas pequenos, otimizados, de "baixo-nível" e portáveis.
- **C++** : programas bem estruturados e documentados, boa reusabilidade, de fácil manutenção e expansão, melhores projetos, várias ferramentas;
- **C++ :** requer mais memória, menor velocidade de execução e difícil otimização do tempo de execução.

Novidades em C++

- pode-se **declarar uma** variável em qualquer ponto de um programa;
- não precisa declarar **estruturas** utilizando **typedef**;
- classes e estruturas podem ter componentes **públicos e privados**;
- `extern int n;`   `int n;`   **`n = ::n; // escopo`**
- `// nova forma de definir comentários`
- `for(`**`int i =0`**`, j=3; i+j<10; i = j++ )`

- **`I/O em C++`**: Streams `<iostream>`
  - biblioteca Stream do C++, terminal: **cin**, **cout** e **cerr**;
- **`cout : saída padrão;`**
  - **Sintaxe:** **`cout << expressão;`**

Variáveis	C++
`int x = 2;`	`cout << "x = " << x;`
`float f = 1.2, g = 3;`	`cout << f << " " << g;`
`double dou = 2.14;` `char ch = 'F';`	`cout << "valor = "` `<< dou << "\nsex = " << ch;`

Novidades em C++

- Caracteres especiais, flags, manipuladores de formatação;

```
cout  << " \n\n" << setiosflags(ios::left);
cout << setprecision(2);
cout << "\n\t" << ”float: " << setw(12) << 12.124;
cout << "\n\t" << ”char:  " << setw(12) << ‘a’;
cout << "\n\t" << setfill('.')
     << “string " << setw(12) << "TYE"
```

- **`cin : entrada padrão;`**
  - **Sintaxe: `cin >> variável;`**
  - Múltiplas entradas : separadas por espaço + enter;
  - `cin >> hex >> n;`
  - funções de I/O em C também estão disponíveis;

Variáveis	C++	C
`int x;`	`cin >> x;`	`scanf(“%d”,&x);`
`float f,g;`	`cin >> f >> g;`	`scanf(“%f  %f”, &f, &g);`
`double dou; char ch;`	`cin >> dou >> ch;`	`scanf(“%Lf %c”, &dou, &ch);`

Funções em C++

- **Inicialização:**

```
void linha( int n = 20, char ch, int cor); // Não Ok
void linha(int n, char ch = '*', int cor = 0); //Ok
```

- **Sobrecarga:**

```
int cubo(int n);
float cubo(float n);
```

- **Inline:** inserção de uma cópia da função em todo local da sua chamada.

```
inline  int cubo(int n);
```

- **Operador Referência & :** outro nome para a mesma variável.
  - passagem de parâmetros;
  - vantagens dos ponteiros sem manipular com endereços de memória;

```
int n;      int & n1 = n; // referência -> inicializada
void reajusta(float& num, float& p); // Protótipo
reajusta( 13.4567, 5.0); // Chamada
void reajusta(float& num, float& p){num *= p;} //Def.
```

Alocação Dinâmica em C++

- Operadores New (Aloca) e Delete (Libera);
- Mais seguros e fáceis de usar;
- Podem ser sobrecarregados (reimplementados);
- Cuidado com ponteiros nulos;

a) como operadores ou função:

```
ptr_tipo = new tipo;          ptr_tipo = new(tipo);
int * ptr1 = new int;         int * ptr2 = new(int);
delete ptr1;                  delete ptr2;
```

b) alocação e liberação de matrizes:

```
ptr_tipo = new tipo[tamanho_matriz];
int * ptr3 = new int[10];
delete[] ptr3;                  //  delete[10] ptr3;
```

c) alocação e liberação de matrizes de ponteiros:

```
ptr_ptr_tipo = new tipo * [tamanho_matriz];
int * * ptr4 = new int * [10];
delete[] ptr4;                  //  delete[10] ptr4;
```

Classes e Objetos

- Modelar o ambiente;
- Componentes do ambiente → objetos;
- Jogo de Xadrez:
  - Atributos: dimensões, número de posições livres e cor do tabuleiro;
  - Métodos: desenhar o tabuleiro na tela e movimentar as peças;
- Peças do Jogo de Xadrez
- Objetos de um mesmo tipo → classes;
- Classe "Peão" x Classe "Peça do Xadrez"
- Outros componentes: Jogador, Juíz, Usuário, ...

