BASE:

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

int main() //função fixa
{
    cout<<"salveeeee \n"; - (\n pode ser trocado por endl)
    return 0;
}
```

Cout: fluxo de saida padrão para o usuario

DECLARAÇÃO DE VARIAVEL:

```
int main() //função fixa
{
   int var1;
   int var2;
   var1 = 20;
   var2 = var1 +10;
   cout <<var2+50;
   return 0;
}
```

REDECLARAÇÃO DE VARIAVEIS:

```
int main()
{
  char c1 = 'A';
  char c2 = '\t'; - Volta para o inicio da linha

  cout << c1;
  cout << c2;
  c1 = 'B';
  cout << c1;
  cout << '\n'; - da um espaço na linha
  return 0;
}
= A     B
```

ENTRADA CIN:

```
int main() //função fixa
{
  int tempf;

  cout << "Digite a temperatura em Faherencheit";
  cin >> tempf;
  int tempe = (tempf-32)*5/9; - (expressao)
```

```
  cout << "A temperatura em Celsus é:" <<tempe<<'\n';
  return 0;
}
```
cin = ele pede um valor ao usuario para declarar na sua variavel

scapes em c++:(!!!!!!!)

TIPOS DE VARIAVEIS:
boo1: False x True
char: (-128) x (127)
short: (-32.768) x (32.767)
int: (-2.147.483.648) x (2.147.483.648)
long: (-2.147.483.648) x (2.147.483.648)
float: (3,4 x 10^-38) x (3,4 x 10^38)
double: (1,7 x 10^-308) x (1,7 x 10^308)

OPERADORES ARITIMETICO:
operadores: +, -, *, /, %(resto)
o. atribuição: +=, -=, *=, /=, %=
EXEMPLO:

```
int main()
{
  int op = 27;
  op += 10;
  cout << op <<" , ";
  op -= 7;
  cout << op <<" , ";
  op *= 2;
  cout << op <<" , ";
  op /= 3;
  cout << op <<" , ";
  op %= 3;
  cout << op << endl;
  return 0;
}
```
MANIPULADOR SETW: serve para expecificar as larguras e a formatação do cout
EXEMPLO:

```
#include<iostream>
#include<iomanip> // para o setw
using namespace std;

int main()
{
  long pop1 = 2617924, pop2 = 47, pop3 = 9761;

  cout << setw(9) << 'LOCATION' << setw(12)
       << 'POPULATION' << endl
       << setw(9) << 'Portcity' << setw(12) << pop1 << endl
```

```
      << setw(9) << 'Higjtown' << setw(12) << pop2 << endl
      << setw(9) << 'Lowville' << setw(12) << pop3 << endl;
  return 0;
} = vira uma tabelinha
```

FUNÇÕES DE BIBLIOTECAS:
Exemplo:
#include<iostream>
#include<cmath> - biblioteca de matematica
using namespace std;

5

//CICLOS DE REPETIÇÃO

-Operadores relacionais:
=Compara dois valores (igual =), (maior que >), (menor que <)

-Ciclo For
= (instrução inicial, instrução do teste, instrução de incremento)

```
int j;
 for (j=0; j<15; j++)
     cout << j * j << " ";
return 0; R= 0, 1, 4, 9, 16, 25
```

'Usa o for quando souber quantas vezes quer repetir o loop

-Ciclo While
= (instrução teste)
'usa quando não se sabe quantas vezes quer repetir o loop
e o criterio para ele parar é a instrução teste'

```
int n = 1; int p = 1;
 while (n < 100)
 {
    cout << setw(2) << p;
    cout << setw(5) << n << endl;
    ++ p;
    n = p * p * p;
 }
return 0;
```

-Ciclo Do while:
'No while se a intrução teste for falsa o programa
ja para, mas se tem o "Do" antes ele roda pelo
menos uma vez antes de uma possivel parada

ESTRUTURA DE DECISÃO
If = se ('se' 5 for maior que 2...)
Else = caso contrario
If else = e se

-----exemplo 1-----

