Trabalho de L.L.P2 e L.P2

Nome: Carlos Daniel Bernardo Silva, Leticya de Souza Arantes e Miguel Rodrigues Silva
Turma: 2º Informática
Grupo: 2

Conjuntos de Bibliotecas

Conjunto de Bibliotecas 1 – Bibliotecas tipo C:

1. <cassert> : A biblioteca assert tem a finalidade de disponibilizar uma função nomeada assert(), cujo objetivo é verificar suposições falsas feitas pelo programa e, caso haja falsificações, ela imprime uma mensagem de diagnóstico.
O código a seguir é um exemplo de comando utilizando esta biblioteca:

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

int main(int argc, char const *argv[])
{
    assert(1 == 1);
    return 0;
}
```

2. <cctype> : Esta biblioteca trabalha com a manipulação de dados, sendo geralmente usada por programadores que buscam o envolvimento de diversos idiomas, alfabetos e sistemas numéricos, buscando disponibilizar um meio mais prático e maleável para o uso de determinados itens.
O código a seguir é um exemplo de comando utilizando esta biblioteca:

```
#include <stdio.h>
#include <ctype.h>
int main(){
char c;
int resultado;
c = '1';
resultado = isalnum(c);
printf("resultado: %d\n", resultado);
```

3. <cerrno> : A biblioteca errno define a variável inteira **errno** , que é definida por chamadas de sistema e algumas funções da biblioteca em caso de erro para indicar o que ocorreu. Esta macro se expande para um lvalue modificável do tipo int, portanto, pode ser lida e modificada por um programa.
Aqui está um exemplo do uso desta biblioteca:

```
#include <stdio.h>
#include <errno.h>
```

```c
#include <string.h>

int main() {
   FILE *file = fopen("arquivo_inexistente.txt", "r");

   if (file == NULL) {
        perror("Erro ao abrir o ficheiro");

        char *errorMessage = strerror(errno);
       fprintf(stderr, "Erro específico: %s\n", errorMessage);
   } else {
       fclose(file);
   }

   return 0;
}
```

4. <cfenv> : A biblioteca `<fenv.h>` na linguagem C é um ficheiro de cabeçalho que define macros e funções para gerir o ambiente de ponto flutuante, incluindo o controlo de exceções (como desbordamento ou divisão por zero) e modos de arredondamento. Para a utilizar no seu código, deve incluir a linha `#include <fenv.h>` no início do seu ficheiro C.

Aqui está um exemplo de uso desta biblioteca:

```c
#include <stdio.h>
#include <fenv.h>

int main() {
   float num1 = 1.0f;
   float num2 = 0.0f;
   float resultado;

    resultado = num1 / num2;

   if (fetestexcept(FE_DIVBYZERO)) {
       printf("Erro: Divisão por zero ocorreu.\n");
   }

   return 0;
}
```

5. <cfloat> : O arquivo de cabeçalhos **<float.h>** fornece alguns macros de números de pontos flutuantes do tipo ‘float’ para o sistema específico e a implementação para o compilador utilizado.

Exemplo de uso:

```c
#include <stdio.h>
#include <float.h>

int main() {
   printf("O valor máximo de um float é: %g\n", FLT_MAX);
```

```
    printf("O valor mínimo de um float é: %g\n", -FLT_MAX);
    printf("O número de casas decimais em um float (precisão) é: %d\n",
FLT_DIG);
    return 0;
}
```

6. <cinttypes> : O arquivo **inttypes.h** é um arquivo de cabeçalho C que faz parte da biblioteca e API padrão C. Ele inclui o cabeçalho stdint.h e define diversos macros para uso com as famílias de funções printf e scanf , bem como funções para trabalhar com o `intmax_t`tipo.

Exemplo de uso:

```
include <inttypes.h>
#include <stdio.h>

int main() {
    intmax_t num = 123456789012345LL;

    printf("O valor de num é: %" PRIdMAX "\n", num);

    return 0;
}
```

7. <ciso646> : A biblioteca <iso646.h> na linguagem C define macros que fornecem nomes alternativos para operadores lógicos e de bitwise, como and, or, not, bitand, bitor, compl, xor, e seus equivalentes and_eq, or_eq, xor_eq. O objetivo é permitir a programação em C mesmo em teclados que não possuem os símbolos tradicionais desses operadores (como os que não são QWERTY) ou para aumentar a legibilidade do código.

Exemplo de uso:

```
#include <stdio.h>
#include <iso646.h>

int main() {
    int a = 1, b = 0;

     if (a && !b) {
        printf("Usando && e !\n");
    }

     if (a and (not b)) {
         printf("Usando and e not\n");
    }

     int bit_a = 1, bit_b = 2;
    int resultado_and;

    resultado_and = bit_a bitand bit_b;
     printf("Bitand: %d\n", resultado_and);
```

```
    return 0;
}
```

8. <climits> : A biblioteca limits.h define macros para os limites dos tipos de dados inteiros(como INT_MAX, SHRT_MIN, etc.). O objetivo é permitir a escrita de código portável, adaptável a diferentes compiladores e arquiteturas, e garantir que as variáveis não excedam os valores que podem armazenar.
Aqui está um exemplo de uso desta biblioteca:

```
#include <stdio.h>
#include <limits.h>

int main() {
    printf("O valor máximo de um int é: %d\n", INT_MAX);

     printf("O valor mínimo de um short é: %d\n", SHRT_MIN);

    return 0;
}
```

9. <clocale> : Em C, a biblioteca locale refere-se ao cabeçalho <locale.h> e às funções associadas, como setlocale(), que são usadas para configurar informações dependentes da localização em programas multilíngues, como formatos de data, números e símbolos de moeda, e para garantir a correta exibição de caracteres especiais como "ç" e "ã" ao definir o idioma do sistema.
Exemplo de uso:

```
#include <stdio.h>
#include <locale.h>

int main() {
   setlocale(LC_ALL, "pt_BR.UTF-8");

   printf("Olá, mundo! Agora podemos usar acentos como 'ç' e 'ã'.\n");

  return 0;
}
```

10. <cmath> : A biblioteca math.h em C oferece funções matemáticas avançadas como raiz quadrada (sqrt), potência (pow), seno (sin), cosseno (cos) e logaritmo (log), exigindo a inclusão do cabeçalho #include <math.h> e, em muitos compiladores (como o GCC), a compilação com a flag -lm para ligar a biblioteca matemática.
Exemplo de uso:

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

int main(){
double cosseno;
scanf(“%lf”, &cosseno);
```

```
printf("o ângulo cujo cosseno eh %.3lf, eh %.3lf aproximadamente",
cosseno, acos(cosseno));
return 0;
}
```

11. <csetjmp> : **setjmp.h** é um cabeçalho definido na biblioteca padrão C para fornecer "saltos não locais": fluxo de controle que se desvia da sequência usual de chamada e retorno de subrotina . As funções complementares **setjmp** e **longjmp** fornecem essa funcionalidade.

Exemplo de uso:

```
#include <stdio.h>
#include <setjmp.h>

jmp_buf pulo;

int main() {
  int retorno_setjmp;

  retorno_setjmp = setjmp(pulo);

   printf("Execução do programa %d\n", retorno_setjmp);

  if (retorno_setjmp == 0) {
     printf("Definindo ponto de retorno...\n");
     longjmp(pulo, 42);
  } else {
     printf("Retorno de longjmp com o valor: %d\n", retorno_setjmp);
  }

  return 0;
}
```

12. <csignal> : No cabeçalho **signal.h** estão presentes funções e macros que lidam com os sinais enviados pelo programa executado no momento em que ele está sendo executado, como, por exemplo, quando o programa é finalizado de forma anormal ou quando algum tipo de operação aritmética apresenta erros, como a divisão por zero.

Exemplo de uso:

```
#include <stdio.h>
#include <signal.h>
#include <unistd.h>

void sig_handler(int signum) {
  printf("Sinal de interrupção recebido: %d\n", signum);
  printf("Saindo do programa.\n");

}

int main() {
```

```
   signal(SIGINT, sig_handler);

  printf("Aguardando sinal SIGINT (Ctrl+C). Pressione Ctrl+C para enviar um sinal.\n");

while(1) {
    pause();

  return 0;
}
```

13. <cstdarg> : **`stdarg.h`** é um cabeçalho da biblioteca padrão do C da linguagem de programação C que permite que funções utilizem um número indefinido de argumentos. Ela provê facilidades de navegação entre uma lista de argumentos sem que a quantidade e o tipo de argumentos seja conhecido. A linguagem C++ implementa tal funcionalidade no cabeçalho `cstdarg`; Apesar de permitido, o cabeçalho C é obsoleto no C + +.

