**Table of Content**

|  |  |
| --- | --- |
| **No** | **Topic Name** |
| 1 | Features of C++ (Why learn C++) |
| 2 | Basics (Variables, Data types, Basic I/0, Operators, comments) |
| 3 | Control Statements |
| 4 | Functions (static also) |
| 5 | Arrays & Strings |
| 6 | Pointers, Reference and Function pointers |
| 7 | Object, Class (Initializer List and Delegate constructors) |
| 8 | OOPS Concepts |
|  | a. Inheritance |
|  | b. Polymorphism (Operator overloading) |
|  | c. Abstraction |
|  | d. Encapsulation |
| 13 | Abstract and derived class |
| 13 | Typecasting (RTTI) |
| 14 | Templates |
| 15 | Containers |
| 16 | Smart pointers |
| 17 | Lambda expressions |
| 18 | Regular expressions |
| 19 | Design patterns |
| 20 | Threads and Semaphore |

**Features of C++ (Why learn C++)**

C++ is a general-purpose, object-oriented programming language that was designed by **Bjarne Stroustrup in 1979** to be an extension of the C language. It has the features of imperative, object-oriented as well as generic programming models. C++ also has some additional facilities to those in C such as*classes, inheritance, default function argument,* etc. C++ plays quite an integral role in modern times as many contemporary systems such as operating systems, web browsers, databases, etc. have C++ code in at least some part of their codebase. Moreover, C++ is quite useful in performance-critical areas because of its speed.

Keeping the popularity and relevance of C++ in mind, the top 10 reasons to learn it are given below:

**1. C++ Popularity and High Salary**

C++ is one of the most popular languages in the world. It is used by some **4.4 million developers** worldwide. Also, C++ Developers are quite sought, and they hold some of the most high-paying jobs in the industry with an average base pay of **$103, 035 per year**.

**2. C++ has Abundant Library Support**

C++ has the **Standard Template Library (STL)** which is very useful as it helps in writing code compactly and quickly as required. It contains mainly four components i.e., algorithms, containers, functions, and iterators. The algorithms are of different types such as sorting, searching, etc. The containers store classes to implement different data structures that are commonly used such as stacks, queues, hash tables, vectors, sets, lists, maps, etc. The functors allow the working of the associated function to be customized with the help of the parameters passed. Also, the iterators are used for working upon a sequence of values.

C++ STL self-paced is full of C++ concepts like how to use functions, loops, arrays, structure, etc., and advanced concepts like algorithms.

### 3. C++ has a Large Community

There is a large online community of C++ users and experts that is particularly helpful in case any support is required. There are a lot of resources like Geekforgeeks etc. available on the internet regarding C++. Some of the other online resources for C++ include **Stack Overflow**, **cppreference.com**, **Standard C++**, etc.

### 4. C++ in Databases

There are many modern-day databases such as **MySQL,**MongDB**, MemSQL,**etc. that are written in C++. This is because C++ is quite modern and it supports features like exceptions, lambda expressions, etc. Many of the databases that are written in C++ are used in almost all the in-use applications such as YouTube, WordPress, Twitter, Facebook, etc.

Want to step into the world of programming, then C++ is the language you need to learn.

### 5. C++ in Operating Systems

All the major operating systems such as **Windows, Linux, Android, Ubuntu, iOS**, etc. are written in a combination of C and C++. The Windows applications are written in C++, while Android applications are written in Java along with C/C++ with non-default run-times for C++ support. Also, C++ can be used to develop the core of the applications in iOS. In general, C or C++ are used in operating systems because of the speed and strongly typed nature of these languages.

### 6. C++ in Compilers

C++ is closer to the hardware level and is a comparatively low-level language. Because of this reason, it is used in many compilers as a backend programming language. An example of this is the **GNU Compiler Collection (GCC)** which is currently written mostly in C++ along with C.

### 7. C++ in Web Browsers

A lot of web browsers are developed using C++ such as **Chrome, Firefox, Safari,**etc. Chrome contains C++ in the rendering engine, JavaScript engine, and UI. Firefox uses mainly in the rendering engine and a little in the UI. Safari also uses C++ in the rendering engine and JavaScript engine. All these web browsers and more use C++, particularly in the rendering engines because it provides the required speed that is necessary for the rendering engines since they need to display the content at an accelerated rate.

### 8. C++ in Graphics

Applications requiring graphics such as **digital image processing, computer vision, screen recording programs,**etc. use C++ due to its high speed. This can also include different games that have graphics as a big part of their structure.

### 9. C++ in Embedded Systems

C++ is closer to the hardware level and so it is quite useful in embedded systems as the software and hardware in these are closely coupled. There are many embedded systems that use C++ such as **smart watches, MP3 players, GPS systems**, etc.

### 10. C++ is Portable

Program developed in C++ that can be moved from one platform to another. This is one of the main reasons that applications requiring multi-platform or multi-device development often use C++.

**Some top applications used in C plus:**

* YouTube
* Amazon
* Adobe Photoshop and illustrator
* Windows
* Microsoft office
* MySQL
* Mozilla Firefox

**Basics (Variables, Data types, Basic I/0, Operators, comments)**

# **C++ Variable**

A variable is a name of memory location. It is used to store data. Its value can be changed, and it can be reused many times. It is a way to represent memory location through symbol so that it can be easily identified.

Let's see the syntax to declare a variable:

* type variable\_list;

The example of declaring variable is given below:

* **int** x;
* **float** y;
* **char** z;

Here, x, y, z are variables and int, float, char are data types.

We can also provide values while declaring the variables as given below:

* **int** x=5, b=10; //declaring 2 variable of integer type
* **float** f=30.8;
* **char** c='A';

## **Rules for defining variables:**

* A variable can have alphabets, digits, and underscore.
* A variable name can start with alphabet and underscore only. It can't start with digit.
* No white space is allowed within variable name.
* A variable name must not be any reserved word or keyword e.g. char, float etc.

**Valid variable names:**

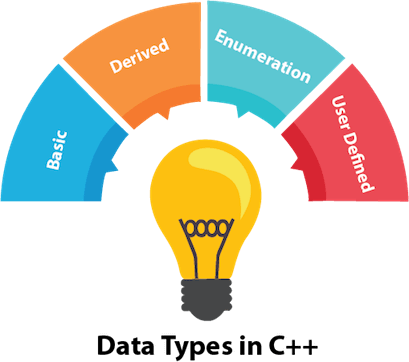
* **int** a;
* **int** \_ab;
* **int** a30;

**Invalid variable names:**

* **int** 4;
* **int** x y;
* **int** **double**;

# **C++ Data Types**

A data type specifies the type of data that a variable can store such as integer, floating, character etc.



There are 4 types of data types in C++ language.

|  |  |
| --- | --- |
| **Types** | **Data Types** |
| Basic Data Type | int, char, float, double, etc. |
| Derived Data Type | array, pointer, etc. |
| Enumeration Data Type | Enum |
| User Defined Data Type | structure |

## **Basic Data Types**

The basic data types are integer-based and floating-point based. C++ language supports both signed and unsigned literals.

The memory size of basic data types may change according to 32- or 64-bit operating system.

Let's see the basic data types. Its size is given according to 32 bit OS.

|  |  |  |
| --- | --- | --- |
| **Data Types** | **Memory Size** | **Range** |
| char | 1 byte | -128 to 127 |
| signed char | 1 byte | -128 to 127 |
| unsigned char | 1 byte | 0 to 127 |
| short | 2 byte | -32,768 to 32,767 |
| signed short | 2 byte | -32,768 to 32,767 |
| unsigned short | 2 byte | 0 to 32,767 |
| int | 2 byte | -32,768 to 32,767 |
| signed int | 2 byte | -32,768 to 32,767 |
| unsigned int | 2 byte | 0 to 32,767 |
| short int | 2 byte | -32,768 to 32,767 |
| signed short int | 2 byte | -32,768 to 32,767 |
| unsigned short int | 2 byte | 0 to 32,767 |
| long int | 4 byte |  |
| signed long int | 4 byte |  |
| unsigned long int | 4 byte |  |
| float | 4 byte |  |
| double | 8 byte |  |
| long double | 10 byte |  |

# **C++ Basic Input/Output**

C++ I/O operation is using the stream concept. Stream is the sequence of bytes or flow of data. It makes the performance fast.

If bytes flow from main memory to device like printer, display screen, or a network connection, etc., this is called as **output operation.**

If bytes flow from device like printer, display screen, or a network connection, etc. to main memory, this is called as **input operation.**

## **I/O Library Header Files**

Let us see the common header files used in C++ programming are:

|  |  |
| --- | --- |
| **Header File** | **Function and Description** |
| <iostream> | It is used to define the **cout, cin and cerr** objects, which correspond to standard output stream, standard input stream and standard error stream, respectively. |
| <iomanip> | It is used to declare services useful for performing formatted I/O, such as **set precision and setw.** |
| <fstream> | It is used to declare services for user-controlled file processing. |

## **Standard output stream (cout)**

The **cout** is a predefined object of **ostream** class. It relates to the standard output device, which is usually a display screen. The cout is used in conjunction with stream insertion operator (<<) to display the output on a console

Let's see the simple example of standard output stream (cout):

#include <iostream>

**using** **namespace** std;

**int** main() {

**char** ary[] = "Welcome to C++ tutorial";

   cout << "Value of ary is: " << ary << endl;

}

**Output:**

Value of ary is: Welcome to C++ tutorial

## **Standard input stream (cin)**

The **cin** is a predefined object of **istream** class. It relates to the standard input device, which is usually a keyboard. The cin is used in conjunction with stream extraction operator (>>) to read the input from a console.

Let's see the simple example of standard input stream (cin):

#include <iostream>

**using** **namespace** std;

**int** main() {

**int** age;

   cout << "Enter your age: ";

   cin >> age;

   cout << "Your age is: " << age << endl;

}

**Output:**

Enter your age: 22

Your age is: 22

## **Standard end line (endl):**

The **endl** is a predefined object of **ostream** class. It is used to insert a new line character and flushes the stream.

Let's see the simple example of standard end line (endl):

#include <iostream>

**using** **namespace** std;

**int** main( ) {

cout << "C++ Tutorial";

cout << " Javatpoint"<<endl;

cout << "End of line"<<endl;   }

**Output:**

C++ Tutorial Javatpoint

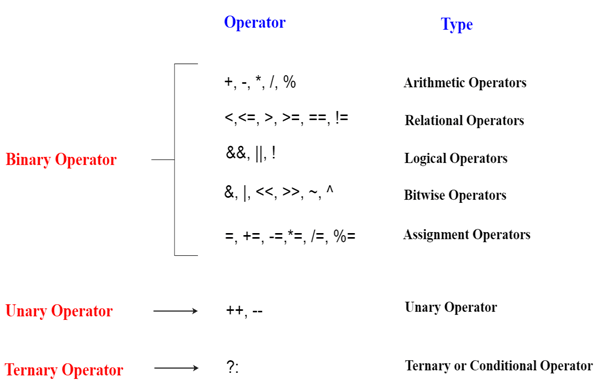
End of line

# **C++ Operators**

An operator is simply a symbol that is used to perform operations. There can be many types of operations like arithmetic, logical, bitwise etc.

There are following types of operators to perform different types of operations in C language.

* Arithmetic Operators
* Relational Operators
* Logical Operators
* Bitwise Operators
* Assignment Operator
* Unary operator
* Ternary or Conditional Operator
* Misc Operator



## **Precedence of Operators in C++**

The precedence of operator species that which operator will be evaluated first and next. The associativity specifies the operator’s direction to be evaluated, it may be left to right or right to left.

Let's understand the precedence by the example given below:y Video

* **int** data=5+10\*10;

The "data" variable will contain 105 because \* (multiplicative operator) is evaluated before + (additive operator).

