C++

Criando a primeira classe:

```
#include <iostream>

/* run this program using the co

int main() {
    printf("Hello word \n");
    system("pause");
    return 0;
}

```

Essa é a estrutura do básica devemos clicar em compilar e executar na IDE.

```
#include <iostream>

/* run this program using the console

int main() {
    std::cout << "Ola mundo \n\n";
    system("pause");
    return 0;
}


```

Podemos utilizar dessa forma também onde o **cout** é um método presente em **std** e atribuímos o valor “Ola mundo \n\n” através do operador logico **<<**.

Vale lembrar que importamos a classe **iostream** através do comendo **#include** dessa forma essa classe é importada antes do programa ser executado.

Diretiva #define ->Essa diretiva é usada para criar constantes no código:

```
#define pi 3.14
```

Dentro do **std** temos outros métodos de interação com o usuário:

```
extern istream cin;       /// Linked to standard input
extern ostream cout;      /// Linked to standard output
extern ostream cerr;      /// Linked to standard error (unbuffered)
extern ostream clog;      /// Linked to standard error (buffered)
```

Então para pegarmos valores de entrada no console, usaríamos o método **cin**

```
int main() {
    int numero;
    std::cout << "Escreva um numero \n";
    std::cin >> numero;

    std::cout << numero;
    system("pause");
    return 0;
}
```

Podemos substituir o \n por **std::endl** :

```
std::cout << "Escreva um numero "<<std::endl;
```

Terá o mesmo efeito, porém podemos refatorar o código de forma que não seja necessário toda vez declarar std::email:

```
#include <iostream>
using std::cout;
using std::endl;
using std::cin;
#define pi 3.14
/* run this program using the console pauser

int main() {
    int numero;
    cout << "Escreva um numero "<<endl;
    cin >> numero;

    cout << numero;
    system("pause");
    return 0;
}
```

Temos o operador **std::setw** também, esse operador define o tamanho ocupado no terminal exemplo:

```cpp
#include <iostream>
using std::cout;
using std::endl;
using std::cin;
#include <iomanip>
using std::setw;
#define pi 3.14
/* run this program using the console pauser or add y

int main() {
    int numero;
    cout << "Escreva um numero "<<setw(10)<<endl;
    cin >> numero;

    cout << numero;
    system("pause");
    return 0;
}
```

Constantes:

Constantes são variáveis que não alteram seu valor durante a execução:

```cpp
int numero;
const int numero2=5;
```

Tipos de variáveis:

Tabela 2.1

Tipo de dado	Intervalo numérico		Bytes de memória
	Inferior	Superior	
Char	-128	127	1
Short	-32.768	32.767	2
Int	-2.147.483.648	2.147.483.647	4
Long	-2.147.483.648	2.147.483.647	4
Float	$-1.2*10^{-38}$	$3.4*10^{38}$	4
Double	$-2.2*10^{-308}$	$1.7*10^{308}$	8

Estrutura de decisão:

```
if(numero<5){
    cout << "o numero é menor que 5";
}
else if(numero == 8){
    cout << "o numero é 8";
}
else{
    cout << "Não sei";
}
```

Mesma coisa....

Estrutura de repetição:

```
for(int x=0; x<10; x++){
    cout << x;
}
while(numero2<20){
    numero2++;
    cout << numero2;
}
do{
numero2++;
    cout << numero2;
}while(numero2<20);
```

MESMA COISA .....

Estrutura:

```
#include <string>
using std::string;

struct Pessoa{
    string name;
    int cpf;
    int idade;
};
/* run this program using the consol

int main() {
    Pessoa pessoa;
    pessoa.name="gabriel";
    pessoa.idade=21;
    pessoa.cpf=1561561;
    cout<<pessoa.name;
    system("pause");
    return 0;
}
```

É uma forma simples de usar um conjunto de variáveis, similar a um objeto.

Podemos colocar estruturas dentro de estruturas:

```
struct Pessoa{
    struct Apelidos{
        string apelido1;
    };
    Apelidos apelidos;
    string name;
    int cpf;
    int idade;
};
```

```
pessoa.apelidos.apelido1="biruleibi";
cout<<pessoa.apelidos.apelido1;
```

Enum:

```
enum diasDaSemana { Seg, Ter, Qua, Qui, Sex, Sab, Dom };
diasDaSemana dia1, dia2;
dia1 = Seg;
dia2 = Qua;
```

Diferenças estruturas e classes:

As classes possuem propriedades de classes:

```
struct exemplo {
  int a;        // dados publicos (por default)
  int b;
  float c;
};]


class exemplo {
  private int a;      // dados privados (por default)
  private int b;
  private int c;
};
```

Funções:

Possui programação funcional:

```
int soma(int x, int y){
    return x+y;
}
```

```
cout<<soma(5,8);
```

Classe:

```
class minhaClasse // especificacao de classe
{
  private:
     int x;
   public:
     void setValor(int d) // funcao membro
.    { x = d; }
.    void mostraValor() // funcao membro
.    { cout << "\nValor = " << x << endl; }
. };
```

Construtor:

```
class NomeClasse
{
     Public:
        NomeClasse(); // construtor default
        NomeClasse(NomeClasse& x); // construtor copia
        NomeClasse(<lista de parâmetros>); // outro construtor
};
```

Destrutor:

```
class Exemplo
  {
  private:
        int dado1
  public:
        Exemplo() { dado1 = 0; } // construtor
        ~Exemplo() { } // destrutor
 }
```

Herança:

A herança no c++ é simples e funciona através do atributo **:**

```
struct Pessoa{
    struct Apelidos{
        string apelido1;
    };
    Apelidos apelidos;
    string name;
    int cpf;
    int idade;
};

struct aluno: Pessoa{
    int matricula;
};
```

```
Aluno.name="gabriel"
```

Herança múltipla:

```
class Z: public X, public Y
```

Ponteiros:

```
int x1 = 10; // declara e inicializa a variável x1
int *x1Ptr; // declara x1Ptr como variável ponteiro
x1Ptr = &x1; //atribui o conteúdo de x1 a x1Ptr
cout << *x1Ptr; // exibe conteúdo de x1(10), que é apontado por x1Ptr
```

Classes em diferentes arquivos:

Teste.h

```
#ifndef TESTE_H
#define TESTE_H

class Teste
{
    public:
        int x;
        Teste();
};

#endif
```

Teste.cpp

```
#include "Teste.h"
#include <iostream>
    Teste::Teste(){

    }
```

main.cpp

```
#include "Teste.h"
```

```
Teste objeto;
objeto.x=3;
cout << objeto.x;
```

Tipo genérico:

```
generic <typename T>

ref class GenericType {};
ref class ReferenceType {};

value struct ValueType {};

int main() {
    GenericType<ReferenceType^> x;
    GenericType<ValueType> y;
}
```