Exemplo de uso:

```
#include <stdio.h>
#include <stdarg.h>

void imprimirNumeros(int num_args, ...) {
   va_list args;
   va_start(args, num_args);

   printf("Você forneceu %d argumentos:\n", num_args);
   for (int i = 0; i < num_args; i++) {
      int valor = va_arg(args, int);
      printf(" Argumento %d: %d\n", i + 1, valor);
   }
   va_end(args);
}

int main() {
   imprimirNumeros(3, 10, 20, 30);
   return 0;
}
```

14. <cstdbool> : É usada para manipular variáveis lógicas, como verdadeiro e falso. Pode ser substituída pela utilização de valores inteiro, sua função é simplesmente facilitar a compreensão do código. Ela é um pouco mais complexas que as outras básicas, mas podemos entende-la perfeitamente.

Exemplo de uso:

```
#include <stdbool.h>
#include <stdio.h>

int main() {
   bool esta_ativo = true;
   bool tem_permissao = false;
```

```
    if (esta_ativo) {
      printf("O estado é: Ativo\n");
    } else {
     printf("O estado é: Inativo\n");
    }

    if (!tem_permissao) {
      printf("Acesso negado.\n");
    } else {
     printf("Acesso concedido.\n");
    }

    int contador = 0;
   while (contador < 3 && esta_ativo) {
       printf("Contador: %d\n", contador);
      contador++;
   }

   return 0;
}
```

15. <cstddef> : O cabeçalho <stddef.h> em C define tipos e macros essenciais para a programação, como size_t para o tamanho de objetos, ptrdiff_t para a diferença entre ponteiros, wchar_t para caracteres largos, NULL para ponteiros nulos e offsetof para o deslocamento de um membro dentro de um struct.
Exemplo de uso:

```
#include <stddef.h>
#include <stdio.h>

int main() {

    int numeros[] = {1, 2, 3, 4, 5};
 size_t tamanho_do_array = sizeof(numeros) / sizeof(numeros[0]);

  printf("O tamanho do array é: %zu\n", tamanho_do_array);

    char *ptr = NULL;
    if (ptr == NULL) {
       printf("O ponteiro é nulo.\n");
    }

   return 0;
}
```

16. <cstdint> : O cabeçalho <stdint.h> na linguagem C fornece tipos inteiros com tamanhos fixos e garantidos, como int32_t (um inteiro de 32 bits com sinal) ou uint64_t (um inteiro de 64 bits sem sinal), resolvendo a variabilidade dos tamanhos de int e long entre diferentes arquiteturas. Isso permite escrever código portátil, onde o tamanho de uma variável inteira é consistente em

qualquer sistema, e garante o comportamento desejado em manipulações de dados.
Exemplo de uso desta biblioteca:

```
#include <stdio.h>
#include <stdint.h>

int main() {
    int8_t meu_inteiro_8bits = 100;

    uint32_t meu_inteiro_32bits = 4000000000;

    int64_t meu_inteiro_64bits = -9000000000000000LL;

   printf("Meu inteiro de 8 bits: %d\n", meu_inteiro_8bits);
   printf("Meu inteiro de 32 bits: %u\n", meu_inteiro_32bits);
    printf("Meu inteiro de 64 bits: %lld\n", meu_inteiro_64bits);

   return 0;
}
```

17. <cstdio> : O ficheiro de cabeçalho <stdio.h> em C define funções essenciais para entrada e saída de dados, como printf() para escrever no ecrã e scanf() para ler do teclado. Para utilizá-lo, basta incluir a linha #include <stdio.h> no início do seu código.
Exemplo de uso:

```
#include <stdio.h>

int main() {
   int numero;
   printf("Por favor, insira um número inteiro: ");

    scanf("%d", &numero);

    printf("Você inseriu o número: %d\n", numero);

   return 0; // Indica que o programa foi executado com sucesso
}
```

18. <cstdlib> : No cabeçalho **stdlib.h** estão localizadas as funções responsáveis pela manipulação da alocação de memória, e da desalocação também, bem como funções para converter números que estão representados em strings para algum tipo de dado responsável por representar números, como, por exemplo, double, entre outras funcionalidades úteis, como funções de algoritmos de ordenação.
Exemplo de implementação no código:

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

int main(){
```

```
char *exemplo = “42\0”;

double resposta  = atof(exemplo);

printf(“A resposta é %.0lf!”, resposta);

return 0;
}
```

19. <cstring> : Esta é uma biblioteca padrão da linguagem C. Nela estão contidos protótipos utilizados para a manipulação de **strings**, também conhecidos como array de **chars**. Estas funções são capazes desde a contagem, cópia e concatenação, comparação e diversas outras modificações.
Exemplo de implementação no código:

```c
#include <stdio.h>
#include <string.h>

int main() {
   char origem[] = "Olá, mundo!";
    char destino[20];

   strcpy(destino, origem);

   printf("String original: %s\n", origem);
   printf("String copiada:  %s\n", destino);

   return 0;
}
```

20. <ctgmath> : O **tgmath.h** é uma biblioteca do C que define macros de matemática genéricas para tipos (type-generic), o que significa que elas podem aceitar diferentes tipos de dados (como float, double, long double ou até inteiros) para realizar operações matemáticas. Ele funciona incluindo as bibliotecas <math.h> e <complex.h> e cria funções que selecionam automaticamente a versão correta da função matemática com base no tipo dos argumentos fornecidos, tornando o código mais versátil.
Exemplo de uso no código:

```
#include <tgmath.h>
#include <stdio.h>

int main() {
float a = 10.0f;
double b = 20.5;
long double c = 30.75L;
```

```
 printf("Raiz quadrada de %f: %f\n", a, sqrt(a));
printf("Raiz quadrada de %f: %f\n", b, sqrt(b));
printf("Raiz quadrada de %Lf: %Lf\n", c, sqrt(c));

return 0;
 }
```

21. <ctime> É uma biblioteca geralmente utilizada para manipulação de tempo, dando ele em segundos geralmente, e manipulando com as funções da biblioteca, podendo até converter em string

```
#include <iostream>
#include <ctime>

int main() {
    time_t agora = time(0); // tempo atual, do tipo de variável time_t

    std::cout << "Data atual: " << strftime(agora); // transforma a variável de tipo time_t em string
legível.
    return 0;
}
```

22. <cuchar> Utilizada para manipular caracteres do tipo UTF-16 e UTF-32 especificamente, caso queira especificar que é esse tipo de dado.
Exemplos de recursos são as variáveis: char16_t e char32_t, para UTF-16 e UTF-32 respectivamente.

23. <cwchar> É utilizada para manipulação de caracteres que são maiores que a tabela ASCII, ou seja, que tem um “id” muito grande, usando dados de tipo wchar_t. Um recurso por exemplo é: swprintf() que imprime o caracter no terminal.

24. <cwctype> É utilizada para identificar tipos de variáveis wchar_t.por exemplo se é uma variação das letras (A-Z, a-z) com a função iswalpha().

Conjunto de Bibliotecas 2 - Containers:

1. <array> uma biblioteca que facilita e adiciona mais utilidades a arrays, como a função size() que retorna o número de elementos.

2. <deque> é uma fila (queue) onde há como inserir um componente pelo ambos fim e início. push_back() e pop_back() são exemplos de que inserem ou deletam dados do fim respectivamente.

3. <forward_list> uma lista ordenada onde os dados são posicionados e ordenados, porém só vai pra frente, nunca indo de forma decrescente, insert() insere algo na lista.

4. <list> é um tipo de container chamado de lista duplamente emparelhada, utilizada para organizar dados em lista e também imprimir eles dos dois modos ao invés de foward list que só vai pra frente, sort() ordena os elementos e há todas as partes necessárias para manipular uma corrente de dados de tipo lista.

5. <map> Ela permite armazenar **pares chave → valor**, onde cada chave é única e os elementos são automaticamente mantidos em ordem crescente pela chave. um exemplo é emplace() onde constrói e insere o par diretamente.

6. <queue> é um container de tipo fila, porém diferente do queue, não é duplo, e só há como remover e inserir do fim da fila

7. <set> é ordenado mas sem duplicatas ideal para quando se quer armazenar dados que não devem se repetir, find() é uma função que acha algum valor que você precisa encontrar ou ver se não existe.

8. <stack> é um container de tipo pilha, onde o primeiro elemento que é criado é o último que sai (filo), um exemplo de função é empty() onde retorna true se a pilha estiver vazia.

9. <unordered_map> o mesmo que map porém não é ordenado. begin() encontra o início do mapa

10. <unordered_set> o mesmo que set porém não é ordenado. end() encontra o fim do set.

11. <vector> é um vetor dinâmico, onde é melhor que um array, porém melhor em alguns modos. Um exemplo de função é clear() onde deleta tudo que tem no vector.

Conjunto de Bibliotecas 3 – Outras bibliotecas:

1. <algorithm> Ela oferece uma coleção poderosa de funções para manipular coleções de dados, como fila, pilha, lista ordenada, entre outros. funções como find() encontram elementos e count() conta quantas vezes um valor aparece.

2. <bitset> utilizado para estruturas de dados em bit e manipulação de números binários, tendo algumas como set() que transforma tudo em 1 ou reset() que transforma todos em 0.

3. <chrono> permite manipulação de tempo de um modo diferente de <time>, com duração, mais como um cronômetro, time_point, onde representa um tempo atual. second e milisecond são tipos de classes da biblioteca <chrono>

4. <codecvt> agora em desuso, era utilizada em conversões de caracterés de um codificador de texto para outro.