The precedence and associativity of C++ operators is given below:

|  |  |  |
| --- | --- | --- |
| **Category** | **Operator** | **Associativity** |
| Postfix | () [] -> . ++ - - | Left to right |
| Unary | + - ! ~ ++ - - (type)\* & sizeof | Right to left |
| Multiplicative | \* / % | Left to right |
| Additive | + - | Right to left |
| Shift | << >> | Left to right |
| Relational | < <= > >= | Left to right |
| Equality | == !=/td> | Right to left |
| Bitwise AND | & | Left to right |
| Bitwise XOR | ^ | Left to right |
| Bitwise OR | | | Right to left |
| Logical AND | && | Left to right |
| Logical OR | || | Left to right |
| Conditional | ?: | Right to left |
| Assignment | = += -= \*= /= %=>>= <<= &= ^= |= | Right to left |
| Comma | , | Left to right |

## **C++ Comments**

Comments can be used to explain C++ code, and to make it more readable. It can also be used to prevent execution when testing alternative code. Comments can be singled-lined or multi-lined.

## **Single-line Comments**

* Single-line comments start with two forward slashes (//).
* Any text between // and the end of the line is ignored by the compiler (will not be executed).

This example uses a single-line comment before a line of code:

### Example

// This is a comment  
cout << "Hello World!";

This example uses a single-line comment at the end of a line of code:

### Example

cout << "Hello World!"; // This is a comment

## **C++ Multi-line Comments**

Multi-line comments start with /\* and ends with \*/.

Any text between /\* and \*/ will be ignored by the compiler:

### Example

/\* The code below will print the words Hello World!  
to the screen, and it is amazing \*/  
cout << "Hello World!";

#### **Single or multi-line comments?**

It is up to you which you want to use. Normally, we use // for short comments, and /\* \*/ for longer.

|  |
| --- |
|  |

**CONTROL STATEMENTS**

**INTRODUCTION:**

Sometimes the program needs to be executed depending upon a particular condition. Control statements redirect’s the programs flow to execute additional code. These are used to control the flow of execution of programs. There are different control statements which are available in C++, which are given below:

* Decision making statements (if, if-else, Nested if, if-else-if)
* Selection statements (Switch)
* Iteration statements (for, for-each, while, do-while)
* Jump statements (break, continue, goto)

**DECISION MAKING STATEMENTS:**

**1. if statement:** The ‘if’ keyword is used to execute a statement or block if and only if a condition is satisfied.

**Syntax**:

If (condition) {

//statement

}

**Example**:

#include<iostream>

using namespace std;

int main(){

int i=20;

if(i>10) {

cout<<” Condition is true”;

}

return 0;

}

**OUTPUT**:

Condition is true

**2**. **if-else** **statement:** if-else statement tells us that if a condition is true, it prints the statements which is inside the ‘if’ block, otherwise it prints the else block.

**Syntax**:

If (condition) {

//statements

} else{

//statements

}

**Example**:

#include <iostream>

using namespace std;

int main() {

int n;

cout << "Enter an integer: ";

cin >> n;

if ( n % 2 == 0)

cout << n << " is even.";

else

cout << n << " is odd.";

return 0;

}

**OUTPUT:**

Enter an integer:18

18 is even

**3**. **nested-if statement:** In nested-if statement, we have ‘if’ statements which is present inside the other ‘if’ block.

**Syntax**:

If (condition1) {

//executes when condition 1 is true

If (condition 2) {

//executes when condition 2 is true

}

}

**Example**:

#include <iostream>

using namespace std;

int main() {

int num;

cout << "Enter an integer: ";

cin >> num;

// outer if condition

if (num != 0) {

// inner if condition

if (num > 0) {

cout << "The number is positive." << endl;

}

// inner else condition

else {

cout << "The number is negative." << endl;

}

}

// outer else condition

else {

cout << "The number is 0 and it is neither positive nor negative." << endl;

}

return 0;

}

**OUTPUT:**  
Enter an integer:5

The number is positive

**4**. **if-else-if** **statement:** This statement is executed in a top-down manner. If the condition is found that evaluates to true, the statement associated with that condition will be executed if the condition is satisfied in the ladder by passing the rest of the ladder. If the condition is false, the ‘else’ statement will be executed. If there is no ‘else’ statement nothing will be printed.

**Syntax**:

If (condition 1) {

//statements

}else if (condition 2) {

//statements

}else if (condition 3) {

//statements

}else {

//statements

}

**Example**:

#include <iostream>

using namespace std;

int main()

{

int i = 20;

if (i == 10)

cout << "i is 10";

else if (i == 15)

cout << "i is 15";

else if (i == 20)

cout << "i is 20";

else

cout << "i is not present";

return 0;

}

**OUTPUT:**

i is 20

**SELECTION STATEMENTS**

* **Switch** **statement:** Switch case statement evaluates a given expression and based on the evaluated value, it executes the statements associated with it. Basically, it is used to perform different actions based on different conditions.
  + Switch case statements follow a selection-control mechanism and allow a value to change control of execution.
  + They are a substitute for long if statements that compare a variable to several integral values.
  + The switch statement is a multiway branch statement. It provides an easy way to dispatch execution to different parts of code based on the value of the expression.
  + Switch statement consists of conditional based cases and a default case.
  + In a switch statement, the “case value” can be of “char” and “int” type.
  + some of the rules while using the switch statement:

1. There can be one or N numbers of cases.

2. The values in the case must be unique.

3. Each statement of the case can have a break statement. It is optional.

**Syntax**:

switch(expression)

{

case value1:

statement\_1;

break;

case value2:

statement\_2;

break;

......

case value\_n:

statement\_n;

break;

default:

default statement;

}

**Example**:

#include<iostream>

using namespace std;

class Day

{

private:

int day;

public:

void set\_data()

{

cout<<"Enter no of day you want to display: ";

cin>>day;

}

void display\_day()

{

switch (day)

{

case 1:

cout<<"MONDAY";

break;

case 2:

cout<<"TUESDAY";

break;

case 3:

cout<<"WEDNESDAY";

break;

case 4:

cout<<"THURSDAY";

break;

case 5:

cout<<"FRIDAY";

break;

case 6:

cout<<"SATURDAY";

break;

case 7:

cout<<"SUNDAY";

break;

default:

cout<<"INVALID INPUT";

break;

}

}

};

int main()

{

Day d1;

d1.set\_data();

d1.display\_day();

return 0;

}

**OUTPUT:**

Enter no of day you want to display :6

SATURDAY

**LOOPING STATEMENTS:**

**1.For loop:** The C++ ‘for’ loop is used to iterate a part of the program several times. If the number of iterations is fixed, we can use for loop.

**Syntax**:

**for**(initialization; condition; incr/decr){

//code to be executed

}

**Example**:

#include <iostream>

**using** **namespace** std;

**int** main() {

**for**(**int** i=1;i<=6;i++){

            cout<<i <<"\t";

          }

    }

**OUTPUT:**

1 2 3 4 5 6

**2**. **for-each** **loop:** for loop is used to iterate over the elements of a containers quickly without performing initialization, testing and increment/decrement. The working of foreach loops is to do something for every element rather than doing something n times.

**Syntax**:

for(type range-declaration: array/vector\_name)

{

Statements to execute here

…

}

**Example**:

#include <iostream>

using namespace std;

int main() {

int numArr[] = { 5, 10, 15, 20, 25 };

for (int x : numArr)

cout << x << “\t”;

}

**OUTPUT:**

5 10 15 20 25

**3.While** **loop:** The while loop loops through a block of code if a specified condition is true. In a while loop, the condition is evaluated first and if returns true, the statements inside the while loop will be executed. This repeatedly happens until the condition returns false. When the condition returns false, the control comes out of the loop and jumps to the program’s next statement after the while loop.

**Syntax**:

while (condition) {  
*// code block to be executed*  
}

**Example**:

#include <iostream>

using namespace std;

int main() {

int i = 0;

while (i < 5) {

cout << i << "\t";

i++;

}

return 0;

}

**OUTPUT:**

0 1 2 3 4

**4.do-while loop:** The do-while loop is like the while loop with one difference i.e., the condition is tested at the end of the loop body thus the loop body will be executed at least once irrespective of the condition which makes it an exit-controlled loop.

**Syntax**:

Initialization;

do{

//statements

update\_expression;

}while(condition);

**Example**:

#include <iostream>

using namespace std;

int main() {

int i = 0;

do {

cout << i << "\t";

i++;

}

while (i < 5);

return 0;

}

**OUTPUT:**

0 1 2 3 4

**JUMP STATEMENTS**

**1.Break** statement: The break in C++ is a loop control statement that is used to terminate the loop. As soon as the break statement is encountered from within a loop, the loop iterations stop there, and control returns from the loop immediately to the first statement after the loop.

**Syntax**:

break;

**Example**:

#include <iostream>

using namespace std;

void findElement(int arr[], int size, int key)

{

for (int i = 0; i < size; i++) {

if (arr[i] == key) {

cout << "Element found at position: " << (i + 1);

break;

}

}

}

int main()

{

int arr[] = { 1, 2, 3, 4, 5, 6 };

int n = 6;

int key = 3;

findElement(arr, n, key);

return 0;

}

**OUTPUT:**

Element found at position:3

**2.continue statement:** C++ continue statement is a loop control statement that forces the program control to execute the next iteration of the loop. As a result, the code inside the loop following the continue statement will be skipped and the next iteration of the loop will begin.

**Syntax**:

Continue;

**Example**:

#include <iostream>

using namespace std;

int main()

{

for (int i = 1; i <= 10; i++) {

if (i == 4)

continue;

else

cout << i << " ";

}

return 0;

}

**OUTPUT:**

1 2 3 5 6 7 8 9 10

**3.goto statement:** The goto statement is a jump statement which is sometimes also referred to as unconditional jump statement. The goto statement can be used to jump from anywhere to anywhere within a function.

**Syntax**:

Syntax1 | Syntax2

----------------------------

goto label; | label:

. | .

. | .

. | .

label: | goto label;

**Example**:

#include <iostream>

using namespace std;

void checkEvenOrNot (int num)

{

if (num % 2 == 0)

goto even;

else

goto odd;

even:

cout << num << " is even";

return;

odd:

cout << num << " is odd";

}

int main()

{

int num = 26;

checkEvenOrNot(num);

return 0;

}

**OUTPUT:**

26 is even

**FUNCTIONS**

**INTRODUCTION:**

A function is a block of code which only runs when it is called. You can pass data, known as parameters, into a function. Functions are used to perform certain actions, and they are important for reusing code. Define the code once and use it many times.

A C++ function consist of two parts:

* **Declaration:** the return type, the name of the function, and parameters (if any)
* **Definition:** the body of the function (code to be executed)

**Syntax**:

return\_type functionname (parameter1, parameter2,etc..){

//function body

}

**Example**:

#include <iostream>

using namespace std;

// Function declaration

void myFunction();

// The main method

int main() {

myFunction(); // call the function

return 0;

}

// Function definition

void myFunction() {

cout << " Executed!";

}

**OUTPUT:**

Executed!

**Types of functions:**

**1. Built in functions:** Built-in functions are also called library functions. These are the functions that are provided by C++, and we need not write them ourselves. We can directly use these functions in our code. These functions are placed in the header files of C++.

**Example**:

#include<iostream>

#include<cmath>

using namespace std;

int main(){

int maximum=max(100,30);

cout<<maximum;

return 0;

}

**OUTPUT**:

100

**2. User defined functions:** C++ also allows its users to define their own functions. These are the user-defined functions. We can define the functions anywhere in the program and then call these functions from any part of the code. Just like variables, it should be declared before using, functions also need to be declared before they are called. There are four combinations of arguments and return type.