Classes & Objetos

```
#include <iostream>
using namespace std;
class Retangulo     // Define a classe
{
private:
      int bas, alt; // atributos privados
public:
      static int n; // atributos públicos
      // Construtores e Destrutores
      Retangulo()                  { bas=0; alt=0; n++; }
      Retangulo(int a, int b=0)    { bas=a; alt=b; n++; }
      ~Retangulo()           { n--; }
      // métodos da classe
      void Init(int b, int h)     { bas = b; alt = h; }
      void PrintData();
};
```

Classes & Objetos

```
void Retangulo::PrintData() // Define função membro
{
        cout << "\nBase = " << bas << " Altura = " << alt;
        cout << "\nArea = " << (bas*alt);
}
// inicialização da variável estática
int Retangulo::n = 0;
int main( int argn, char ** argc )
{
        Retangulo X, Y[5]; // Declaração de objetos
        Retangulo C[2] = {Retangulo(2,3),Retangulo(5,1)};
        X.Init( 5, 3);
        X.PrintData();        // Chama função membro
        Y[1].PrintData();
        (C+1)->PrintData();
        cout << "\n Quantidade de Objetos : " << C[1].n;
        return 0;
}
```

Sobrecarga de Operadores

- **Reimplementar as operações existentes para novos tipos de dados.**
- **A = B + C;**
- **Função operadora `->` Membro da classe;**
- **Respeitar a definição original do operador;**
- **Limitados pelo conjunto de operadores já existentes;**
- **A precedência entre os operadores é mantida;**
- **Operadores que não podem ser sobrecarregados: `(.)(->)(::)(?:)`**
- **Operadores Unários - 1 parâmetro: `(++, --, -)`**
- **Operadores Binários - 2 parâmetros: `(+, -, *, /,...)`**
- **Operadores Binários requerem um argumento;**
- **Reimplementar os operadores associados a `cout` e `cin`: `(<<)` e `(>>)`;**

Sobrecarga de Operadores

```
#include <iostream>
using namespace std;
class Ponto
{
private:
  int X, Y;
public:
  Ponto(int aX=0, int aY=0)
       { X = aX; Y = aY; }
  void PrintPt() const {
       cout << '(' << X
       << ',' << Y << ')';
  }
  Ponto operator ++() {
       ++X; ++Y;
       Ponto Temp(X,Y);
       return Temp;
  }
};
```

```
int main(int n, char ** argc)
{
       Ponto P1, P2(2,3);
       cout << "\n p1 = ";
       P1.PrintPt();
       cout << "\n++p1 = ";
       (++P1).PrintPt();

       cout << "\n++p2 = ";
       (++P2).PrintPt();
       P2 = ++P1;
       cout << "\n p2 = ";
       P2.PrintPt();
}
```

Sobrecarga de Operadores

```
#include <iostream>
using namespace std;
enum bool
{
	false,
	true
};
class Ponto
{
 public:
	int X, Y;
	Ponto(int aX = 0, int aY = 0) { X = aX; Y = aY; }
	Ponto operator ++(); // prefixado
	Ponto operator ++(int); // pos-fixado
	Ponto operator +(Ponto const aP) const;
	Ponto operator +=(int const aN);
	bool operator ==(Ponto const aP) const;
};
```

Sobrecarga de Operadores

```
class PFloat
{
public:
 float X, Y;
 PFloat(float aX = 0, float aY = 0)     { X = aX; Y = aY; }
 operator Ponto() const    { return Ponto((int)X, (int)Y);}
};
```

```
Ponto Ponto::operator ++()
{ //prefixado
      ++X; ++Y;
      return *this;
}

Ponto Ponto::operator ++(int)
{ //pos-fixado
      ++X; ++Y;
      return Ponto(X-1, Y-1);
}
```

Sobrecarga de Operadores

```
Ponto Ponto::operator +(Ponto const aP) const
{
	return Ponto(X + aP.X, Y + aP.Y);
}

Ponto Ponto::operator +=(int const aN)
{
	X += aN; Y += aN;
	return *this;
}

bool Ponto::operator ==(Ponto const aP) const
{
	return ((X == aP.X)&&(Y == aP.Y));
}

ostream & operator<<(ostream & OS, Ponto const aP)
{
	OS << '(' << aP.X << ','<< aP.Y <<')';
}
```

Sobrecarga Operadores

```
int main(int argn, char ** argc)
{
        Ponto P1, P2(2,3);
        PFloat Pf( 2.12, 3.14);
        P1 = Pf; \\ Conversao
        cout << "\n p1 = " << P1;
        cout << "\n ++p1 = " << ++P1;
        cout << "\n p2 == p1 ? -> " << (P1 == P2);
        cout << "\n p2 = " << P2;
        cout << "\n p2 + p1 = " << (P2+P1);
        cout << "\n pf = " << Pf; \\ Conversao
        return 0;
}
```