```
#include <codecvt>
#include <locale>
#include <string>

std::wstring utf8_to_wstring(const std::string& str) {
   std::wstring_convert<std::codecvt_utf8<wchar_t>> converter;
   return converter.from_bytes(str);
}
```

5. <complex> utilizada para manipulação e calculos com números complexo. Como exemplo: real() que retorna a parte real de um cálculo com números complexos.

6. <exception>
Objetivo: Lidar com erros e situações inesperadas durante a execução de um programa. Como eventos anormais ou possíveis erros, permitindo que o fluxo normal do código seja interrompido e a situação seja tratada de forma controlada, evitando a falha total do programa.
Ex.:

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

class DivisaoPorZero : public exception {
public:
    const char* what() const noexcept override {
        return "Erro: divisao por zero nao permitida!";
    }
};
// serve para escrever por cima caso de erro e o cost serve
também para valores nãos serem alterados, e o ponteira aponta
para o texto de erro no caso de exeção.


double dividir(double a, double b) {
    if (b == 0) {
        throw DivisaoPorZero();
    }
```

```
    return a / b;
}

int main() {
    try {
        double x, y;
        cout << "Digite o numerador: ";
        cin >> x;
        cout << "Digite o denominador: ";
        cin >> y;

        double resultado = dividir(x, y);
        cout << "Resultado: " << resultado << endl;
    }
    catch (const DivisaoPorZero& e) {
        cout << e.what() << endl;
    }

    return 0;
}
```

7. <functional>
Objetivo: Serve para reunir e reutilizar funções pré-definidas, permitindo escrever um código mais limpo, modular e eficiente. Isso é feito organizando essas funções em coleções (bibliotecas) que podem ser chamadas a partir de qualquer parte de um programa, evitando a duplicação de código e facilitando a manutenção.
Ex.:

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

int soma(int a, int b) {
   return a + b;
}

int multiplica(int a, int b) {
   return a * b;
}

int main() {

   function<int(int, int)> operacao;

   int x = 8, y = 4;
```

```
    operacao = soma;
    cout << "Soma: " << operacao(x, y) << endl;

    operacao = multiplica;
    cout << "Multiplicacao: " << operacao(x, y) << endl;

    return 0;
}
```

8. <initializer_list>
Objetivo: Permite inicializar coleções de dados com {} de forma prática. Muito usado em construtores e funções que recebem listas de valores.
Ex.:

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

int soma(initializer_list<int> valores) {
    int total = 0;
    for (int v : valores) {
        total += v;
    }
    return total;
}

int main() {
    // Passa para a lista os valores diretamente
    cout << "Soma de {1, 2, 3}: " << soma({1, 2, 3}) << endl;
    cout << "Soma de {10, 20, 30, 40}: " << soma({10, 20, 30,
40}) << endl;

    return 0;
}
```

9. <iterator>
Objetivo: Fornece funções e classes para manipulação de dados, que são ponteiros inteligentes usados em vetores, listas e outros containers.
Ex.:

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

int main() {
    int arr[] = {1, 2, 3, 4, 5};

    auto it = begin(arr);
    advance(it, 3);
    cout << "Elemento na posicao 3: " << *it << endl;

    return 0;
}
```

10. <limits>
Objetivo:Contém informações sobre os limites dos tipos numéricos, como valores máximos e mínimos que podem ser representados
Ex.:

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

int main() {
    cout << "Tipo int:" << endl;
    cout << "  Minimo: " << numeric_limits<int>::min() << endl;
    cout << "  Maximo: " << numeric_limits<int>::max() << endl;

    cout << "\nTipo double:" << endl;
    cout << "  Minimo: " << numeric_limits<double>::min() << endl;
    cout << "  Maximo: " << numeric_limits<double>::max() << endl;
    cout << "  Digitos maximos de precisao: "
         << numeric_limits<double>::digits10 << endl;

    cout << "\nTipo bool:" << endl;
    cout << "  Falso: " << numeric_limits<bool>::min() << endl;
    cout << "  Verdadeiro: " << numeric_limits<bool>::max() << endl;

    return 0;
}
```

11. <locale>
Objetivo: Fornece suporte à localização (internacionalização), permitindo formatar números, moedas e textos de acordo com diferentes culturas.

Ex.:

```
#include <iostream>
#include <locale>

using namespace std;

int main() {
    // Cria um locale para português do Brasil
    locale loc("pt_BR.utf8");

    // Aplica o locale ao cout
    cout.imbue(loc);

    // Exibe número formatado
    cout << "Número formatado: " << 1234567.89 << endl;

    // Converte caractere para maiúscula
    char letra = 'ç';
    cout << "Maiúscula de '" << letra << "': " << toupper(letra, loc) << endl;

    return 0;
}
```

12. <memory>

Objetivo:Serve para cuidar da memória automaticamente com “ponteiros inteligentes”, que limpam a memória sozinhos quando não é mais usada.

Ex.:

```
#include <iostream>
#include <memory>

using namespace std;

int main() {
    unique_ptr<int> ptr(new int(42));
    cout << "Valor: " << *ptr << endl;
    return 0;
}
```

13. <new>

Objetivo:Usado para trabalhar com alocação manual de memória, criando e liberando espaço com `new` e `delete`.

Ex.:

```
#include <iostream>
```

```
#include <new>

using namespace std;
int main() {

    int* p = new int(2025);

    cout << "Valor alocado: " << *p << endl;

    delete p;

    return 0;
}
```

14. <numeric>
Objetivo: Traz funções matemáticas que trabalham em listas, como somar todos os valores (`accumulate`) ou gerar sequências (`iota`).
Ex.:

```
#include <iostream>
#include <numeric>
#include <vector>

using namespace std;

int main() {
    vector<int> v = {1, 2, 3, 4, 5};
    int soma = accumulate(v.begin(), v.end(), 0);
    cout << "Soma: " << soma << endl;
    return 0;
}
```

15. <random>
Objetivo: Gera números aleatórios de forma moderna e segura, como simular um dado ou sorteio.
Ex.:

```
#include <iostream>
#include <random>

using namespace std;

int main() {
    random_device rd;
    mt19937 gen(rd());
```

```
    uniform_int_distribution<> dist(1, 100);

    cout << "Número aleatório: " << dist(gen) << endl;
    return 0;
}
```

16. <ratio>
Objetivo:Permite representar frações fixas (como 1/2, 3/4) já no tempo de compilação. É útil em cálculos de tempo e unidades.
Ex.:

```
#include <iostream>
#include <ratio>

using namespace std;

int main() {
    typedef ratio<1, 1000> mili; // 1 milésimo
    cout << "mili: " << mili::num << "/" << mili::den << endl;
    return 0;
}
```

17. <regex>
Objetivo:Serve para procurar padrões em textos. Exemplo: verificar se uma palavra é só números ou se um texto parece um e-mail.
Ex.:

```
#include <iostream>
#include <regex>
#include <string>

using namespace std;

int main() {
    string email = "exemplo@dominio.com";
    regex padrao(".+@.+\\..+");

    if (regex_match(email, padrao)) {
        cout << "E-mail válido!" << endl;
    } else {
        cout << "E-mail inválido!" << endl;
    }

    return 0;
}
```