```
#include<iostream>
#include<conio.h>     //para o getche()
using namespace std;

int main()
{
int chc = 0; //contador de letras
int wd = 1;  //contador de palavras
char ch;

cout << "Digite uma frase"; cin >> ch;

while ( (ch=getche()) != '\r')  //laço ate digitar ENTER
{
   if ( ch == ' ' )        //Se for um espaço
      wd++;                //conta uma palavra
   else                 //caso contrario
      chc++;            //conta um caracter
}
cout << "\nPalavras=" << wd << endl
    << "Letras" << chc << endl;  // mostra o resultado

return 0;
}
```

---exemplo 2---

```
#include<iostream>
#include<conio.h>
using namespace std;

int main()
{
char dir = 'a';
int x =10, y=10;

cout << "digite enter para sair \n";
while ( dir != '\r')
{
   cout << "\n sua localização é: " << x << ", " << y;
```

```
    cout << "\n Pressione a tecla (n, s, l, o): ";
    dir = getche();
    if (dir == 'n')
    y--;
    else if ( dir == 's')
    y++;
    else if ( dir == 'l')
    x++;
    else if (dir == 'o')
    x--;
}
return 0;
}
```

Switch: ao inves de colocar varios else if
usasse o case e o break no switch para criar
varias circustancias

Exemplo 2 com Switch:

```
dir = getche();
    switch(dir)
    {
        case 'n': y--; break;
        case 's': y++; break;
        case 'l': x--; break;
        case 'o': x++; break;
        case '\r': cout << "saida \n"; break;
        default: cout << "tente novamente \n";
        }
}
```

OPERADORES DE CONDIÇÃO:
? :

OPERADORES LOGICOS
&& = and (e)
|| = or (ou)
! = not (não)

ESTRUTURAS
- Uma estrutura é uma coleção de dados (variaveis)
- struct "nomeDaEstrutura" = declara uma estrutura
-Exemolo:

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

```
struct dat_nasc // declara uma estrutura
{
    int dia;
    int mes;
    float ano;
};

int main()
{
 dat_nasc data; //transforma a estrutura em uma variavel

cout << "Digite o dia do seu nascimento: ";
 cin >> data.dia; //agrega o valor inserido na variavel da estrutura
cout << "Digite o mes: ";
 cin >> data.mes;
cout << "Digite o ano: ";
 cin >> data.ano;

cout << "\n A SUA DATA DE NASCIMENTO É: "
<< data.dia << "/" << data.mes << "/" << data.ano;

return 0;
}
```

EXEMPLO 2:

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

struct peca
{
    int nModelo;
    int Npeca;
    float custo;
};

int main()
{
 peca p1;
 peca p2;
 int p;

 p1.nModelo = 10;
 p1.Npeca = 13;
 p1.custo = 50.5;
 p2.nModelo = 20;
 p2.Npeca = 22;
 p2.custo = 100.1;
```

```
 cout << " qual peça quer ver? "; cin >> p;

if (p == 1)
{
    cout << "Modelo: " << p1.nModelo;
    cout << "\nPeca: " << p1.Npeca;
    cout << "\nCusto: " << p1.custo;
} else
   cout << "Modelo: " << p2.nModelo;
   cout << "\nPeça: " << p2.Npeca;
   cout << "\nCusto: " << p2.custo;

return 0;
}
```

ESTRUTURA DENTRO DE OUTRA ESTRUTURA:

exemplo:

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

struct distancia
{
    int metros;
    float cms;
};

struct espaco
{
    distancia comp;
    distancia larg;

};


int main()
{
 espaco salaJantar;
 salaJantar.comp.metros = 5;
 salaJantar.comp.cms = 0.5;
 salaJantar.larg.metros = 3;
 salaJantar.larg.cms = 0.4;

 float c = salaJantar.comp.metros + salaJantar.comp.cms;
 float l = salaJantar.larg.metros + salaJantar.larg.cms;
```

```
cout << " A area da sala de jantar é: " << c * l;

return 0;

}
```

ENUMERAÇÃO
-uma lista finita de itens para seus valores
serem atribuidos a variaveis
-exemplo 1:

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

enum diasSemana{dom, seg, ter, qua, qui, sex, sab};

int main()
{
    diasSemana dia1;
    diasSemana dia2;
    int dif;

    dia1 = dom;
    dia2 = seg;
    dif = dia2 - dia1;

    cout << "A diferença entre os dias é: " << dif << endl;

    return 0;
}
```

exemplo 2:

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

enum ano{jan, fev, mar, abr, mai, jun, jul, ago, set, out, nov, dez};

int main()
{
   ano escolhaMes;
   escolhaMes = jul;
   cout << "O valor do mes escolhido é: " << escolhaMes;

   return 0;
}
```

FUNÇÕES

-um bloco de comandos que pode ser chamado por nome (que é declarado)
-exemplo:

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

void art()
{
    int j;
    for(j=0; j<45; j++) cout << '*';
    cout << endl;
}

int main()
{
  art();
  cout << "Tipo         preco" << endl;
  art();
  cout << "Prensado       5$" << endl;
  cout << "Colombia       20$" << endl;
  cout << "Hash           50$" << endl;
  art();
  return 0;
}
```

PASSAGEM DE ARGUMENTO PARA FUNÇÃO
-Cria uma função que deixa em aberto oque vai ser inserido, mantendo somente as regras que aquilo deve seguir.
-exemplo:

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

void artA(char , int);

void artA(char ch, int n)
{
    int j;
    for(j=0; j< n; j++) cout << ch;
    cout << endl;
}

int main()
{
  artA('/', 45);
  cout << "Tipo         preço" << endl;
  artA('=', 25);
```

```
  cout << "Prensado       5$" << endl;
  cout << "Colombia       20$" << endl;
  cout << "Hash           50$" << endl;
  artA('-', 45);
  return 0;
};
```

exemplo 2: PASSAGEM POR VALOR

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

void repetidor(char , int);

int main()
{
  char c;
  int nro;
  cout << "Digite um caractere: ";
  cin >> c;

  cout << "Digite o numero de vezes pra repetir: ";
  cin >> nro;
  repetidor(c, nro);
  return 0;
}

void repetidor(char ch, int n)
{
    int j;
    for(j=0; j<n; j++) cout << ch;
    cout << endl;
}
```

PASSAGEM DE ESTRUTURA COM ARGUMENTO DE FUNÇÃO:

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

struct distancia
{
    int metros; float cms;
};

void mostra(distancia);
```

```
int main()
{
  distancia d1;
  distancia d2;
  cout << "Digite a d em metros: "; cin >> d1.metros;
  cout << "Digite a d em cms: "; cin >> d1.cms;

  cout << "Digite a d em metros: "; cin >> d2.metros;
  cout << "Digite a d em cms: "; cin >> d2.cms;

  cout << "\n d1= ";
  mostra(d1);
  cout << "\n d2= ";
  mostra(d2);
  return 0;
}



void mostra(distancia dx)
{
    cout << dx.metros << " / " << dx.cms << "/";
}
```

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

float covertePeso(float);

int main()
{
    float libras, kilos;
    cout << "digite seu peso em libras: "; cin >> libras;
    kilos = covertePeso(libras);
    cout << "seu pelo em kilo é: " << kilos << endl;
    return 0;


}
float covertePeso(float aux)
{
    float kilogramas = 0.453592 * aux;
    return kilogramas;
}
```

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

struct distancia
{
   int metros;
   float cms;

};

distancia addSt(distancia, distancia);
void exibe(distancia);

int main()
{
  distancia d1, d2, d3;
  cout << "\n 1 Digite metros: "; cin >> d1.metros;
  cout << "\n 1 Digite cms: "; cin >> d1.cms;

  cout << "\n 2 Digite metros: "; cin >> d2.metros;
  cout << "\n 2 Digite cms: "; cin >> d2.cms;

  d3= addSt(d1, d2); cout << endl;
  exibe(d1); cout << " + ";
  exibe(d2); cout << " =";
  exibe(d3); cout << endl;

  return 0;

}
```

PASSAGEM DE ARGUMENTOS POR REFERENCIAS
- sao valores atribuidos a variaveis que nao irao
se alterar, usa o & para especificar isso

exemplo:

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

void intFrac(float, float&, float&);

int main()
{
  float nro, intPart, fracPart;
do{
   cout << "\n Digite um nro real: "; cin >> nro;
```

```
    intFrac(nro, intPart, fracPart);
    cout << "Parte inteira é: " << intPart;
    cout << ", parte fracionaria é: " << fracPart << endl;
} while(nro != 0);
return 0;
}

void intFrac(float n, float& intp, float& fracp)
{
    long temp = static_cast<long>(n); //função que separa a parte inteira
    intp = static_cast<float>(temp); //função que separa a parte fracionada
    fracp = n - intp; //função que mostra o valor fracionada separado
}
```

exemplo 2 com estruturas, função e passagem de argumentos por referencia

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

struct distancia
{
    int metros;
    float cms;
};

void escala(distancia&, float);
void exibe(distancia);

int main()
{
    distancia d1 = {12, 6};
    distancia d2 = {10, 5};
    cout << "d1= "; exibe(d1);
    cout << "\n d2= "; exibe(d2);

    escala(d1, 0.5);
    escala(d2, 0.25);

    cout << "\n nd1 = "; exibe(d1);
    cout << "nd2 = "; exibe(d2);
    return 0;
}

void escala (distancia& dd, float fator)
{
    float cms = (dd.metros*100 + dd.cms)*fator;
    dd.metros = static_cast<int>(cms/100);
    dd.cms = cms - dd.metros*100;
}
```