* Function with no argument and no return value
* Function with arguments and no return value
* Function with no arguments and returns a value
* Function with arguments and return type

**Example**: **Function with argument and no return type**

#include<iostream>

using namespace std;

void checkevenodd( int num){

if (num%2==0)

cout<<"Even number";

else

cout<<"Odd number";

}

int main(){

int num=20;

checkevenodd(num);

return 0;

}

**OUTPUT:**

Even number

**PASSING PARAMETERS TO A FUNCTION**:

The parameters passed to a function is called actual parameters. The parameters received by the function are called formal parameters. There are 2 types of parameters passing techniques.

**1.Pass by value**: **Pass by value** means that a copy of the actual parameter’s value is made in memory, i.e., the caller and callee have two independent variables with the same value. If the callee modifies the parameter value, the effect is not visible to the caller.

**Example**:

#include<iostream>

using namespace std;

void func(int x){

x=50;

cout<<x<<endl;

}

int main(){

int x=10;

func(x);

cout<<x;

}

**OUTPUT:**

50

10

**2.Pass by reference: Pass by reference** means to pass the reference of an argument in the calling function to the corresponding formal parameter of the called function so that a copy of the address of the actual parameter is made in memory, i.e., the caller and the callee use the same variable for the parameter. If the callee modifies the parameter variable, the effect is visible to the caller’s variable.

**Example**:

#include<iostream>

using namespace std;

void func(int &x){

x=50;

cout<<x<<endl;

}

int main(){

int x=10;

func(x);

cout<<x;

}

**OUTPUT:**

50

50

**ARRAYS**

An array is a sequence of objects of the same type that occupy a contiguous area of memory. Traditional C-style arrays are the source of many bugs, but are still common, especially in older code bases.

An array is a collection of similar data items stored at contiguous memory locations and elements can be accessed randomly using indices of an array. They can be used to store the collection of primitive data types such as int, float, double, char, etc. of any type. To add to it, an array can store derived data types such as structures, pointers etc.

Given below is the picture representation of an array.



**Why do we need arrays?**

The main idea to use arrays is to represent many instances in one variable. The storing of all the variables becomes more difficult so to reduce the time and easy way to store is through the arrays.

**Initialization Of Arrays:**

int a[10];

for (int i = 0; i < 10; ++i)

{

a[i] = i + 1;

}

int b[10]{ 1, 2, 3, 4, 5, 6, 7, 8, 9, 10 };

**Accessing Array Elements:**

In C++, we can access array element using array name and subscript/index written inside pair of square brackets [].

Example :

Suppose we have an integer array of length 5 whose name is age.

int age[5] = {8,6,9,4,3};

**age[0]** = First element of array age = 8

**age[4]** = Last element of array age = 7

**Program for accessing array elements**

#include <iostream>

**using** **namespace** std;

**int** main()

{

**int** age[7] = {8,5,2,3,7,6,7};

**int** i;

**for**(i = 0; i < 7; i++)

{

        cout << "Element at index " << i <<" is " << age[i];

        cout << endl;

     }

**return** 0;

}

**OUTPUT:**

Element at index0 is 8

Element at index1 is 5

Element at index2 is 2

Element at index3 is 3

Element at index4 is 7

Element at index5 is 6

Element at index6 is 7

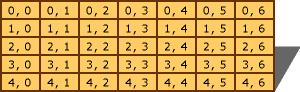
**MULTIDIMENSIONAL ARRAYS:**

Arrays constructed from other arrays are multidimensional arrays. These multidimensional arrays are specified by placing multiple bracketed constant expressions in sequence.

For example, consider this declaration:

***int i2[5][7];***

It specifies an array of type **int**, conceptually arranged in a two-dimensional matrix of five rows and seven columns, as shown in the following figure:



**Program for demonstrating multi-Dimensional Arrays.**

#include<iostream>

using namespace std;

int main()

{

int row, col, i, j, arr[10][10];

cout<<"Enter the Row and Column Size for Array: ";

cin>>row>>col;

cout<<"Enter "<<row\*col<<" Array Elements: ";

for(i=0; i<row; i++)

{

for(j=0; j<col; j++)

cin>>arr[i][j];

}

cout<<"\nThe Array is:\n";

for(i=0; i<row; i++)

{

for(j=0; j<col; j++)

cout<<arr[i][j]<<" ";

cout<<endl;

}

cout<<"\nArray Elements with its Index:\n";

for(i=0; i<row; i++)

{

for(j=0; j<col; j++)

cout<<"arr["<<i<<"]["<<j<<"] = "<<arr[i][j]<<" ";

cout<<endl;

}

cout<<endl;

return 0;

}

**OUTPUT:**

Enter the Row and Column Size for Array:2 2

Enter 4 Array Elements: 1 2 3 4

The Array is:

1 2

3 4

Array Elements with its Index:

arr[0][0]=1 arr[0][1]=2

arr[1][0]=3 arr[1][1]=4

**STRINGS**

A string in C++ is a type of object representing a collection (or sequence) of different characters. Strings in C++ are a part of the standard string class (std::string).

The string class stores the characters of a string as a **collection of bytes** in contiguous memory locations.

**Different ways of defining a string**

* Using the string keyword is the best and the most convenient way of defining a string. We should prefer using the string keyword because it is easy to write and understand.

string str\_name = "hello";

// or

string str\_name("hello");

* Then we have the C-style strings. Here are the different ways to create C-style strings:

char str\_name[6] = {'h', 'e', 'l', 'l', 'o', '\0'};

// or

char str\_name[] = {'h', 'e', 'l', 'l', 'o', '\0'};

**Concatenation of strings:**

Combining two or more strings to form a resultant string is called the concatenation of strings.

In C++,we have three ways to concatenate strings. These are as follows.

**Using strcat () function**

To use the strcat () function, we need to include the cstring header file in our program. The strcat () function takes two-character arrays as the input. It concatenates the second array to the end of the first array.

**Syntax:**

strcat (char\_array1, char\_array2);

**Program:**

#include <iostream>

#include <cstring>

using namespace std;

int main()

{

char str1[] = "Gayathri ";

char str2[] = "Chiddapuram";

strcat(str1, str2);

cout << str1 << endl;

return 0;

}

**OUTPUT:**

Gayathri Chiddapuram

**Using append () function**

The append () function appends the first string to the second string. The variables of type string are used with this function.

**Syntax:**

str1.append(str2);

**program:**

#include <iostream>

using namespace std;

int main()

{

string str1 = "Gayathri";

string str2 = "Chiddapuram";

str1.append(str2);

cout << str1 << endl;

return 0;

}

**OUTPUT:**

GayathriChiddapuram

**Using ‘+’ operator**

This is the easiest way to concatenate strings. The + operator takes two (or more) strings and returns the concatenated string.

**Syntax:**

str1 + str2;

**Program:**

#include <iostream>

using namespace std;

int main() {

string str1 = "Gayathri ";

string str2 = "Chiddapuram";

str1 = str1 + str2;

cout << str1 << endl;

return 0;

}

**OUTPUT:**

Gayathri Chiddapuram

**POINTERS**

Pointer is a variable in C++ that holds the address of another variable. They have [data type](https://beginnersbook.com/2017/08/cpp-data-types/) just like variables, for example an integer type pointer can hold the address of an integer variable and an character type pointer can hold the address of char variable.

**Syntax:**

data\_type \*pointer\_name;

**Declaration:**

int \*p, var;

**Assignment**: An integer type pointer can hold the address of another int variable. Here we have an integer variable var and pointer p holds the address of var. To assign the address of variable to pointer we use **ampersand symbol** (&).

P = &var;

**Program:**

#include <iostream>

using namespace std;

int main(){

int \*p;

int arr[]={1, 2, 3, 4, 5, 6};

p = arr;

for(int i=0; i<6;i++){

cout<<\*p<<”\t”;

p++;

}

return 0;

}

**OUTPUT:**

1 2 3 4 5 6

**REFERENCES**

When a variable is declared as a reference, it becomes an alternative name for an existing variable. A variable can be declared as a reference by putting ‘&’ in the declaration.

#include <iostream>

using namespace std;

int main(){

int \*p;

int arr[]={1, 2, 3, 4, 5, 6};

p = arr;

for(int i=0; i<6;i++){

cout<<\*p<<”\t”;

p++;

}

return 0;

}

**OUTPUT:**

1 2 3 4 5 6

**FUNCTION POINTERS**

A **function pointer** in C++, just like variable pointers points to the address of a function, specifically it points to the address of the first line of code in a function.

Function pointer in C++ is very useful as it can be passed as a parameter to a different function, thus making the functionality of **callbacks** easy to implement in C++.

Apart from this, a function pointer in C++ can also be used as a return type for functions in C++.

Diagram

Description automatically generated

**Syntax:**

return\_type (\*pointer\_name) (data\_type1, data\_type2. . .);

**Example:**

void (\*fun\_ptr) (int, double);

**Program**

#include <iostream>

**using** **namespace** std;

**int** add(**int** a , **int** b)

{

**return** a+b;

}

**int** main()

{

**int** (\*funcptr)(**int**,**int**);  // function pointer declaration

 funcptr=add; // funcptr is pointing to the add function

**int** sum=funcptr(5,5);

 cout << "Value of sum is :" <<sum<< std::endl;

**return** 0;

}

**OUTPUT:**

Value of sum is:10

**Benefits of Function Pointer in C++**

* A function pointer in C++ can be passed as a parameter to a function, thus providing the functionality of implementing callback functions.
* A function pointer in C++ allows you to send along instructions for how to perform something.
* By supplying function pointers in C++ as parameters, you may construct flexible functions and libraries that allow the programmer to determine behavior.

**CLASS**

It is a user-defined data type, which holds its own data members and member functions, which can be accessed and used by creating an instance of that class. A C++ class is like a blueprint for an object.

**Example**:

* Consider the Class of cars. There may be many cars with different names and brand but all of them will share some common properties like all of them will have *4 wheels*, *Speed Limit*, *Mileage range* etc. So here, Car is the class and wheels, speed limits, mileage are their properties.
* A Class is a user defined datatype which has data members and member functions.
* Data members are the data variables and member functions are the functions used to manipulate these variables and together these data members and member functions defines the properties and behavior of the objects in a Class.
* In the above example of class *Car*, the data member will be *speed limit*, *mileage* etc. and member functions can be *applied brakes*, *increase in speed* etc.

**Syntax:**

**class name\_of\_class {**

**// access specifiers**

**public:**

**// data members and member functions**

**int variable1;**

**void functioName1(){**

**// function definition here**

**}**

**Example:**

**// name of the class**

**class Student {**

**// access specifier**

**public:**

**// data members variables**

**int roll No, weight, age;**

**string name;**

**// member functions**

**void display Details(){**

**cout << "Roll No: " << roll No << endl;**

**cout << "Name: " << name << endl;**

**}**

**};**

**Object**:

An Object is an instance of a Class. When a class is defined, no memory is allocated but when it is instantiated (i.e., an object is created) memory is allocated.

**Declaring Objects:**

When a class is defined, only the specification for the object is defined; no memory or storage is allocated. To use the data and access functions defined in the class, you need to create objects.

**Example:**

int main(){

// declare Student object s1

Student s1;

// assigning values

s1.rollNo = 1;

s1.weight = 80;

s1.age = 21;

s1.name = "Peter";

// calling function for s1 object

s1.displayDetails();

return 0;

}

**Defining Member Functions**

There are 2 ways to define a member function:

* Inside class definition
* Outside class definition

**Example of function declaration (Both ways):**

#include<iostream>

using namespace std;

// name of the class

class Student

{

// access specifier

public:

int rollNo,weight,age;

string name;

// function declaration & definition inside

void printRollNo()

{

cout<<"Roll No: " << rollNo << endl;

}

// function definition inside

void printName();

};

// function definition outside

void Student::print Name(){

cout << "Name: " << name << endl;

}

int main(){

// declare Student object s1

Student s1;

// assigning values

s1.rollNo = 1;

s1.name = "Deepthi";

s1.printRollNo();

s1.printName();

return 0;

}

**OUTPUT:**

Roll No:1

Name: Deepthi

**Initializer list**:

The **initializer list** is used to directly initialize data members of a class. An initializer list starts after the constructor’s name and its parameters. The list begins with a colon (: ) and is followed by the list of variables that are to be initialized – all of​ the variables are separated by a comma with their values in curly brackets.