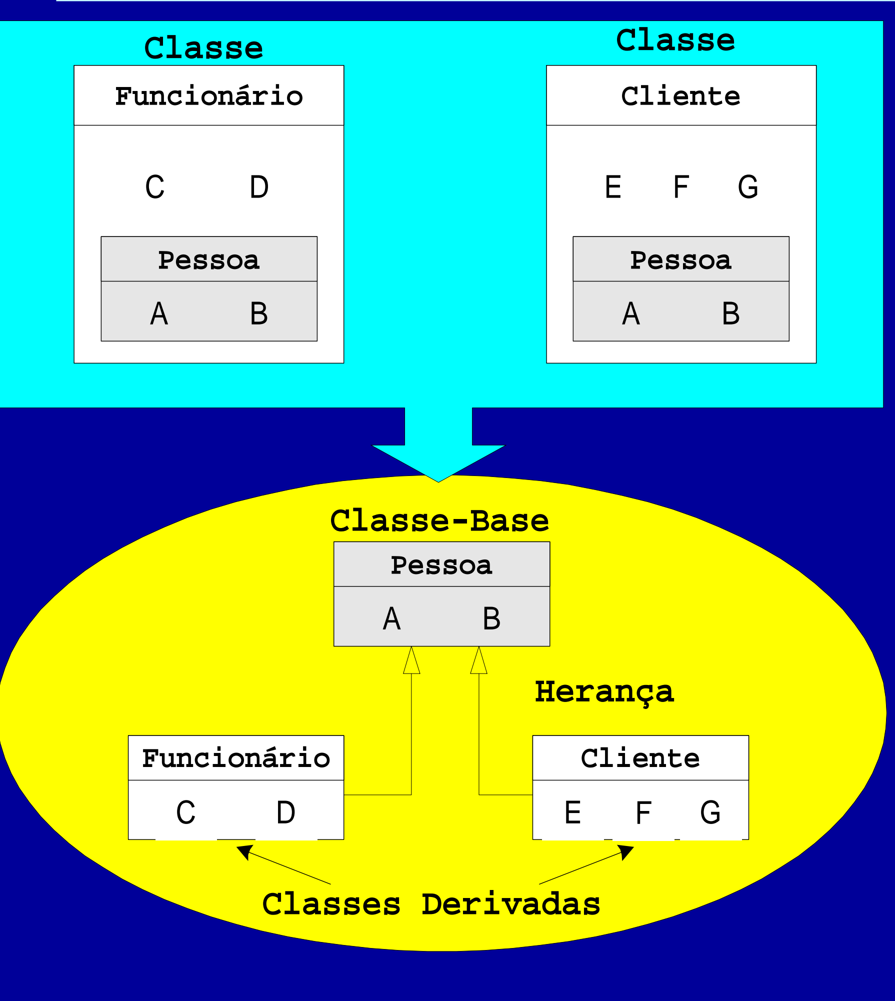
. A conversão pode ser feita no construtor, definindo-se um construtor para o tipo definido.

```
Ponto::Ponto( PFloat P) { X =(int)P.X; Y =(int)P.Y;}
```

. Construtor de Cópia;

```
Ponto::Ponto( Ponto P) { X = P.X; Y = P.Y; }
```

Herança



- dividimos classes em sub-classes;
- herda as características;
- classe-base x classe derivada;
- classe derivada acrescenta atributos e métodos específicos;
- flexibilidade;
- organização;
- reusabilidade;
- expansibilidade;
- Programa bem modelado : claro, veloz, enxuto, flexível, pequeno tempo de projeto e boa manutenção.
- Herança Múltipla

Herança

```
#include <iostream>
#include <string>
using namespace std;
#define  Nome_Size   80
class Pessoa
{
protected:
       char * Nome;
       int            Idade;
       long int       RG;
public:
       Pessoa(){ Nome = new char[Nome_Size]; Idade=0; RG=0;}
       Pessoa(char* aNome, int aId=0, long int aRG=0);
       Pessoa(Pessoa const & aP);
       ~Pessoa() {   if(Nome != 0) { delete Nome; Nome = 0;} }
       char const * GetNome()      const { return Nome;   }
       int      GetIdade() const   { return Idade; }
       long int GetRG()   const   { return RG;    }
       void                 GetData();
};
```

Herança

```
Pessoa::Pessoa(char* aNome, int aId=0, long int aRG )
{
        Nome = new char[Nome_Size];
        strcpy( Nome, aNome);  Idade = aId; RG = aRG;
}
Pessoa::Pessoa(Pessoa const & aP)
{
        Nome = new char[Nome_Size];
        strcpy( Nome, aP.GetNome());
        Idade = aP.GetIdade();
        RG = aP.GetRG();
}
void Pessoa::GetData()
{
 cout << "\nNome : "; cin.getline( (char *)Nome, Nome_Size);
 cout << "Idade : ";  cin  >> Idade;
 cout << "RG : ";     cin  >> RG;
}
```