18. <stdexcept>
Objetivo:Define exceções (erros) comuns já prontas, como “argumento inválido” ou “erro de execução”, que podem ser lançadas no código. Ao contrário do exception que pega qualquer exceção esse só pega as específicas ou pré programadas.
Ex.:

```
#include <iostream>
#include <stdexcept>

using namespace std;

int main() {
    try {
        throw out_of_range("Índice fora do limite!");
    } catch (const exception& e) {
        cout << "Erro específico: " << e.what() << endl;
    }
    return 0;
}
```

19. <string>
Objetivo: É a biblioteca que dá suporte a `std::string`, usada para manipular textos facilmente (concatenar, comparar, buscar palavras).
Ex.:

```
#include <iostream>
#include <string>

using namespace std;

int main() {
    string nome = "Maria";
    string saudacao = "Olá, " + nome + "!";

    cout << saudacao << endl;

    if (saudacao.find("Maria") != string::npos) {
        cout << "Nome encontrado na saudação." << endl;
    }

    return 0;
}
```

20. <system_error>

Objetivo: Lida com erros vindos do sistema operacional, transformando códigos de erro em mensagens entendíveis.
Ex.:

```
#include <iostream>
#include <system_error>

using namespace std;

int main() {
    try {
        error_code ec(5, generic_category()); // código 5 = acesso negado
        throw system_error(ec, "Falha ao acessar recurso");
    } catch (const system_error& e) {
        cout << "Erro: " << e.what() << endl;
        cout << "Código: " << e.code() << endl;
    }
    return 0;
}
```

21. <tuple>
Objetivo: Permite guardar vários valores de tipos diferentes juntos em um só pacote, acessando cada um por índice (`get<0>`, `get<1>`, etc.).Tipo banco de dados.
Ex.:

```
#include <iostream>
#include <tuple>

using namespace std;

int main() {
    tuple<int, string, double> dados(42, "idade", 3.14);

    cout << "Inteiro: " << get<0>(dados) << endl;
    cout << "Texto: " << get<1>(dados) << endl;
    cout << "Decimal: " << get<2>(dados) << endl;

    return 0;
}
```

22. <type_traits>
A `<type_traits>` é uma biblioteca de metaprogramação introduzida no C++11 que permite inspecionar e modificar tipos em tempo de compilação. Ela é usada principalmente com templates para tornar o código mais genérico, seguro e adaptável.

Ex.:

```
#include <iostream>
#include <type_traits>

using namespace std;

template<typename T>
void verificaTipo(T valor) {
    if (is_integral<T>::value) {
        cout << "É um tipo inteiro." << endl;
    } else {
        cout << "Não é um tipo inteiro." << endl;
    }
}

int main() {
    verificaTipo(42);       // int → inteiro
    verificaTipo(3.14);     // double → não inteiro
    return 0;
}
```

23. <typeindex>

Objetivo:Permite comparar tipos em tempo de execução, ou seja, descobrir de qual tipo é um objeto enquanto o programa roda.

Ex.:

```
#include <iostream>
#include <typeindex>
#include <typeinfo>

using namespace std;

int main() {
    type_index t1 = typeid(int);
    type_index t2 = typeid(double);

    if (t1 != t2) {
        cout << "Tipos diferentes: " << t1.name() << " vs " << t2.name() << endl;
    }

    return 0;
}
```

24. <typeinfo>
Objetivo:Usado com `typeid` para identificar tipos em tempo de execução, especialmente útil em herança e polimorfismo.
Ex.:

```
#include <iostream>
#include <typeinfo>

using namespace std;

int main() {
    double valor = 3.14;
    cout << "Tipo de valor: " << typeid(valor).name() << endl;
    return 0;
}
```

25. <utility>
Objetivo:Traz funções práticas como `swap` (trocar valores) e `pair` (guardar dois valores juntos). É bem usado em algoritmos da STL.
Ex.:

```
#include <iostream>
#include <utility>

using namespace std;

int main() {
    pair<string, int> pessoa("João", 30);
    cout << pessoa.first << " tem " << pessoa.second << " anos." << endl;
    return 0;
}
```

26. <valarray>
Objetivo:É como um vetor, mas feito para cálculos matemáticos rápidos em todos os elementos ao mesmo tempo.
Ex.:

```
#include <iostream>
#include <valarray>

using namespace std;

int main() {
    valarray<int> v = {1, 2, 3, 4, 5};
    valarray<int> resultado = v * 2;
```

```
    for (int x : resultado) {
        cout << x << " ";
    }
    cout << endl;

    return 0;
}
```