**Example:**

The following example uses an initializer list in the default constructor to set the value for the variable value of the class.

#include<iostream>

using namespace std;

class Base

{

private:

int value;

public:

// default constructor

Base(int v):value(v)

{

cout << "Value is " << value;

}

};

int main()

{

Base myobject(10);

return 0;

}

**OUTPUT:**

Value is 10

#### **Delegating Constructor:**

* If the name of the class itself appears as *class-or-identifier* in the member initializer list, then the list must consist of that one-member initializer only; such a constructor is known as the *delegating constructor*, and the constructor selected by the only member of the initializer list is the *target constructor*
* In this case, the target constructor is selected by overload resolution and executed first, then the control returns to the delegating constructor and its body is executed.
* Delegating constructors cannot be recursive.

class Foo

{

public:

Foo(char x, int y) {}

Foo(int y) : Foo('a', y) {} // Foo(int) delegates to Foo(char, int)

};

**INHERITANCE**

* Inheritance is a feature or a process in which, new classes are created from the existing classes. The new class created is called “derived class” or “child class” and the existing class is known as the “base class” or “parent class”. The derived class now is said to be inherited from the base class.
* When we say derived class inherits the base class, it means, the derived class inherits all the properties of the base class, without changing the properties of base class and may add new features to its own. These new features in the derived class will not affect the base class. The derived class is the specialized class for the base class.
* **Sub Class:** The class that inherits properties from another class is called Subclass or Derived Class.
* **Super Class:**The class whose properties are inherited by a subclass is called Base Class or Superclass.

**Modes of Inheritance:**

There are 3 modes of inheritance.

* **Public Mode**: If we derive a subclass from a public base class. Then the public member of the base class will become public in the derived class and protected members of the base class will become protected in the derived class.
* **Protected Mode**: If we derive a subclass from a Protected base class. Then both public members and protected members of the base class will become protected in the derived class.
* **Private Mode**: If we derive a subclass from a Private base class. Then both public members and protected members of the base class will become Private in the derived class.

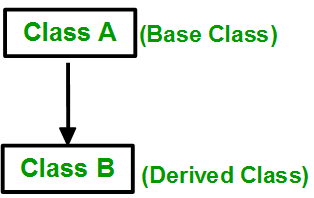
**Types Of Inheritance: -**

* Single inheritance
* Multilevel inheritance
* Multiple inheritance
* Hierarchical inheritance
* Hybrid inheritance

**Types of Inheritance in C++**

**Single Inheritance**:

* In single inheritance, a class is allowed to inherit from only one class. i.e., one subclass is inherited by one base class only.



**Example:**

#include<iostream>  
using namespace std;  
  
// base class  
class Vehicle {  
  public:  
    Vehicle()  
    {  
      cout << "This is a Vehicle\n";  
    }  
};  
  
// sub class derived from a single base classes  
class Car : public Vehicle {  
   public:  
       Car(){  
           cout<<"This is a car";  
       }  
};  
  
// main function  
int main()  
{    
    // Creating object of sub class will  
    // invoke the constructor of base classes  
    Car obj;  
    return 0;  
}

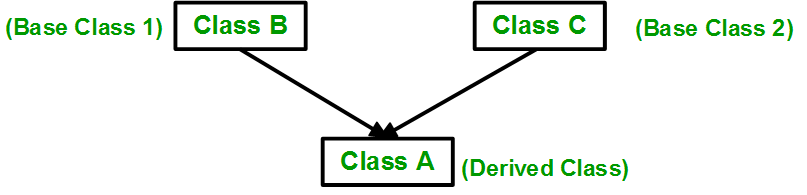
**OUTPUT:**

This is a Vehicle

This is a car

**Multiple Inheritance:**

Multiple Inheritance is a feature of C++ where a class can inherit from more than one class. i.e, one **subclass** is inherited from more than one **base class**.



**Example:**

#include <iostream>  
using namespace std;

// first base class  
class Vehicle {  
public:  
    Vehicle() {   
    cout << "This is a Vehicle\n"; }  
};

 // second base class  
class Four Wheeler {  
public:  
    Four Wheeler()  
    {  
        cout << "This is a 4-wheeler Vehicle\n";  
    }  
};

// sub class derived from two base classes  
class Car : public Vehicle, public Four Wheeler {  
  
};

 int main()  
{  
  Car obj;  
    return 0;  
}

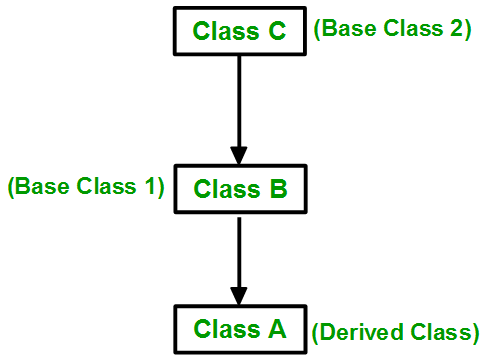
**OUTPUT:**

This is a Vehicle

This is a 4-wheeler Vehicle

**Multilevel Inheritance**:

In this type of inheritance, a derived class is created from another derived class.



**Example:**

#include <iostream>  
using namespace std;

// base class  
class Vehicle {  
public:  
    Vehicle() {   
    cout << "This is a Vehicle\n";  
     }  
};

 // first subclass derived from class vehicle  
class four Wheeler: public Vehicle {  
public:  
    four Wheeler()  
    {  
        cout << "Objects with 4 wheels are vehicles\n";  
    }  
};  
// sub class derived from the derived base class four Wheeler  
class Car : public four Wheeler {  
public:  
    Car() {  
     cout << "Car has 4 Wheels\n";   
     }  
};

 int main()  
{

  Car obj;  
    return 0;  
}

**OUTPUT:**

This is a vehicle

Objects with 4 wheels are vehicles

**Hierarchical Inheritance**:

In this type of inheritance, more than one subclass is inherited from a single base class. i.e., more than one derived class is created from a single base class.



**Example:**

#include <iostream>  
using namespace std;

// base class  
class Vehicle {  
public:  
    Vehicle () {   
    cout << "This is a Vehicle\n";   
    }  
};

// first sub class  
class Car : public Vehicle {  
  
};

// second sub class  
class Bus : public Vehicle {   }  
};

// main function  
int main()  
{  
  
    Car obj1;  
    Bus obj2;  
    return 0;  
}

**OUTPUT:**

This is a Vehicle

This is a Vehicle

**POLYMORPHISM**

* The term "Polymorphism" is the combination of "poly" + "morphs" which means many forms.
* Let's consider a real-life example of polymorphism. A lady behaves like a teacher in a classroom, mother or daughter in a home and customer in a market. Here, a single person behaving differently according to the situation.

**Types:**

* Compile time polymorphism
* Run time polymorphism

Diagram

Description automatically generated

**Compile time polymorphism**:

The overloaded functions are invoked by matching the type and number of arguments. This information is available at the compile time and, therefore, compiler selects the appropriate function at the compile time. It is achieved by function overloading and operator overloading.

**Function overloading:**

* When the function name is overloaded with different jobs then it is called Function overloading.
* In function overloading “Function” name should be same and the arguments should be different.

**Program:**

#include<iostream>

using namespace std;

class A

{

public:

void one()

{

cout<<"Without argument"<<endl;

}

void one(int x)

{

cout<<"Integer argument"<<endl;

}

void one(double y)

{

cout<<"Double argument"<<endl;

}

};

int main()

{

A;

a.one();

a.one(10);

a.one(3.9);

}

**OUTPUT:**

Without argument

Integer argument

Double argument

**Operator overloading:**

* It is an idea of giving special meaning to an existing operator without changing its original meaning.

**Program:**

#include <iostream>

using namespace std;

class TestClass {

private:

int count;

public:

TestClass() : count(5) {}

void operator --() {

count = count - 3;

}

void Display()

{

cout << "Count: " << count; }

};

int main () {

TestClass tc;

--tc;

tc.Display();

return 0;

}

**OUTPUT:**

2

**Run time polymorphism:**

Run time polymorphism is achieved when the object's method is invoked at the run time instead of compile time. It is achieved by method overriding which is also known as dynamic binding or late binding.

**Program:**

#include <iostream>

using namespace std;

class Shape { // base class

public:

virtual void draw (){ // virtual function

cout<<"drawing..."<<endl;

}

};

class Rectangle: public Shape // inheriting Shape class.

{

public:

void draw()

{

cout<<"drawing rectangle..."<<endl;

}

};

class Circle: public Shape // inheriting Shape class.

{

public:

void draw()

{

cout<<"drawing circle..."<<endl;

}

};

int main(void) {

Shape \*s; // base class pointer.

Shape sh; // base class object.

Rectangle rec;

Circle cir;

s=&sh;

s->draw();

s=&rec;

s->draw();

s=&cir;

s->draw();

}

**OUTPUT:**

drawing…

drawing rectangle…

drawing circle…

**DATA ABSTARCTION**

* Data Abstraction is a process of providing only the essential details to the outside world and hiding the internal details.
* Let's take a real-life example of AC, which can be turned ON or OFF, change the temperature, change the mode, and other external components such as fan, swing. But we don't know the internal details of the AC, i.e., how it works internally. Thus, we can say that AC separates the implementation details from the external interface.
* Abstraction can be achieved in two ways:
* Abstraction using classes
* Abstraction using header files.

**Abstraction using classes:**

An abstraction can be achieved using classes. A class is used to group all the data members and member functions into a single unit by using the access specifiers. A class has the responsibility to determine which data member is to be visible outside and which is not.

**Program:**

#include <iostream>

using namespace std;

class Sum {

private: int x, y, z; // private variables

public:

void add() {

cout<<"Enter two numbers: ";

cin>>x>>y;

z= x+y;

cout<<"Sum of two number is: "<<z<<endl;

}

};

int main()

{

Sum sm;

sm.add();

return 0;

}

**OUTPUT:**

Enter two numbers:

3

6

Sum of two numbers is:9

**Abstraction in header files:**

 Another type of abstraction is header file. For example, pow () function available is used to calculate the power of a number without knowing which algorithm function uses to calculate the power. Thus, we can say that header files hide all the implementation details from the user.

**Program:**

#include <iostream>

#include<math.h>

using namespace std;

int main()

{

int n = 4;

int power = 3;

int result = pow(n,power); // pow(n,power) is the power function

std::cout << "Cube of n is : " <<result<< std::endl;

return 0;

}

**OUTPUT:**

Cube of n is:64

**ENCAPSULATION**

* It is one of the key features of object-oriented programming.
* It is also called as information hiding. The sensitive data is hidden from the users.
* It is wrapping up of data members and functions into a single unit. The data of class is not accessible to outside the class, only functions can access data which are wrapped in the class.
* To achieve encapsulation, we must declare data members as private and member functions as public. If you want others to read and modify the value of a private member, you can provide public get and set methods.
* This concept is applicable in the marketing and finance sector, where there is a high demand for security and restricted data access to various departments.