Herança

```
ostream & operator <<(ostream & OS, Pessoa & const P)
{
       OS << "\n\n Dados Pessoais ---------------"
              << "\n Nome  : " << P.GetNome()
              << "\n Idade : " << P.GetIdade()
              << "\n RG    : " << P.GetRG();
       return OS;
}
class Cliente : public Pessoa
{
protected:     int    Conta;
               int    Saldo;
public: Cliente() {  Conta = 0; Saldo = 0; }
        Cliente(char * aNome, int aConta = 0, int aId = 0,
                 long int aRG = 0, int aSaldo = 0);
       int GetConta() const { return Conta;      }
       int GetSaldo()const  { return Saldo;      }
       void                 GetData();
};
```

Herança

```
Cliente::Cliente(char * aNome, int aConta, int aId, long int aRG,
int aSaldo) : Pessoa(aNome, aId, aRG)
{
       Conta = aConta;
       Saldo = aSaldo;
}
void Cliente::GetData()
{
       Pessoa::GetData();
       cout << "Conta : ";  cin  >> Conta;
       cout << "Saldo : ";  cin  >> Saldo;
}
ostream & operator <<(ostream & OS, Cliente & const C)
{
       OS << Pessoa(C);
       return OS << "\n Conta : " << C.GetConta()
                 << "\n Saldo : " << C.GetSaldo();
}
```

Herança

```
int main(int argn, char ** argc)
{
        Pessoa P1("Carlos", 18, 1315678);
        Cliente C1, C2("Pedro", 1234, 17, 123432);

        C2.GetData();
        C1.Pessoa::GetData();

        Pessoa P2 = P1;
        Pessoa P3 = C1;

        cout << P1 << P2 << P3 << C1 << C2;

        return 0;
}
```

Herança Múltipla

```
class Idade
{
protected:
        int idade;
public:
        Idade(int n = 0) { idade = n; }
        Print()          { cout << "Idade: " << idade; }
};

class Nome
{
protected:
        char nome[80];
public:
        Nome(char n[] ="" ) { strcpy(nome, n); }
        Print()             { cout << "Nome: " << nome; }
};
```

Herança Múltipla

```
class Profissao
{
protected:
        char nome[80];
public:
        Profissao(char p[] ="") { strcpy(nome, p); }
        Print() { cout << "Profissao : " << nome; }
};

class Pessoa : public Nome, public Profissao, private Idade
{
public:
        Pessoa(char n[] = "", char p[]= "", int id = 0)
            : Nome(n), Profissao(p), Idade(id)  { }
        Print()
        { Nome::Print(); Profissao::Print(); Idade::Print();  }
};
```

Funções Virtuais

- Redefinição de funções membros de classes bases em classes derivadas;
- chamadas indiretas;

```
#include <iostream>
using namespace std;
class Base
{       public:
                virtual void print() { cout << "\nBASE"; }
};
class Deriv1 : public Base
{       public:
                void print()  { cout << "\nDeriv1"; }
};
class Deriv2 : public Base
{       public:
                void print() { cout << "\nDeriv2";}
};
```

Funções Virtuais

```
int main( int argn, char ** argc )
{
        Base * p[3]; // matriz de ponteiros da classe-base
        Base    B;              // Objeto da classe base
        Deriv1 D1;              // Objeto da classe Derivada 1
        Deriv2 D2;              // Objeto da classe Derivada 2

        p[0] = &B;              // inicializa ponteiros ...
        p[1] = &D1;
        p[2] = &D2;

        p[0]->print();           // p[0]->Base::print();
        p[1]->print();           // p[1]->Deriv1::print();
        p[2]->print();           // p[2]->Deriv2::print();

        return 0;
}
```

Classe-Base Virtual

- compartilhar classes-bases;

```
class Base
{
protected:
	int valor;
public:
	Base(int n = 0) {valor = n;}
	virtual void Print(){ cout << "\nValor : " << valor; }
};
class Deriv1 : virtual public Base
{
public:
	Deriv1(int n = 0)   :   Base(n)     { }
};
class Deriv2 : virtual public Base
{
public:
	Deriv2(int n = 0)   :   Base(n)     { }
};
```

Classe-Base Virtual

```
class Super : public Deriv1, public Deriv2
{
public:
        Super(int n = 0) : Base(n) { }
        int RetValor()       { return valor; }
        void Print()
                {cout << "\nValor do Super-Objeto : "<<valor;}
};
int main(int argn, char ** argc)
{
        Base    B(5);
        Deriv1 D1(10);
        Super  S(343);
        B.Print();
        D1.Print();
        S.Print();
        return 0;
}
```

Funções Amigas

- Funções externas mas que acessam dados protegidos;

```
#include <iostream>
#include <strstrea>
using namespace std;

class Data; // Declara que existe esta classe

class Tempo
{
private:
        long h, min, s;
public:
        Tempo(long hh = 0, long mm = 0, long ss = 0)
                { h = hh; min = mm; s = ss; }
        friend char * PrintTime( Tempo &, Data &);
};
```