```
void exibe(distancia dd)
{
   cout << dd.metros << "--" << dd.cms << "-";
}
```

Orientação a objeto

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

class pequenoonj     // declara a classe
{
   private:         // dados que ficam guardados
     int algunsdados;
   public:         // funções acessiveis ao usuarios
     void setdados(int d)
   {algunsdados = d; }
     void mostrardados()
   {cout << "os dados sao: " << algunsdados << endl;}
};

void main()
{
   pequenoonj s1, s2;     //declara 2 objetos com a classe

   s1.setdados(1066);    //puxa uma ação da classe
   s2.setdados(1760);

   s1.mostrardados();
   s2.mostrardados();


}
```

---

exemplo 2

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

class peca    // declara a classe
{
   private:         // dados que ficam guardados
     int nmodelo;
     int npeca;
     float custo;
   public:         // dados/ações acessiveis ao usuarios
```

```
    void setdados(int nm, int np, float c)
   {
      nmodelo = nm;
      npeca = np;
      custo = c;
   }
    void mostrardados()
   {
      cout << " modelo: " << nmodelo;
      cout << "peca: " << npeca;
      cout << "custo R$:" << custo;



    }
};

int main()
{
   peca p1;     //declara 2 objetos com a classe

   p1.setdados(6, 73, 88.99);
   p1.mostrardados();
   return 0;

}
```

Construtores:

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

class contador
{
   private:
     unsigned int cont;
   public:
     contador(): cont(0)
       {cout << "vazio ";}
     void incCont()
       {cont++;}
     int obtemCont()
       {return cont;}
};

int main()
{
   contador c1, c2;
```

```
    cout << "c1 = " << c1.obtemCont();
    cout << "c2 = " << c2.obtemCont();

    c1.incCont();
    c2.incCont();
    c2.incCont();

    cout << "c1 = " << c1.obtemCont();
    cout << "c2 = " << c2.obtemCont();


    return 0;
}
```

Obejto com argumentos de função

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

class distancia
{
    private:
      int metros;
      int cms;
    public:
      distancia (): metros(0), cms(0)
      {cout << "inicio do construtor\n";}
      distancia(int mt, int cm)
      {
          metros = mt;
          cms = cm;
      }
      void obtemDist()
      {
        cout << "\nDigite metros: "; cin >> metros;
        cout << "\nDigite centimetros: "; cin >> cms;
      }
      void mostradist()
      {
        cout << metros << "\'" << cms << "\'";
      }

      void addDist(distancia , distancia);
};

void distancia :: addDist(distancia d2, distancia d3)
```

```
{
    cms = d2.cms + d3.cms;
    metros = 0;
    if(cms>=100.0)
    {
        cms -= 100.0;
        metros++;
    }
    metros += d2.metros + d3.metros;
}

int main()
{
    distancia dist1, dist3;
    distancia dist2(10, 5.50);

    dist1.obtemDist();
    dist3.addDist(dist1, dist2);

    cout << "\n dist1 = "; dist1.mostradist();
    cout << "\n dist2 = "; dist2.mostradist();
    cout << "\n dist3 = "; dist3.mostradist();
    return 0;


}
```

-VETORES-
-matrizes:

```
int main()
{
  int idade [4];

  for(int j=0; j<4; j++)
  {
    cout << "Digite uma idade: ";
    cin >> idade[j];

  }
  for (int j=0; j<4; j++)
  {
    cout << "idade digitada: " << idade[j] << endl;
  }
  return 0;
}
```

Obs: precisa do ciclo 'for' para preencher e ler matrizes
Exemplo 2:

```
int main()
{
 int tamanho = 5;
 double total = 0, media;
 double vendas [tamanho];

 cout <<"Insira as vendas para os 5 dias\n ";
 for(int j=0; j< tamanho; j++)
 {
  cin >> vendas[j];
  total+=vendas[j];
 };

 media = total/tamanho;
   cout << "A media é: " << media << endl;

return 0;
}
```

Exemplo 3:

```
int main()
{
 int dia, mes, total_dias;
 const int mesAno = 12;

 int diasPorMes [mesAno] = { 31, 28, 31, 30, 31, 30,
31, 30, 31, 30, 31, 30 };

cout << "\n Digite o mes (1 a 12): ";
cin >> mes;
cout << "\n Digite o dia (1 a 31): ";
cin >> dia;

total_dias = dia;

for(int j=0; j< mes-1; j++)
{
  total_dias+=diasPorMes[j];
  cout << "O total de dias do inicio do ano é: " << total_dias << endl;
}
return 0;
}
```

MATRIX BIODIMENSIONAL:

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

```
const int ESTADOS = 4;
const int MESES = 3;

int main()
{
  int e, m;
  double vendas [ESTADOS] [MESES];

  cout << endl;

for(e=0; e<ESTADOS; e++) // inserir dados
{
  for(m=0; m<MESES; m++)
    {
      cout << "\n Digite as vendas por estado  " << e+1;
      cout <<", mes" << m+1 << ": ";
      cin >> vendas [e] [m];
    }
} // formatar tabeela
  cout << "\n\n";
  cout << "                    MES\n";
  cout << "                1    2    3";

for(e=0; e< ESTADOS; e++) //mostrar os dados na tabela
{
  cout << "\nEstado" << e+1;
    for(m=0; m<MESES; m++)
      {
      cout << setiosflags(ios::fixed)
           << setiosflags(ios::showpoint)
           << setprecision(2)
           << setw(10)
           << vendas[e][m];
       }
}
return 0;
}
```

Exemplo 2: Passagem de matriz como argumento de função

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


const int ESTADOS = 4;
const int MESES = 3;
void mostra(double [ESTADOS][MESES]);
```

```
int main()
{
  int e, m;
  double vendas [ESTADOS] [MESES];

  cout << endl;

for(e=0; e<ESTADOS; e++) // inserir dados
{
  for(m=0; m<MESES; m++)
    {
      cout << "\n Digite as vendas por estado  " << e+1;
      cout <<", mes" << m+1 << ": ";
      cin >> vendas [e] [m];
    }
}
mostra(vendas);

return 0;
}

void mostra(double funVendas[ESTADOS][MESES])
{
  int e, m;
    cout << "\n\n";
  cout << "                    MES\n";
  cout << "               1    2    3";

for(e=0; e< ESTADOS; e++) //mostrar os dados na tabela
{
  cout << "\nEstado" << e+1;
    for(m=0; m<MESES; m++)
      {
      cout << setiosflags(ios::fixed)
           << setiosflags(ios::showpoint)
           << setprecision(2)
           << setw(10)
           << funVendas[e][m];
       }
}
}
```

MATRIZES COMO DADOS DE MEMBROS DE CLASSE

-Criando uma pilha:

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

```
class Pilha
{
  private:
    enum {MAX = 10};
    int st[MAX];
    int top;
  public:
    Pilha()
    {top = 0;}

    void push(int var)
    {st[++top] = var;}

    int pop()
    {return st[top--];}
};

int main()
{
  Pilha p1;
  p1.push(11);
  p1.push(22);
  cout << "1: " << p1.pop() << endl;
  cout << "2: " << p1.pop() << endl;
  p1.push(33);
  p1.push(44);
  p1.push(55);
  p1.push(66);
  cout << "3: " << p1.pop() << endl;
  cout << "4: " << p1.pop() << endl;
  cout << "5: " << p1.pop() << endl;
  cout << "6: " << p1.pop() << endl;

return 0;
}
```

Exemplo de matriz no objeto:

```
class distancia
{
  private:
    int metros;
    float cms;

  public:
    void obetemDist()
    {
      cout << "\n Digite metros: "; cin >> metros;
```

```
      cout << "\n Digite cms: "; cin >> cms;
    }
    void mostraDist()
    {
      cout << metros << "\'-" << cms << "\'-" << endl;
    }
};

int main()
{
  distancia dist [20];
  int n = 0;
  char resp;

  do{
    cout << "Digite o nro da distancia: " << n+1;
    dist[n++].obetemDist(); //indice sendo gravado

    cout << " Digitar outra distancia? (y/n) ";
    cin >> resp;
  }while((resp != 'n'));

  for(int j=0; j<n; j++)
  {
    cout << "\n Nro da distancia" << j+1 << "é: ";
    dist[j].mostraDist();  //indice sendo puxado para mostrar
  }
return 0;
}
```