Diagram

Description automatically generated

**Applications:**

* Suppose you have a profile on social networking website, say Facebook. If your profile password is declared as private to make your account safe so that anyone cannot log into your account. So other details of your account or post, you can share can be made public or private. We can see the data hiding concept here.
* Suppose you have an account in the bank. If your balance variable is declared as a public variable in the bank software, your account balance will be known as public, in this case, anyone can know your account balance. So, they declare balance variable as private for making your account safe, so that anyone cannot see your account balance. The person who must see his account balance, will have to access only private members through methods defined inside that class and this method will ask your account holder name or user Id, and password for authentication. Thus, we can achieve security by utilizing the concept of data hiding. This is called Encapsulation

**Features:**

* Protect data from unauthorized users.
* It ensures better control of the data, because we can change one part of code without affecting other parts.
* It helps us make code flexible, which is easy to change and maintain.

**Advantages:**

* The encapsulated code is more flexible and easier to change with new requirements.
* It prevents the other classes from accessing the private fields.
* Encapsulation allows modifying the implemented code without breaking other code that has implemented the code.
* It keeps the data and codes safe from external inheritance. Thus, encapsulation helps to achieve security.
* It improves the maintainability of the application.

**Disadvantages:**

* The length of code Size increases drastically in encapsulation as we need to provide methods with specifiers.
* As code size increases, we need to provide more instructions.
* Encapsulation increases the duration of the program execution.

**Example-1:**

#include <iostream>

using namespace std;

class A {

private:

// Private attribute

int a,b;

public:

void show (){

cout<<” This is show function”;

}

Void Sum (){

cout<<” Enter values:\n”;

cin>>a>>b;

cout<<”Sum= ”<<a+b;

};

int main() {

A ob,ob1;

ob.show();

ob1.Sum();

return 0;

}

**OUTPUT:**

This is show function

Enter values:

1

2

Sum=3

**Example-2:**

#include <iostream>

using namespace std;

class Rectangle {

private: // Variables required for area calculation

int length;

int breadth;

public:

// Setter function for length

void setLength(int len) {

length = len;

}

// Setter function for breadth

void setBreadth(int brth) {

breadth = brth;

}

// Getter function for length

int getLength() {

return length;

}

// Getter function for breadth

int getBreadth() {

return breadth;

}

// Function to calculate area

int getArea() {

return length \* breadth;

}

};

int main() {

Rectangle rectangle1; // Create object of Rectangle class

rectangle1.setLength(8); // Initialize length using Setter function

rectangle1.setBreadth(6); // Initialize breadth using Setter function

cout << "Length = " << rectangle1.getLength() << endl; // Access length using Getter function

cout << "Breadth = " << rectangle1.getBreadth() << endl; // Access breadth using Getter function

cout << "Area = " << rectangle1.getArea(); // Call getArea() function

return 0;

}

**OUTPUT:**

Length=8

Breadth=6

Area=48

**Difference between Data Abstraction and Data Encapsulation**

* Data Abstraction is the property by virtue of which only the essential details are displayed to the user. The trivial or the non-essential units are not displayed to the user.
* Encapsulation is a method to hide the data in a single entity or unit along with a method to protect information from outside.

**ABSTRACT AND DERIVED CLASS**

**Abstract Class:**

* An abstract class is a class that is designed to be specifically used as a base class. The purpose of abstract class is to provide an appropriate base class from which other classes can inherit. A class Which Contains at least one pure virtual Function.
* Abstract classes cannot be used to instantiate objects and serves only as an interface.

**Syntax:**

class A {

public:

virtual void Show () = 0;

};

**Pure virtual function:**

* A pure virtual function is specified by placing=0 in its declaration.
* Pure Virtual functions are virtual functions which have no definition. They Start with virtual keyword and ends with equals to zero.
* If we don’t override the pure virtual function in derived class, then derived class also becomes an abstract class.

**Virtual Function:**

* A virtual function is a member function which is declared within base class and is redefined(overridden) by derived class.
* Functions are declared with a virtual keyword in base class.

**Derived Class:**

* It is a class that is constructed from a base class or an existing class. It tends to acquire all the methods and properties of a base class. The derived class can inherit all members and member functions of a base class.

**Syntax:**

class derived\_classname: Visibility mode base\_classname

{

//members of derived class

}

Ex:

class ABC: public XYZ {  
//members of ABC }

**Example-1:**

#include<iostream>

Class A {

public:

virtual void show ()=0;

void display () {

cout<<” This is base class”;}

};

Class B: public A {

public:

void show () {

cout<<” This is derived class”;}

};

void main () {

B obj;

obj.display();

obj.show();}

(or)

void main () {

A \*p;

B obj;

P=&obj;

p->display();

p->show();}

**OUTPUT:**  
This is base class

This is derived class

* From the above example we can say that from class B object we can access abstract class normal member function and class B member functions.
* We can access abstract class normal member function and member function of inherited class using class A pointer.

**Example-2**

#include <iostream>

using namespace std;

//Base class

class Animal {

public:

virtual void eat () =0;

};

//Derived class

class Cat: public Animal {

public:

void eat () {

cout<<"I am Eating a rat\n";

}

};

//Derived class

class Dog: public Animal {

public:

void eat () {

cout<<"I am Eating food\n";

}

};

void function (Animal\*animal) {

animal->eat ();

}

int main () {

Animal \*ptr;

Cat catobj ;

Dog dogobj;

ptr=&catobj;

function(ptr);

ptr=&dogobj;

function(ptr);

return 0;

}

**OUTPUT:**

I am Eating a rat

I am Eating food

**TYPE CONVERSION IN C++**

A type cast is basically a conversion from one type to another. There are two types of type conversion:

**1.Implicit Type Conversion**:

It is also known as ‘automatic type conversion’. Done by the compiler on its own, without any external trigger from the user. Generally, takes place when in an expression more than one data type is present. In such condition type conversion (type promotion) takes place to avoid loss of data.

All the data types of the variables are upgraded to the data type of the variable with largest data type.

bool -> char -> short int -> int ->

unsigned int -> long -> unsigned ->

long -> float -> double -> long double

It is possible for implicit conversions to lose information, signs can be lost (when signed is implicitly converted to unsigned), and overflow can occur (when long is implicitly converted to float).

**Example:**

#include <iostream>

using namespace std;

int main()

{

int x = 10; // integer x

char y = 'a'; // character c

// y implicitly converted to int. ASCII

// value of 'a' is 97

x = x + y;

// x is implicitly converted to float

float z = x + 1.0;

cout << "x = " << x << endl<< "y = " << y << endl<< "z = " << z << endl;

return 0;

}

**OUTPUT:**

x = 107

y = a

z = 10

**2.Explicit Type Conversion**:

This process is also called type casting and it is user-defined. Here the user can typecast the result to make it of a particular data type.

In C++, it can be done by two ways:

**Converting by assignment:** This is done by explicitly defining the required type in front of the expression in parenthesis. This can be also considered as forceful casting.

**Syntax:**

(type) expression

where *type* indicates the data type to which the result is converted.

**Example:**

#include <iostream>

using namespace std;

int main()

{

double x = 1.2;

// Explicit conversion from double to int

int sum = (int)x + 1;

cout << "Sum = " << sum;

return 0;

}

**OUTPUT:**

Sum = 2

**Conversion using Cast operator:**

A Cast operator is a **unary operator** which forces one data type to be converted into another data type.  
C++ supports four types of casting:

* [Static Cast](https://www.geeksforgeeks.org/static_cast-in-c-type-casting-operators/)
* Dynamic Cast
* [Const Cast](https://www.geeksforgeeks.org/casting-operators-in-c-set-1-const_cast/)
* [Reinterpret Cast](https://www.geeksforgeeks.org/reinterpret_cast-in-cpp/)

**Example:**

#include <iostream>

using namespace std;

int main ()

{

float f = 3.5;

// using cast operator

int b = static\_cast<int>(f);

cout << b;

}

**OUTPUT:**

3

**Advantages of Type Conversion:**

* This is done to take advantage of certain features of type hierarchies or type representations.
* It helps to compute expressions containing variables of different data types

# **TEMPLATES IN C++**

A **template** is a simple yet very powerful tool in C++. The simple idea is to pass data type as a parameter so that we don’t need to write the same code for different data types. For example, a software company may need to sort () for different data types. Rather than writing and maintaining multiple codes, we can write one sort () and pass data type as a parameter.

C++ adds two new keywords to support templates: *‘template’*and *‘typename’*. The second keyword can always be replaced by the keyword ‘class’.

Templates are expanded at compiler time. This is like macros. The difference is, that the compiler does type checking before template expansion. The idea is simple, source code contains only function/class, but compiled code may contain multiple copies of the same function/class.

**Function Templates**:

We write a generic function that can be used for different data types. Examples of function templates are sort (), max (), min (), printArray().

**Example:**

#include <iostream>

using namespace std;

template <typename T> T myMax (T x, T y)

{

return (x > y) ? x : y;

}

int main ()

{

cout << myMax<int> (3, 7) << endl; // Call myMax for int

cout << myMax<double> (3.0, 7.0) << endl;

cout << myMax<char> ('g', 'e’) << endl;

return 0;

}

**OUTPUT**

7

7

g

**Class Templates** like function templates, class templates are useful when a class defines something that is independent of the data type. Can be useful for classes like LinkedList, Binary Tree, Stack, Queue, Array, etc.

// C++ program to demonstrate the use of class template

#include <iostream>

using namespace std;

// Class template

template <class T>

class Number {

private:

// Variable of type T

T num;

public:

Number(T n) : num(n) {} // constructor

T getNum() {

return num;

}

};

int main() {

// create object with int type

Number<int> numberInt(7);

// create object with double type

Number<double> numberDouble(7.7);

cout << "int Number = " << numberInt.getNum() << endl;

cout << "double Number = " << numberDouble.getNum() << endl;

return 0;

}

**OUTPUT:**

**Int number= 7**

**Double number=7.7**

**STL (Standard Template Library)**

The Standard Template Library (STL) is a set of C++ template classes to provide common programming data structures and functions such as lists, stacks, arrays, etc. It is a library of container classes, algorithms, and iterators. It is a generalized library and so, its components are parameterized. Working knowledge of [template classes](https://www.geeksforgeeks.org/templates-cpp/) is a prerequisite for working with STL.

**STL has 4 components:**

* Algorithms
* Containers
* Functions
* Iterators

# **Containers in C++ STL**

A container is a holder object that stores a collection of other objects (its elements). They are implemented as class templates, which allows great flexibility in the types supported as elements.

The container manages the storage space for its elements and provides member functions to access them, either directly or through iterators (reference objects with similar properties to pointers).

**Sequence containers**

Sequence containers implement data structures that can be accessed sequentially.

* [**array:**](https://www.geeksforgeeks.org/array-class-c/)Static contiguous array (class template)
* [**vector:**](https://www.geeksforgeeks.org/vector-in-cpp-stl/)Dynamic contiguous array (class template)
* [**deque:**](https://www.geeksforgeeks.org/deque-cpp-stl/)Double-ended queue (class template)
* [**forward\_list:**](https://www.geeksforgeeks.org/forward-list-c-set-1-introduction-important-functions/)Singly linked list (class template)
* [**list:**](https://www.geeksforgeeks.org/list-cpp-stl/)Doubly linked list (class template)

**Associative containers**

Associative containers implement sorted data structures that can be quickly searched (O(log n) complexity).

* [**Set:**](https://www.geeksforgeeks.org/set-in-cpp-stl/) Collection of unique keys, sorted by keys   
  (class template)
* [**Map:**](https://www.geeksforgeeks.org/map-associative-containers-the-c-standard-template-library-stl/)Collection of key-value pairs, sorted by keys, keys are unique (class template).
* [**multiset:**](https://www.geeksforgeeks.org/multiset-in-cpp-stl/)Collection of keys, sorted by keys (class template)
* [**multimap:**](https://www.geeksforgeeks.org/multimap-associative-containers-the-c-standard-template-library-stl/)Collection of key-value pairs, sorted by keys   
  (class template)

**Unordered associative containers**

Unordered associative containers implement unsorted (hashed) data structures that can be quickly searched (O (1) amortized, O(n) worst-case complexity).