Funções Amigas

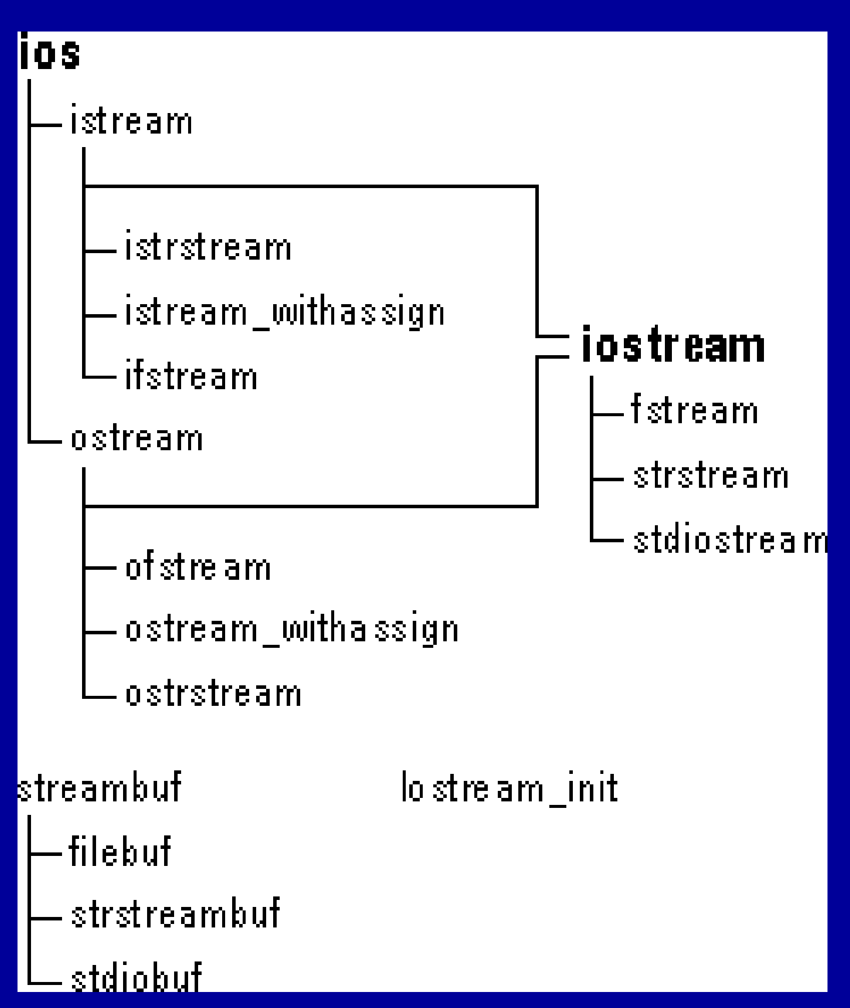
```
class Data
{
private:
        int d, m, a;
public:
        Data(int dd = 0, int mm = 0, int aa = 0)
                { d = dd; m = mm; a = aa; }
        friend char * PrintTime( Tempo &, Data &);
};
char * PrintTime( Tempo & Tm, Data & Dt)
{
 char * temp = new char[50];
 memset(temp, '\0', 50);
 strstream sIO(temp, 50, ios::out);
 sIO << "\n Relogio-> \t" << Tm.h << ":" << Tm.min << ":" << Tm.s;
 sIO << "\n Data->    \t" << Dt.d << "/" << Dt.m   << "/" << Dt.a;
 return temp;
}
```

Funções Amigas

```
int main( int argn, char ** argc)
{
        Tempo Tm( 15, 56, 4);
        Data   Dt( 9, 5, 2000);

        char * str = PrintTime( Tm, Dt);

        cout << "\n" << str;
        delete str;

        return 0;
}
```

- **`friend class Data`**
- **`char * Data::PrintTime( Tempo & Tm) {...}`**
- **`Dt.PrintTime(Tm);`**

Classes I OSTREAM



- Buffer para enviar e receber dados;
- Arquivos, teclado, vídeo, impressora, portas de comunicação (TCP/IP) ....
- ≠ Objetos ≠ Aplicações
- ostream_withassign : cout
- istream_withassign : cin
- operadores de inserção >> e <<
- ios: classe base da biblioteca
- istream: get( ), getline( ), read( )
- ostream: put( ), write( )
- iostream: istream + ostream
- fstream : arquivos
- strstream : buffer de caracteres