Classe de string padrao:

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


int main()
{
 const int MAX = 40;
 char str[MAX];

 cout << "Digite uma palavra ";
 cin >> str;


 cout << "\nVoce digitou: " << str;

 return 0;
```

}
obs: Por mais que esteja estipulado que o maximo é 40 caracteres
o software printa tudo que for digitado independente
para resolver isso é so inserir essa linha dessa forma:
- cin >> setw(MAX) >> str; -
obs2: esse codigo nao aceita espaços,
para ele aceitar devemos puxar da biblioteca do cin o seguinte:
- cin.get(str, MAX); -
obs3: para copiar uma declaração de string para outra, usasse:
- strcpy(str2, str1) - // agora a str2 tem o mesmo elemento que str1

String em matriz:

```
int main()
{
 const int DIAS = 7;
 const int MAX = 18;

 char str [DIAS] [MAX] = {"Segunda", "Terça", "Quarta", "Quinta", "Sexta", "Sabado",
"Domingo"};

 for(int j=0; j<DIAS; j++) cout << str[j] << endl;


return 0;

}
```

String como membro de classe:

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

class Peca{
  private:
    char pecaNome[40];
    int pNumero;
    double custo;
  public:
    void definePeca(char pNome[], int pn, double c)
    {
      strcpy(pecaNome, pNome);
      pNumero = pn;
      custo = c;
    }
    void mostraPeca()
    {
      cout << "\nNome" << pecaNome;
```

```
      cout << "\nNumero" << pNumero;
      cout << "\nCusto = $" << custo;
    }
};

int main()
{
  Peca p1, p2;

  p1.definePeca("Parafuso", 4, 5.99);
  p2.definePeca("Martelo", 8, 49.99);

  cout << "\n Primeira parte: "; p1.mostraPeca();
  cout << "\n Segunda parte: " ; p2.mostraPeca();
  cout << endl;
  return 0;
}
```

Atribuindo objetos de string:

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

int main()
{
  string s1("Carro");
  string s2 = "bonito";
  string s3;

  s3 = s1;
  cout << "s3 = " << s3;

  s3 = "nem " + s2 + " nem feio";
  cout << "\n S3 agora e: " << s3;

  s1.swap(s2);   //troca s1 por s2
  cout << "\n novo s1= " << s1;
}
```

obs: pode usar o operador += para anexar uma string ao final de uma string existente

---

Ferramentas de atribuição:

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

```
int main()
{
  string nome, vulgo, quebrada;
  string salve("Slvv");

  cout << "Digite seu nome:";
  getline(cin, nome);            //abre para o usuario digitar o nome
  cout << "seu nome e: " << nome;

  cout << "Digite seu vulgo:"; cin >> vulgo;
  salve += vulgo;               // soma a string na variavel salve ao oq foi inserido em vulgo
  cout << salve;

  cout << "Digite onde vc mora: \n";
  getline(cin, quebrada, '$');     // pode escrever varias linhas usando o enter, ate digitar $ no
fim
  cout << "Sua quebrada e: " << quebrada;

  return 0;
}
```

Encontrar objetos de string:

```
  int n;
  string s1 = "aaaaa quero laricar e fumar um beck";
  n = s1.find("fumar");
  cout << "encontrado fumar em: " << n; // r= 22

  n = s1.find_first_of("la");
  cout << "\nprimeiro la: " << n;   // r= 0 (a no começo)

  n= s1.find_first_not_of("aeiou");
  cout << "\nPrimeira consoante é: " << n; // r= 5
```

Modificar objetos de strings:

erase() = remove uma substring de uma string. Seu primeiro argumento
é a posição do primeiro caractere na substring e o segundo é o comprimento da
substring

insert() = insere uma string especificada. Seu primeiro argumento a posição
e o segundo argumento oque voce quer inserir.

replace() = substitui. Seu primeiro argumento faz igual ao erase e o segundo é
o argumento novo que você quer substituir

Herança
-A herança fornece a ideia de reutilização, ou seja, o código, uma vez escrito, pode ser usado repetidamente em várias novas classes.
-Herança é o processo de criação de novas classes, chamadas classes derivadas, a partir de classes existentes ou de base
- Formas de chamar:
  - Base = classe super, classe de base ou classe pai.
  - Derivada = subclasse, classe derivada ou classe filha.
- Tipos de Heranças
  - Herança unico nivel
  - Herança mutavel
  - Herança multipla
  - Herança hieranquica
  - Herança hibrida

Exemolo:

```
class baseA
{
   public:
   void print(){
      cout << "classe da variavel A";
   }
};

class derivadaA : public baseA   //classse tipo unico
{};

class derivadaB : public derivadaA  // Herança multinivel
{};

class derivadaMista : public baseA, public derivadaB   // Herança multipla
{};

class d1 : public baseA {}; class d2 : public baseA {}; class d3 : public baseA {};  // Herança hierarquica

class derivadaC : public derivadaA, derivadaB {};  // Herança hibrida

void main()
{
   derivadaB obj1;
   obj1.print();
}
```

- PONTEIROS -

Mostra no seu programa o endereço daquela variavel declarada (pode ser numeros inteiros, floats e etc,
pode ser tambem objetos e strings)

Exemplo1:

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

int main()
{
   int var1 = 33;
   float var2 = 22.33;

   cout << &var1 << endl;    // 0x61ff0c (local na memoria da variavel var1)
   cout << &var2 << endl;    // 0x61ff08
   cout << var1 << endl;     // 33
   cout << var2 << endl;     // 22.33

   return 0;
}
```

Exemplo 2: criando uma variavel de ponteiros para descobrir valores de variaveis

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

int main()
{
   int var1 = 33;
   int var2 = 22;

   int* ptr;

   ptr = &var1;


   cout << *ptr << endl;    //33
   return 0;
}
```

- Um "int*" so pode apontar para uma variavel do tipo int
- void* pode apontar para todo tipo de dado

- PONTEIROS E MATRIX

//exemplo 1:

```
int main()
{
   int intV[4] = {11, 22, 33, 44};

   for(int i=0; i<4; i++)
   {
      cout << intV+i << endl;         // Mostra os  4 endereços das variaveis
      cout << *(intV+i) << endl;      // Mostra as  4 variaveis
      cout << intV << endl;           // Mostra 1 endereço de variavel
      cout << *(intV) << endl;        // Mostra 1 valor de variavel
   }
   return 0;
}
```

//exemplo 2:
ponteiro como uma variavel para ser incrementada

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

int main()
{
   int intVetor[3] = {33, 44, 55};
   int *ptrInt;
   ptrInt = intVetor;

   for(int i=0; i<3; i++)
   {
      cout << *(ptrInt++) << endl;     // Mostra os 3 valores das variaveis
      cout << (ptrInt++) << endl;     // Mostra o endereço das 3 variaveis
   }
   return 0;
}
```

PONTEIROS E FUNÇÃO
3 tipos de forma de passar argumentos para função:
- Por valor
- Por referencia
- Por ponteiro
    - Passar por ponteiro vai ser util quando uma função tiver como objetivo modificar variáveis no programa chamador, essas variáveis não podem ser passadas por valor.

Exemplo 1: Por referencia

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

int main()
```

```
{
   void centralizar(double&);
   double var = 10;
   cout << " Var = " << var << " polegadas" << endl;    //  "Var = 10 polegadas"

   centralizar(var);
   cout << "Var = " << var << " centrimetros" << endl;  //  "Var = 50 polegadas"

   return 0;
}

void centralizar(double &v)
{
   v*= 5;    // multiplica o conteudo do q for colocado como argumento na função
}
```

Exemplo 2: Por ponteiro

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

int main()
{
   void centralizar(double*);
   double var =10;
   cout << " Var = " << var << " polegadas" << endl;  //  "Var = 10 polegadas"

   centralizar(&var);
   cout << " Var = " << var << " centrimetros" << endl;  //  "Var = 10 polegadas"

   return 0;
}

void centralizar(double *v)
{
   *v *= 5;
}
```

//exemplo 3:
Aplicando uma função em um vetor de 5 elementos por meio de ponteiros:

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


int main()
{
  void centralizar(double *);
  double array[5] = {10, 20, 30, 40, 50};
```

```
   centralizar(array);
   for(int i=0;i<5;i++)
   {
    cout << "vetor array [" << i <<"]= ";
    cout << array[i] << "cms" << endl;
   }