* [**unordered\_set:**](https://www.geeksforgeeks.org/unorderd_set-stl-uses/)Collection of unique keys, hashed by keys. (Class template)
* [**unordered\_map:**](https://www.geeksforgeeks.org/unordered_map-in-stl-and-its-applications/)Collection of key-value pairs, hashed by keys, keys are unique. (Class template)
* [**unordered\_multiset:**](https://www.geeksforgeeks.org/unordered_multiset-and-its-uses/)Collection of keys, hashed by keys (class template)
* [**unordered\_multimap:**](https://www.geeksforgeeks.org/unordered_multimap-and-its-application/)Collection of key-value pairs, hashed by keys (class template)

**Container adapters**

Container adapters provide a different interface for sequential containers.

* [**stack:**](https://www.geeksforgeeks.org/stack-in-cpp-stl/)Adapts a container to provide stack (LIFO data structure) (class template).
* [**queue:**](https://www.geeksforgeeks.org/queue-cpp-stl/)Adapts a container to provide queue (FIFO data structure) (class template).
* [**priority\_queue:**](https://www.geeksforgeeks.org/priority-queue-in-cpp-stl/) Adapts a container to provide priority queue (class template).

**MAPS:**

Maps are[associative containers](https://www.geeksforgeeks.org/containers-cpp-stl/)that store elements in a mapped fashion. Each element has a key value and a mapped value. No two mapped values can have the same key values

**Some basic functions associated with Map:**

* [**begin ()**](https://www.geeksforgeeks.org/mapbegin-end-c-stl/) – Returns an iterator to the first element in the map.
* [**end ()**](https://www.geeksforgeeks.org/mapbegin-end-c-stl/) – Returns an iterator to the theoretical element that follows the last element in the map.
* [**size ()**](https://www.geeksforgeeks.org/mapsize-c-stl/) – Returns the number of elements in the map.
* [**max\_size ()**](https://www.geeksforgeeks.org/map-max_size-in-c-stl/) – Returns the maximum number of elements that the map can hold.
* [**empty ()**](https://www.geeksforgeeks.org/mapempty-c-stl/) – Returns whether the map is empty.
* [**pair insert (keyvalue, mapvalue)**](https://www.geeksforgeeks.org/map-insert-in-c-stl/)– Adds a new element to the map.
* [**erase (iterator position)**](https://www.geeksforgeeks.org/map-erase-function-in-c-stl/) – Removes the element at the position pointed by the iterator.
* [**erase (const g)**](https://www.geeksforgeeks.org/map-erase-function-in-c-stl/)– Removes the key-value ‘g’ from the map.
* [**clear ()**](https://www.geeksforgeeks.org/mapclear-c-stl/) – Removes all the elements from the map.

**Example:**

#include <iostream>

#include <map>

int main()

{

// Create a map of strings to integers

std::map<std::string, int> map;

// Insert some values into the map

map["one"] = 1;

map["two"] = 2;

map["three"] = 3;

// Get an iterator pointing to the first element in the map

std::map<std::string, int>::iterator it = map.begin();

// Iterate through the map and print the elements

while (it != map.end())

{

std::cout << "Key: " << it->first << ", Value: " << it->second << std::endl;

++it;

}

return 0;

}

**OUTPUT:**

Key: one, Value: 1

Key: three, Value: 3

Key: two, Value: 2

**Array:**

* Array classes knows its own size, whereas C-style arrays lack this property. So, when passing to functions, we don’t need to pass size of Array as a separate parameter.
* With C-style array there is more risk of [array being decayed into a pointer](https://www.geeksforgeeks.org/what-is-array-decay-in-c-how-can-it-be-prevented/). Array classes don’t decay into pointers
* Array classes are generally more efficient, lightweight and reliable than C-style arrays.

**Operations on array**:-   
**1. at ()** :- This function is used to access the elements of array.   
**2. get ()** :- This function is also used to access the elements of array. This function is not the member of array class but overloaded function from class tuple.   
**3. operator []** :- This is similar to C-style arrays. This method is also used to access array elements

**Example:**

#include<iostream>

#include<array> // for array, at()

#include<tuple> // for get()

using namespace std;

int main()

{

// Initializing the array elements

array<int,6> ar = {1, 2, 3, 4, 5, 6};

// Printing array elements using at()

cout << "The array elements are (using at()) : ";

for ( int i=0; i<6; i++)

cout << ar.at(i) << " ";

cout << endl;

// Printing array elements using get()

cout << "The array elements are (using get()) : ";

cout << get<0>(ar) << " " << get<1>(ar) << " ";

cout << get<2>(ar) << " " << get<3>(ar) << " ";

cout << get<4>(ar) << " " << get<5>(ar) << " ";

cout << endl;

// Printing array elements using operator[]

cout << "The array elements are (using operator[]) : ";

for ( int i=0; i<6; i++)

cout << ar[i] << " ";

cout << endl;

return 0;

}

**OUTPUT**

The array elements are (using at()) : 1 2 3 4 5 6

The array elements are (using get()) : 1 2 3 4 5 6

The array elements are (using operator[]) : 1 2 3 4 5 6

# **Stack in C++ STL**

Stacks are a type of container adaptors with LIFO (Last in First Out) type of working, where a new element is added at one end (top) and an element is removed from that end only.  Stack uses an encapsulated object of either [vector](https://www.geeksforgeeks.org/vector-in-cpp-stl/)or [deque](https://www.geeksforgeeks.org/deque-cpp-stl/)(by default) or [list](https://www.geeksforgeeks.org/list-cpp-stl/)(sequential container class) as its underlying container, providing a specific set of member functions to access its elements.

[**empty()**](https://www.geeksforgeeks.org/stack-empty-and-stack-size-in-c-stl/) – Returns whether the stack is empty – Time Complexity : O(1)   
[**size()**](https://www.geeksforgeeks.org/stack-empty-and-stack-size-in-c-stl/) – Returns the size of the stack – Time Complexity : O(1)   
[**top()**](https://www.geeksforgeeks.org/stack-top-c-stl/) – Returns a reference to the top most element of the stack – Time Complexity : O(1)   
[**push(g)**](https://www.geeksforgeeks.org/stack-push-and-pop-in-c-stl/) – Adds the element ‘g’ at the top of the stack – Time Complexity : O(1)   
[**pop()**](https://www.geeksforgeeks.org/stack-push-and-pop-in-c-stl/) – Deletes the most recent entered element of the stack – Time Complexity : O(1)

**Example:**

#include <iostream>

#include <stack>

using namespace std;

int main () {

stack<int> stack;

stack.push (21);// The values pushed in the stack should be of the same data which is written during declaration of stack

stack.push(22);

stack.push(24);

stack.push(25);

int num=0;

stack.push(num);

stack.pop();

stack.pop();

stack.pop();

while (!stack.empty()) {

cout << stack.top() <<" ";

stack.pop();

}

}

**OUTPUT:**

22 21

**SMART POINTERS**

Smart pointers are class objects that behave like raw pointers but manage objects that are new and when or whether to delete them – smart pointers automatically delete the object when no longer needed.

Why Smart Pointers?

Pointers are used for accessing the resources which are external to the program like heap memory, using pointers we can make change in original resource.

**Problem with pointers**

#include <iostream>

using namespace std;

class Rectangle {

private:

int length;

int breadth;

};

void fun()

{

// By taking a pointer p and

// dynamically creating object

// of class rectangle

Rectangle\* p = new Rectangle();

}

int main()

{

// Infinite Loop

while (1) {

fun();

}

}

In function *fun*, it creates a pointer that is pointing to the *Rectangle* object. When the function *fun* ends, p will be destroyed as it is a local variable. But the memory it consumed won’t be deallocated because we forgot to use *delete p;* at the end of the function.

As fun is an infinite loop, it will keep creating p, allocate more and more memory but won’t free it which creates the problem called memory **leak.**

Here C++ comes with its own mechanism that is Smart Pointer, in which programmer doesn’t have to worry about any memory leak or deallocating a pointer.

A *Smart Pointer* is a wrapper class over a pointer with an operator like \* and -> overloaded. The objects of the smart pointer class look like normal pointers. But, unlike *Normal Pointers* it can deallocate and free destroyed object memory.

**Implement Smart Pointer**

We will implement smart pointers such that they will release the memory of unused resources.

* Create a class with a pointer, overloaded operators (->, \*) and destructors.
* The destructor will be automatically called when its object goes out of the scope, and automatically the dynamically allocated memory will be deleted.

**Example:**

#include <iostream>

using namespace std;

class SmartPtr {

int\* ptr; // Actual pointer

public:

explicit SmartPtr(int\* p = NULL) { ptr = p; }

// Destructor

~SmartPtr() { delete (ptr); }

// Overloading dereferencing operator

int& operator\*() { return \*ptr; }

};

int main()

{

SmartPtr ptr(new int());

\*ptr = 20;

cout << \*ptr;

// We don't need to call delete ptr: when the object ptr goes out of scope, the destructor for it is automatically

// called and destructor does delete ptr.

return 0;

}

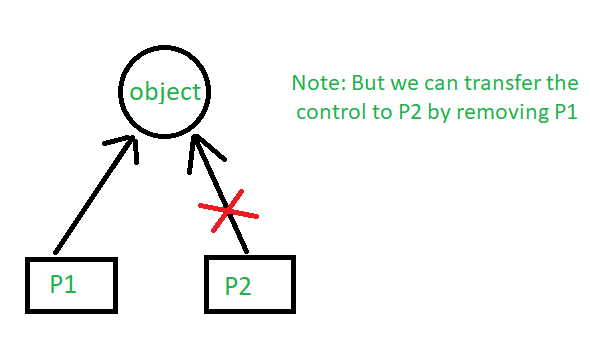
**OUTPUT:**

20

**Types of Smart Pointers**

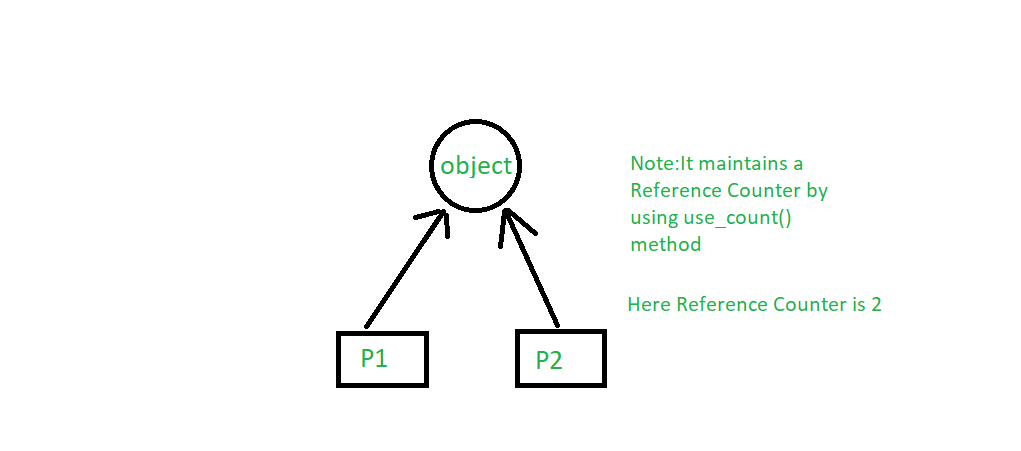
**unique\_ptr:**

*unique\_ptr* stores one pointer only. We can assign a different object by removing the current object from the pointer.



**shared\_ptr:**

In shared\_ptr, more than one object can point to a single pointer at the same instance of time. A reference counter is maintained for denoting the object using the use\_count () method.