Classes I OSTREAM

```
#include <iostream>
#include <fstream>
#include <iomanip>
using namespace std;
#define  Nome_Size  80
class Pessoa
{
protected:
  char * Nome;       int Idade;    long int RG;
public:
       Pessoa();
       Pessoa(char * aNome, int aId = 0, long int aRG = 0);
       Pessoa(Pessoa const & aP);
       ~Pessoa();
       char const * GetNome()     const { return Nome;  }
       int     GetIdade() const { return Idade; }
       long int GetRG()             const  { return RG;   }
       void   GetData();
friend ostream & operator <<(ostream & OS, Pessoa const & P);
friend istream & operator >>(istream & IS, Pessoa & P);
};
```

Classe IOSTREAM

```
Pessoa::Pessoa()
{
  Nome = new char[Nome_Size];
  Idade = 0;
  RG = 0;
}
Pessoa::Pessoa( char * aNome, int aId, long int aRG)
{
  Nome = new char[Nome_Size];
  strcpy( Nome, aNome);
  Idade = aId;
  RG = aRG;
}
Pessoa::Pessoa(Pessoa const & aP)
{
  Nome = new char[Nome_Size];
  strcpy( Nome, aP.GetNome());
  Idade = aP.GetIdade();
  RG = aP.GetRG();
}
```

```
Pessoa::~Pessoa()
{
  if(Nome != 0)
  {
     delete Nome;
     Nome = 0;
  }
}
```

Classe I OSTREAM

```
void Pessoa::GetData() {
        cout << "\nNome : ";
        cin.getline( (char *)Nome, Nome_Size);
        cout << "Idade : ";
        cin  >> Idade;
        cout << "RG : ";
        cin  >> RG;
}
ostream & operator <<(ostream & OS, Pessoa const & P){
        OS      << P.GetNome()
                << "\n" << P.GetIdade()
                << "\n" << P.GetRG();
        return OS;
}
istream & operator >>(istream & IS, Pessoa & P){
  IS.getline( (char *)P.Nome, Nome_Size);
  IS  >> P.Idade;
  IS  >> P.RG;
  return IS;
}
```

Classe IOSTREAM

```
int main(int argc, char* argv[])
{
        Pessoa        Eu("Rodrigo", 20, 231123);
        ofstream      FOut("Teste.TXT", ios::out);
        FOut << Eu << "\n Um dia a casa cai! \n "
             << setw(12) << setprecision(3) << 12.2345;
        FOut.close();
        ifstream FIn;
        Fin.open("Teste.TXT", ios::in)
        Pessoa Vc;
        char Buffer[Nome_Size];
        FIn >> Vc;
        cout << "\nVoce: \n" << Vc;
        while(FIn) //Enquanto nao acabar o arquivo
        {
                FIn.getline( Buffer, Nome_Size);
                cout << "\nBuffer : " << Buffer;
        }
        FIn.close();
        return 0;
}
```

IOSTREAM - Função Open( )

Modos	Descrição
ios::in	Abre para leitura (default de ifstream)
ios::out	Abre para gravação (default de ofstream)
ios::ate	Abre e posiciona no final do arquivo – leitura e gravação
ios::app	Grava a partir do fim do arquivo
ios::trunc	Abre e apaga todo o conteúdo do arquivo
ios::nocreate	Erro de abertura se o arquivo não existir
ios::noreplace	Erro de abertura se o arquivo existir.
ios::binary	Abre em binário (default é texto).

Bits	Função	Comentário
ios::goodbit	good()	Nenhum bit setado. Sem erros
ios::eofbit	eof()	Encontrado o fim-de-arquivo
ios::failbit	fail()	Erro de leitura ou gravação
ios::badbit	bad()	Erro irrecuperável

```
ifstream fin(“Teste.txt”, ios::in|ios::binary);
fstream fio; \\ Arquivo para leitura e gravaçao!!!!
fio.open(“Lista.dat”, ios::in|ios::out|ios::ate);
```

IOSTREAM - Buffer de Caracteres

```
#include <iostream>
#include <strstrea>
using namespace std;
int main(int argc, char* argv[])
{
  ostrstream OS; // Cria buffer para a escrita
  OS << "123.45 \t" << 555.55 << "\t" << 333;
  OS << "\n\n Sorte = Estar_Preparado + Oportunidade";
  OS << ends;
  char * ptr = OS.str();
  double x, y;
  int i;
  istrstream IS(ptr);
  IS >> x >> y >> i;
  cout << x << '\t' << y << '\t' << i << '\n' << ptr;
  return 0;
}
```

IOSTREAM - Periféricos

Nome	Descriçao
CON	Console (teclado e vídeo)
AUX ou COM1	Primeira porta serial
COM2	Segunda porta serial
PRN ou LPT1	Primeira porta paralela
LPT2	Segunda porta paralela
LPT3	Terceira porta paralela
NUL	Periférico Inexistente

```
ofstream      oprn("PRN");
ofstream      ocom1("COM1");
ifstream      ilpt2("LPT2");
```