}
void centralizar(double *ptr)
{
  for(int j=0; j<5;j++){
    *ptr++ *=2.54;     //incrimento para acessar todos os argumentos do vetor
  }
}
```

SAIDA:

```
vetor array [0]= 25.4cms
vetor array [1]= 50.8cms
vetor array [2]= 76.2cms
vetor array [3]= 101.6cms
vetor array [4]= 127cms
```

Exemplo 4:

Declarar uma string com ponteiro permite você acessar mais facilmente cade elemento da string

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


int main()
{
    char str[] = "maryjuana is my love";
    char* strP = "colombia o sativo";

    cout << str << endl;
    cout << strP << endl;

    str++      //nao pode
    strP++;    //começa a string pulando o 1* elemento
    strP++;    //pula o 2* argumento
    strP++;
    strP++;
    cout << strP << endl;  // SAIDA: "mbia o sativo"

    return 0;
}
```

Exemplo 5: mostrar uma frase por meio de função com ponteiros

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


int main()
{
  void mostrar(char*);
  char str[] = "Viva o carpie diem";

  mostrar(str);

return 0;

}

void mostrar(char* p)
{
  while(*p) {
    cout << *p++;
    cout << endl;
  }
}
```

GERENCIAMENTO DE MEMORIA
-New: Cria um espaço na memoria, muitas vezes usados para se definir um tamanho para oq sera
guardado no vetor, então criamos um ponteiro com o tamanho ideal
-Delete: Apos criar um espaço na memoria é essencial para muitos processos apagalos dps apos seu uso

Exemplo de uso:

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

int main()
{
  char* str = "Um dia frio um bom lugar pra ler um livro";
  int len = strlen(str);

  char* p;
  p = new char[len+1];

  strcpy(p, str);

  cout << "ponteiro = " << p << endl;
```

```
  delete[] p;

  return 0;
}
```

PONTEIROS PARA OBJETOS
-Criar um ponteiro para uma classe:
  - "distancia* ptr"
-Gravar um endereço de um objeto em um ponteiro (criado anterioamente na classe do obj):
  - "distancia d; ptr = &d;"
-Acessando as funções da clase pelo ponteiro:
  - "(*ptr).entrada();" ou "ptr -> entrada();"
-Criando objetos dinamicamente para armazer seu endereço num ponteiro:
  - "Distancia* ptr = new Distancia;
-Criando um ponteiro para uma matrix de objetos
  - "Distancia d[5]; Distancia* ptr = d;

Exemplo: Criando um ponteiro do objeto que executa suas funções

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

  class Demo
  {
    public:

      void indica()
      {
       cout << "Maryjuana indica e a melhor!";
      }

      void sativa()
      {
       cout << "Maryjuana sativa e melhor.";
      }
  };

int main()
{
  Demo d;
  Demo* ptr;              // podia ser simplificado assim: Demo* ptr = &d;
  ptr = &d;              // &d = ptr E d = *ptr

  ptr ->indica();   // string completa
  ptr ->sativa();   // string completa

  //formas possiveis de usar as funções
```

```
  d.indica();
  (&d)->indica();
  ptr ->indica();
  (*ptr).indica();

  return 0;
}
```

Exemplo 2:

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


  class Demo
  {
     int dx, dy;
    public:
       Demo(int x, int y)
       {
        dx = x;
        dy = y;
       }

       void mostra()
       {
        cout << "dx = " << dx << endl;
        cout << "dy = " << dy << endl;
       }
  };

int main()
{
  Demo* ptr = new Demo(10, 20);
  (*ptr).mostra();

  return 0;
}
```

exemplo 3:

```
#include <iostream>
#include <string.h>
using namespace std;
#define S 2
```

```
  class Abstrata
  {
     char nome[40];
    public:
       Abstrata(){};
       Abstrata(char* s)
       {
        strcpy(nome, s);
       }

       void mostra()
       {
        cout << "nome = " << nome << endl;
       }

  };

int main()
{

  Abstrata* ptr = new Abstrata[S];
  Abstrata* temp = ptr; char s[4];

  for(int i=0; i<S; i++ )
  {
    cout << "Digite seu nome: " << endl;
    cin.getline(s, 40);
    ptr[i] = Abstrata(s);
  };

  cout << "Os nome são: " << endl;

  for(int i=0; i<S;i++)
  {
    temp++->mostra();
    delete[]ptr;
  }

  return 0;
}
```