**weak\_ptr:**

weak\_ptr is like the shared pointer. The difference is that it does not maintain a reference counter and there is no strong hold of the object on the pointer. This property may result in a deadlock as different objects will try to hold the pointer.

**LAMBDA EXPRESSION**

Lambda expressions are basically inline function or small snippets of code that can be used inside a function or even function call statements. They are not named or reused.

We can declare lambdas as “auto” and use them anywhere in the program.

**Syntax:**

(Capture clause) (parameter\_list) mutable exception ->return\_type {

Method definition;

}

Graphical user interface, application

Description automatically generated

1. capture clause*(Also known as the*lambda-introducer*in the C++ specification.)*
2. parameter list*Optional. (Also known as the*lambda declarator*)*
3. mutable specification*Optional.*
4. exception-specification*Optional.*
5. trailing-return-type*Optional.*
6. lambda body*.*

A **capture clause** of lambda definition is used to specify which variables are captured and whether they are captured by reference or by value.

An **empty capture closure** [], indicates that no variables are used by lambda which means it can only access variables that are local to it.

* The capture closure [&] means the variables are captured by reference.
* The capture closure [=] indicates that the variables are captured by value.

The **mutable** specification enables the body of a lambda expression to modify variables that are captured by value.

Example:

#include <iostream>

#include <string>

using namespace std;

int main()

   {

   // generalized lambda

      auto sum = [](auto a, auto b) {

         return a + b;

    };

     cout <<"Sum(5,6) = "<< sum(5, 6) << endl; // sum of two integers

     cout <<"Sum(2.0,6.5) = "<<sum(2.0, 6.5) << endl; // sum of two float numbers

     cout <<"Sum((string("abc"), string("xyz")) = "<<sum(string("abc"), string("xyz")) << endl;

// sum of two strings

     return 0;

   }

**OUTPUT:**

Sum (5,6) = 11  
Sum (2.0,6.5) = 8.5  
Sum((string(“abc”), string(“xyz”)) = abcxyz

Thus, in this program, we have used a generic lambda sum, which can be used to find the sum of the two objects of any type. Note that we have used ‘auto’ keyword to indicate that the data type of the parameter will be deduced based on the data.

To demonstrate the usage of this lambda, we have used it with three different data types, int, float, and string. From the output, we know that according to the type of data, sum operation is carried out. **For Example,** when we supply string parameters to lambda sum, it concatenates the two strings.

**Note:**

**Inline function**: In C++, we can declare function as inline function, this copies the function to the location of the function call in compile time and may make the program execution faster.

**Syntax:**

Inline returnType functionName(parameter){

Body;

}

**auto keyword:** (used for type deduction) We have used the auto keyword to automatically deduce the return type of the lambda function.

**REGULAR EXPRESSION**

Regular Expression or regexes or regexp as they are commonly called are used to represent a particular pattern of string or text. Regexes are often used to denote a standard textual syntax of a string.

A regular expression or regex is an expression containing a sequence of characters that define a particular search pattern that can be used in string searching algorithms, find or find/replace algorithms, etc. Regexes are also used for input validation.

C++ provides regex support by means of standard library vis the <regex> header.

A regex processor that is used to parse a regex translates it into an internal representation that is executed and matched against a string that represents the text being searched.

**Range specifications:**

**For example,** we can specify a range of lowercase letters from a to z as follows:

**[a-z]**

This will match exactly one lowercase character.

**[A-Za-z0-9]**

The above expression specifies the range containing one single uppercase character, one lowercase character and a digit from 0 to 9.

The brackets ([]) in the above expressions have a special meaning i.e., they are used to specify the range. If you want to include a bracket as part of an expression, then you will need to escape it.

**[\ [0-9]**

The above expression indicates an opening bracket and a digit in the range 0 to 9 as a regex.

**Specification Rules**

[abc] à a, b or c

[^abc] à any character except a, b, c

[a-z] à a to z

[A-Z] à A to Z

[a-zA-Z] à a to z, A to Z

[0-9] à 0 to 9

**Quantifiers**

[ ]? Occurs 0 or 1 time

[]+ Occurs 1 or more times

[]\* Occurs 0 or more times

[]{n} Occurs n times

[]{n, } Occurs n or more time

[]{y,z} Occurs atleast y times but less than z times

**Regex Metacharacter**

\d [0-9]

\D [^0-9]

\w [a-zA-Z\_0-9]

\W [^\w]

**Note:**

Backslash (\) tells computer to treat following character as search character (‘+’, ‘.’, ‘-‘).

**Repeated pattern:**

If we want to match more than one character, we usually specify the “expression modifier” along with the pattern thereby making it a repeated pattern.

An expression modifier can be “+” that suggests matching the occurrence of a pattern one or more times or it can be “\*” that suggests matching the occurrence of a pattern zero or more times.

**For Example, the following expression,**

**[a-z] +** matches the strings like a, aaa, abcd, softwaretestinghelp, etc. Note that it will never match a blank string.

The expression, **[a-z] \*** will match a blank string or any of the above strings. If you want to specify a group of characters to match one or more times, then you can use the parentheses as follows:

**(Xyz)+**

The above expression will match Xyz, XyzXyz, and XyzXyzXyz, etc.

**Program:**

#include <regex>

#include <string>

#include <iostream>

using namespace std;

int main ()

{

string seq = "She sells\_sea shells in the sea shore.";

regex rgx("se[a-z\_] +");

for(sregex\_iterator it(seq.begin(), seq.end(), rgx), it\_end; it != it\_end; ++it )

cout << (\*it) [0] << "\n";

}

**OUTPUT:**

sea

sea

**Functions used in regex:**

**1.regex\_match ()**

This function template is used to match the given pattern. This function returns true if the given expression matches the string. Otherwise, the function returns false.

Example:

#include <iostream>

#include <string>

#include <regex>

using namespace std;

 int main () {

 if (regex\_match ("softwareTesting", regex("(soft)(.\*)") ))

      cout << "string:literal => matched\n";

   const char mystr[] = "SoftwareTestingHelp";

   string str ("software");

   regex str\_expr ("(soft)(.\*)");

   if (regex\_match (str,str\_expr))

      cout << "string:object => matched\n";

   if ( regex\_match ( str.begin(), str.end(), str\_expr ) )

      cout << "string:range(begin-end)=> matched\n";

   cmatch cm;

   regex\_match (mystr,cm,str\_expr);

   smatch sm;

   regex\_match (str,sm,str\_expr);

   regex\_match ( str.cbegin(), str.cend(), sm, str\_expr);

   cout << "String:range, size:" << sm.size() << " matches\n";

   regex\_match ( mystr, cm, str\_expr, regex\_constants::match\_default );

   cout << "the matches are: ";

   for (unsigned i=0; i<sm.size(); ++i) {

      cout << "[" << sm[i] << "] ";

   }

   cout << endl;

 return 0;

}

**OUTPUT:**

string:literal=>matched

string:object=>matched

string:range(begin-end)=>matched

String:range,size:3 matches

the matches are:[software] [soft] [ware]

2.**regex\_search ()**

The function regex\_search () is used to search for a pattern in the string that matches the regular expression.

**Example**

#include <iostream>

#include <regex>

#include<string.h>

using namespace std;

   int main() {

    //string to be searched

    string mystr = "She sells\_sea shells in the sea shore";

    // regex expression for pattern to be searched

    regex regexp("s[a-z\_]+");

    // flag type for determining the matching behavior (in this case on string objects)

     smatch m;

    // regex\_search that searches pattern regexp in the string mystr

    regex\_search(mystr, m, regexp);

    cout<<"String that matches the pattern:"<<endl;

    for (auto x : m)

        cout << x << " ";

    return 0;

}

**OUTPUT:**

String that matches the pattern:

sells\_sea

**3.regex\_replace ()**

The function regex\_replace () is used to replace the pattern matching to a regular expression with a string.

Example:

#include <iostream>

#include <string>

#include <regex>

#include <iterator>

using namespace std;

int main()

{

    string mystr = "This is software testing Help portal \n";

    cout<<"Input string: "<<mystr<<endl;

    // regex to match string beginning with 'p'

    regex regexp("p[a-zA-z]+");

    cout<<"Replace the word 'portal' with word 'website' : ";

    // regex\_replace () for replacing the match with the word 'website'

    cout << regex\_replace (mystr, regexp, "website");

    string result;

    cout<<"Replace the word 'website' back to 'portal': ";

    // regex\_replace() for replacing the match back with 'portal'

    regex\_replace(back\_inserter(result), mystr.begin(), mystr.end(),  regexp,  "portal");

    cout << result;

   return 0;

}

**OUTPUT:**  
Input string: This is software testing Help portal

Replace the word 'portal' with word 'website' : This is software testing Help website

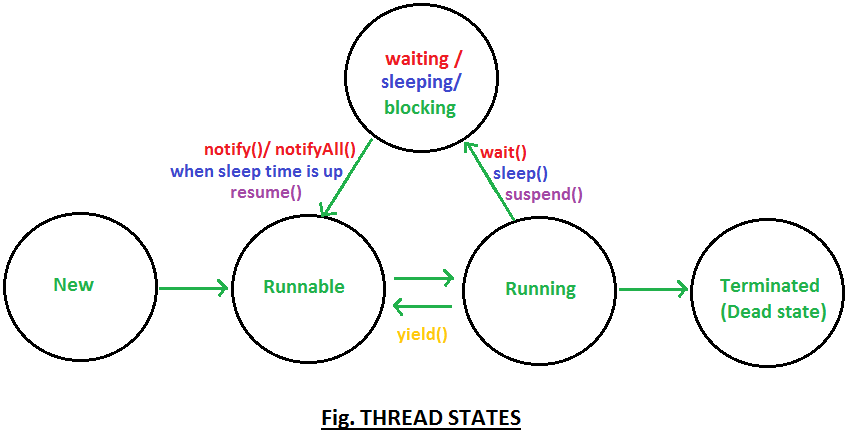
Replace the word ' website ' back to ‘portal' : This is software testing Help portal

**THREADS**

Thread-based multitasking deals with the concurrent execution of pieces of the same program. A multithreaded program contains two or more parts that can run concurrently.

Each part of such a program is called a thread, and each thread defines a separate path of execution.

**THREAD STATES**



**New**

When instance of thread is created using new operator it is in new state, but the start () method has not been invoked on the thread yet, thread is not eligible to run yet.

Thread object is considered alive, but thread is not alive yet.

**Runnable:**

* When start () method is called on thread it enters runnable state.
* As soon as Thread enters runnable state it is eligible to run, but not running. (Thread scheduler has not scheduled the Thread execution yet, Thread has not entered in run () method yet)
* A thread first enters the runnable state when the start () method is invoked, but a thread can also return to the runnable state after either running or coming back from a
* blocked, waiting, or sleeping state.
* Thread is considered alive in runnable state.
* Thread is in Runnable pool.

**Running**

Thread scheduler selects thread to go from runnable to running state. In running state Thread starts executing by entering run () method.

Thread scheduler selects thread from the runnable pool on basis of priority, if priority of two threads is same, threads are scheduled in unpredictable manner. Thread scheduler behavior is completely unpredictable.

When threads are in running state, yield() [method](http://www.javamadesoeasy.com/2015/03/yield-method-in-threads-8-key-features.html) can make thread to go in Runnable state.

**waiting/blocked/sleeping:**

In this state a thread is not eligible to run.

Thread is still alive, but currently it’s not eligible to run. In other words.

* How can Thread go from running to waiting state?

  By calling wait () [method](http://www.javamadesoeasy.com/2015/03/wait-and-notify-methods-definition-8.html) thread go from running to waiting state. In waiting state, it will wait for other threads to release object monitor/lock.

* How can Thread return from waiting to runnable state?