Namespace

- Serve para criar grupos lógicos;
- Evita duplicidade de declaração;
- As definições podem estar em arquivos diferentes;
- **std** é a namespace que define a biblioteca padrão;
- Declaração da Namespace:

```
namespace MyGroup
{
  // Definições
  void Draw()
  {
  }
}
```

- Namespace sem nome: escopo;
- Acessando a Namespace:

```
using namespace MyGroup;
Draw();
// ou
MyGroup::Draw();
```

Namespace

```
namespace RobotControler
{
class RobotSensor
{
//declarations and some implementation
};

class RobotDCMotors
{
        //declarations and some implementation
};

class RobotMovimentAlgorithm
{
        //declarations and some implementation
};
}
```

Namespace

```
using namespace std;
namespace RobotVision
{
    class RobotEye
    {
       //declarations and some implementation
        RobotControler::RobotSensor S1;
    };

    class RobotImageProcessing
    {
       //declarations and some implementation
    };
}
```

Templates

- Tipo do dado é desconhecido;
- Compilação: define o tipo;
- Pode assumir qualquer tipo;
- Declaração:
- Especificação:

```
String< char > S1;
String< int > C1;
String< double > D2;
```

```
Template < class Anyone >
class String
{
 private:
        struct Srep;
        Srep* rep;
 public:
        String();
        String(const Anyone*);
        String(const String&);

        Anyone read(int i) const;
        //...
};
```

Templates

```
template< class Anyone >
struct String< Anyone >::Srep
{
   Anyone* s; // pointer to elements
   int size;
//...
};
template< class Anyone >
Anyone String< Anyone >::read(int i) const
{
   return rep->s[i];
}
template< class Anyone >
String< Anyone >::String()
{
   rep = new Srep;
}
```

Templates

```
template< class That, int x >
class Buffer
{
  private
       That stack[x];
       int size;

  public:
       Buffer():size(x){}
       //...
};

Buffer< char, 100 > char_buff;
Buffer< AClass, 12 > aclass_buff;
```

Funções Templates

```
template< class ThisOne >
void sort( vector< ThisOne > & ); //declaration




void f( vector< int > & vi, vector< string > & vs)
{
   sort(vi); // sort(Vector<int>&)
   sort(vs); // sort(Vector<String>&)
}
```

Conteineres

- Classes para armazenamento e gerenciamento de dados;
- **Iteradores:** estrutura utilizada para acessar o conteúdo do conteiner;
- vector<T>: armazena na forma de um vetor com tamanho variável;
- deque<T>: vetor com tamanha variável e acesso aleatório;
- list<T>: conjunto (lista) de dados de um tipo com tamanho variável;
- set<T, compare>: suporta chaves únicas, com acesso rápido;
- multiset<T, compare>: suporta chaves duplicadas, com acesso rápido;
- map<T, compare>: suporta chaves únicas, com acesso rápido;
- multimap< T, compare>: suporta chaves duplicadas, com acesso rápido;

Conteineres

```
#include <iostream>
#include <vector>
using namespace std;
void pause()
{
   char lixo;
   cout << "\nPress any key to continue";
   cin >> lixo;
}
void printvector(vector<int> v)
{
   int i=0;
   cout << "Vector: ";
   for (i=0; i < v.size(); i++)
       cout << v[i] << " ";
   cout << "\n";
}
```

Conteineres

```
int main(int argc, char* argv[])
{
   cout << "Programa exemplo de contaneres\n”;
   cout << “\nvector<int> v(3,2)\n";

   vector< int > v(3,2);

   // Declara v como container vector do tipo int
   printvector(v);
   pause();

   cout << "\nv.push_back(5);
   // Insere o elemento 5 no final da sequencia\n";
   v.push_back(5);
   // Insere o elemento 5 no final da sequencia
   printvector(v);
   pause();
```

Conteineres

```
    cout << "\nInserindo mais elementos...\n";
    v.push_back(3);
    v.push_back(7);
    v.push_back(15);
    v.push_back(1);
    printvector(v);
    pause();

    cout << "\nv.pop_back(5);
    // Apaga o elemento do final da sequencia\n";
    v.pop_back();
    // Apaga o elemento do final da sequencia
    printvector(v);
    pause();
    return 0;
}
```

Tratamento de Exceções

- Exceção:
  - Disco Inválido;
  - Queda de Energia;
  - Divisão por Zero;
  - Ponteiro Inválido;
  - …
- Opções quando ocorre uma exceção:
  - Derrubar o programa;
  - Informar a ocorrência ao usuário e finalizar o programa;
  - Informar a ocorrência ao usuário, permitir que ele tente reparar o erro e continuar a execução do programa;
  - Executar uma ação corretiva automática e prosseguir sem avisar o usuário.
- Derrubar o programa: ocorre sempre que a exceção não é tratada;
- Entretanto, pode-se fazer melhor que isto.

Tratamento de Exceções

```
try      //Definindo a região sobre monitoramento
{
   Memory  MemObj()
   Função_Perigosa();
   throw MemObj;
}
//Tratando as exceções
catch(Memory a_oMem)  //SemMemoria
{
   //Ações para quando não há memória!
}
catch(File F) // ArquivoNaoEncontrado
{
    //Ações para quando o arquivo não é encontrado!
}
catch(...) //Opção Default
{
   //Ações para um erro inesperado!
}
```

Tratamento de Exceções

```
#include <iostream>
usind namespace std;

void S2iFunc( void );
class S2iTest
{
public:
   S2iTest(){};
   ~S2iTest(){};
   const char* ShowReason() const
                    { return "Exceção na classe S2iTest!";}
};
class S2iDemo
{
public:
    S2iDemo();
    ~S2iDemo();
};
```

Tratamento de Exceções

```
S2iDemo::S2iDemo()
{
    cout << "Construindo S2iDemo." << endl;
}
S2iDemo::~S2iDemo()
{
    cout << "Destruindo S2iDemo." << endl;
}
void S2iFunc()
{
    S2iDemo D; // cria um objeto da classe S2iDemo
    cout << "Em S2iFunc(). Throwing a exceção S2iTest."
         << endl;
    throw S2iTest(); //Envia S2iTest para o tratador!
}
```

Tratamento de Exceções

```
int main(int argc, char* argv[]) {
    cout << "Estamos em main." << endl;
    try   {
        cout << "No bloco try, chamando S2iFunc()." << endl;
        S2iFunc();
    }
    catch( S2iTest E ) //Recebe um objeto S2iTest
    {
        cout << "Estamos no  tratador catch." << endl;
        cout << "Tratando exceção do tipo S2iTest: ";
        cout << E.ShowReason() << endl;
    }
    catch( char *str ) {
        cout << "Trata outra exceção " << str << endl;
    }
    cout << "De volta a main. Fim." << endl;
    return 0;
}
```

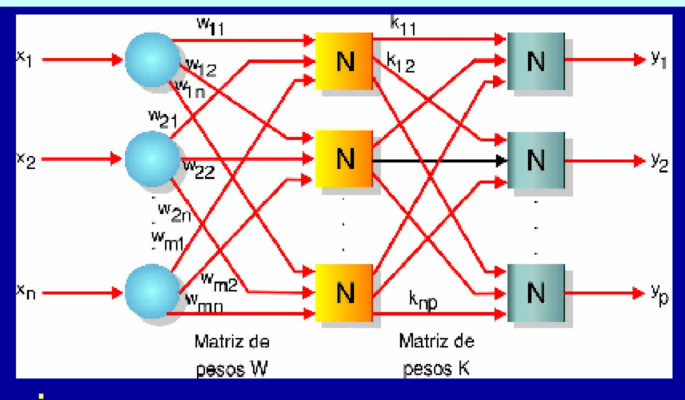
Tratamento de Exceções

- **terminate( ):** terminar o programa aplicativo;
- **set_terminate(MyTerminate):** redefine a função *terminate( )*, que será usada para finalizar o aplicativo em caso de erro grave;
- **unexpected( ):** função chamada quando uma exceção não é tratada. Se nada for definido, está chama *terminate( )*;
- **set_unexpected(MyUnexpected):** função utilizada para redefinir a função que trata execções não previstas.

Classe String

```
int main(int argc, char* argv[])
{
   char t_sBuffer[] = “O rato roeu a roupa do rei de roma!”;
   string Str = “”;
   string Name = “Janaina”;
   Str = t_sBuffer;
   Name += “ robou queijo”;
   Name.append(“ do vizinho”);
   if( Str.find(‘e’) >= 0 )
        cout << “\nLetra ‘e’ encontrada!”;

   Str[1] = ‘r’;
   char temp[] = Str.c_str();
   cout << “\n” << temp;
   cout << “\n” << Name;
   return 0;
}
```

Sistemas Industriais Inteligentes



- Parceria com a Pollux;
- Parceria com outras empresas;
- Parceria com o WZL/Alemanha;
- Projetos de Pesquisa pelo FINEP;
- Oportunidade de Pesquisas em Alta Tecnologia;
- Grupo Multidisciplinar e Internacional;
- Alto padrão de qualidade;
- Proposta: Centro de Excelência em Visão na América Latina;
- Quer saber mais?
- FALE COM A GENTE!

ORTH@DAS.UFSC.BR

HTTP://S2I.DAS.UFSC.BR

This document was created with Win2PDF available at http://www.daneprairie.com.
The unregistered version of Win2PDF is for evaluation or non-commercial use only.