  Once notify() or notifyAll() [method](http://www.javamadesoeasy.com/2015/03/difference-between-notify-and-notifyall.html) is called object monitor/lock becomes available and thread can again return to runnable state.

* How can Thread go from running to sleeping state?

  By calling sleep () [method](http://www.javamadesoeasy.com/2015/03/sleep-method-in-threads-10-key-features.html) thread go from running to sleeping state. In sleeping state, it will wait for sleep time to get over.

* How can Thread return from sleeping to runnable state?

  Once specified sleep time is up thread can again return to runnable state.

**Suspend ()**

Suspend() [method](http://www.javamadesoeasy.com/2015/03/using-suspend-and-resume-method-in.html) can be used to put thread in waiting state and resume() [method](http://www.javamadesoeasy.com/2015/03/using-suspend-and-resume-method-in.html) is the only way which could put thread in runnable state.

Thread also may go from running to waiting state if it is waiting for some I/O operation to take place. Once input is available thread may return to running state.

**Terminated (Dead)**

A thread is considered dead when its run() method completes.

Once thread is dead it cannot be started again doing so will throw runtime Exception i.e., IllegalThreadStateException.

destroy () method puts thread directly into dead state.

**Thread Termination:**

Thread terminates when:

* It reaches the end of the start\_routine.
* It calls return.

**Note:**

* The thread releases its stack during termination
* Return value
* It is not possible to obtain return code from thread
* If you need to return a value you have to use... hmm... no, wait for next week ;-)

**Joining Threads**

**Diagram

Description automatically generated**

● void thread.join();

* The function waits for the thread to terminate.
* It is not possible to join one thread more than once.

● bool thread.joinable() - checks if it is possible to join the thread.

* After the thread was terminated, the internal data are stored for further usage.
* The thread.join() function reads this data to provide status information about terminated thread. Afterwards, the function wipes the date out.
* If the thread.join() function is not called we need to let system know that we do not care about the thread and it can release the data.
* It can cause a serious memory leak problem when huge number of threads is used or each thread returns huge structure if those data are not wiped out.

**Detaching Threads**

**void thread.detach();**

The function marks the thread identified by thread as detached. When a detached thread terminates, its resources are automatically released back to the system without the need for another thread to join with the terminated thread.

**Program:**

#include <iostream>

#include <thread>

#include <vector>

using namespace std;

int counter = 0;

void counterThread ()

{

for (int i = 0; 1 < 10000000; i++)

counter++;

return;

}

int main()

{

vector <thread> threads;

for (int i = 0; i < 4; i++)

threads.push\_back(thread (counterThread));

for (int i = 0; i < 4; i++)

threads [i].join();

cout << counter << endl;

return 0;

}

**Creation of threads using lambda expressions.**

**Example:**

#include<iostream>

#include<thread>

using namespace std;

/\*void threadFn(int & value, int v2) {

cout<<"I am inside a thread function"<<endl;

cout<<"value =>" <<value++<<endl;

}\*/

int main()

{

int localvalue = 100;

int value2 = 200;

thread t1 {[&](){

cout<<"I am inside a thread function"<<endl;

cout<<"value =>" <<localvalue++<<endl;

cout<<"value =>" <<value2--<<endl;

}};

t1.join();

cout<<"value in Main thread ==>" <<localvalue<<endl;

cout<<"value in Main thread ==>" <<value2<<endl;

return 0;}

OUTPUT:  
I am inside a thread function

value=>100

value=>200

value in Main thread==>101

value in Main thread==>199

**Race Condition**

* The problem is that many operations “take time” and can be “interrupted” by other threads attempting to modify the same data.
* This is called a race condition: the result depends on the precise order in which the instructions are executed.
* Unless Thread 0 completes its update before Thread 1 (or vice versa) we get an incorrect result.
* This issue is addressed using mutexes (mutual exclusion).
* They ensure that certain common pieces of data are accessed and modified by a single thread.

**MUTEX**

* A mutex can only be in two states: locked or unlocked.
* Once a thread locks a mutex:
* Other threads attempting to lock the same mutex are blocked.
* Only the thread that initially locked the mutex can unlock it.
* This allows to protect regions of code
* Typical mutex workflow:
* Create and initialize a mutex variable
* Several threads attempt to lock the mutex
* Only one succeeds and that thread owns the mutex
* The owner thread performs some set of actions
* The owner unlocks the mutex
* Another thread acquires the mutex and repeats the process
* The mutex should be destroyed at the end.
* #include<mutex>

–Include the header file with mutex interface

* void mutex.lock()

– Locks a mutex; blocks if another thread has locked this mutex and owns it.

* void mutex.unlock()

– Unlocks mutex; after unlocking, other threads get a chance to lock the mutex.

* bool mutex.try\_lock()

– Tries to lock the mutex. Returns immediately. On successful lock acquisition returns true, otherwise returns false.

**Program to demonstrate Mutex locks:**

#include<iostream>

#include<mutex>

#include<thread>

using namespace std;

void ThreadFn( mutex & mtx) {

lock\_guard<mutex> lock(mtx);

cout<<"I locked the Mutex..."<<endl;

this\_thread::sleep\_for(chrono::seconds(5));

}

int main()

{

mutex mtx;

thread th { ThreadFn, ref(mtx)};

this\_thread::sleep\_for(chrono::seconds(1));

unique\_lock<mutex> lock(mtx);

cout<<"I am inside the Main Thread..."<<endl;

lock.unlock();

lock.lock();

th.join();

return 0;

}

OUTPUT:

I locked the Mutex..."

I am inside the Main Thread

**DEADLOCKS**

Four conditions that must hold for a deadlock to be possible:

* Mutual exclusion: In general, this condition cannot be disallowed.
* Hold And Wait: The hold and wait condition can be prevented by requiring that a process request all its required resources at one time, and blocking the process until all requests can be granted simultaneously
* No preemption: One solution is that if a process holding certain resources is denied a  further request, the process must release its unused resources and  request them again, together with the additional resource.
* Circular Wait:The circular wait condition can be prevented by defining a linearordering of resource types (e.g. Directed Acyclic Graph). If a process has been allocated resource of type R, then it may subsequentlyrequest only those resources of types following R in the ordering.

If thread A is executing and isn’t holding mutex lock 1 yet and thread B acquires mutex lock 2, neither of the threads can continue past the second lock acquisition:

#include <iostream>

#include <boost/thread/thread.hpp>

#include <boost/thread/mutex.hpp>

boost::mutex mutex1, mutex2;

**void** ThreadA()

{

    // Creates deadlock problem

    mutex2.lock();

    std::cout << "Thread A" << std::endl;

    mutex1.lock();

    mutex2.unlock();

    mutex1.unlock();

}

**void** ThreadB()

{

    // Creates deadlock problem

    mutex1.lock();

    std::cout << "\nThread B" << std::endl;

    mutex2.lock();

    mutex1.unlock();

    mutex2.unlock();

}

**void** ExecuteThreads()

{

    boost::**thread** t1( ThreadA );

    boost::**thread** t2( ThreadB );

    t1.join();

    t2.join();

    std::cout << "Finished" << std::endl;

}

**int** main()

{

    ExecuteThreads();

**return** 0;

}

**OUTPUT:**

Thread B

Thread A

**Avoiding Deadlocks**

#include <iostream>

#include <boost/thread/thread.hpp>

#include <boost/thread/mutex.hpp>

boost::mutex mutex1, mutex2;

**void** ThreadA()

{

    // Solves deadlock problem

    mutex1.lock();

    std::cout << "Thread A" << std::endl;

    mutex2.lock();

    mutex2.unlock();

    mutex1.unlock();

}

**void** ThreadB()

{

    // Solves deadlock problem

    mutex1.lock();

    std::cout << "Thread B" << std::endl;

    mutex2.lock();

    mutex1.unlock();

    mutex2.unlock();

}

**void** ExecuteThreads()

{

    boost::**thread** t1( ThreadA );

    boost::**thread** t2( ThreadB );

    t1.join();

    t2.join();

    std::cout << "Finished" << std::endl;

}

**int** main()

{

    ExecuteThreads();

**return** 0;

}

**OUTPUT:**

Thread A

Thread B

Finished

**SEMAPHORES**

Semaphores are compound data types with two fields one is a non-negative integer S.V and the second is Set of processes in a queue S.L. It is used to solve critical section problems, and by using two atomic operations, it will be solved. In this, wait and signal that is used for process synchronization.

Let’s go through the stages of the process that comes in its lifecycle. This will help in understanding semaphore.

* **Running**   
  It states that the Process in execution.
* **Ready**  
  It states that the process wants to run.
* **Idle**   
  The process runs when no processes are running
* **Blocked**   
  The processes not ready not a candidate for a running process. It can be awakened by some external actions.
* **Inactive**   
  The initial state of the process. The process is activated at some point and becomes ready.
* **Complete**   
  When a process executes its final statement

**Initialization of semaphore**

Semaphore S must be initialized with a value of S.V. > 0 and with empty S.L.

**Types of semaphores**

**Type-1**  
**General semaphore:**

A semaphore whose integer component can take arbitrary non-negative values of S.L. these are called General Semaphore. They are kind of weak semaphore.

**Type-2**

**Binary semaphore:**

A semaphore whose integer component S.L. takes only the values 0 and 1 is called a binary semaphore. This is also known as “mutex” which stands for mutual exclusion.

**SIGNALS**

Signals are the interrupts that force an OS to stop its ongoing task and attend the task for which the interrupt has been sent. These interrupts can pause service in any program of an OS. This signal () function is provided by the signal library and is used to trap unexpected interrupts or events.

**Syntax:**

Signal (registered signal, signal handler)

The first argument is an integer, representing the signal number and second is the pointer to a signal handling function. We must keep in mind that the signal that we would like to catch must be registered using a signal function and it must be associated with a signal handling function. The signal handling function should be of the void type.

**Program to demonstrate the signal () function**

#include <csignal>

#include <iostream>

using namespace std;

void signal\_handler(int signal\_num)

{

cout << "The interrupt signal is (" << signal\_num<< "). \n";

exit(signal\_num);

}

int main()

{

signal(SIGABRT, signal\_handler);

while (true)

cout << "Hello…" << endl;

return 0;

}

**OUTPUT:**

Hello…

Hello…

Hello…

Hello…

Hello…

* Hello will be printed infinite times

**DESIGN PATTERNS**

**Design patterns** are typical solutions to common problems in software design. Each pattern is like a blueprint that you can customize to solve a particular design problem in your code.

## **What does the pattern consist of?**

Most patterns are described very formally so people can reproduce them in many contexts. Here are the sections that are usually present in a pattern description:

* **Intent** of the pattern briefly describes both the problem and the solution.
* **Motivation** further explains the problem and the solution the pattern makes possible.
* **Structure** of classes shows each part of the pattern and how they are related.
* **Code example** in one of the popular programming languages makes it easier to grasp the idea behind the pattern.

# **Why should I learn patterns?**

* Design patterns are a toolkit of **tried and tested solutions** to common problems in software design. Even if you never encounter these problems, knowing patterns is still useful because it teaches you how to solve all sorts of problems using principles of object-oriented design.
* Design patterns define a common language that you and your teammates can use to communicate more efficiently. You can say, “Oh, just use a Singleton for that,” and everyone will understand the idea behind your suggestion. No need to explain what a singleton is if you know the pattern and its name.

# **Classification of patterns**

Design patterns differ by their complexity, level of detail and scale of applicability to the entire system being designed. I like the analogy to road construction: you can make an intersection safer by either installing some traffic lights or building an entire multi-level interchange with underground passages for pedestrians.

The most basic and low-level patterns are often called idioms. They usually apply only to a single programming language.

The most universal and high-level patterns are architectural patterns. Developers can implement these patterns in virtually any language. Unlike other patterns, they can be used to design the architecture of an entire application.

In addition, all patterns can be categorized by their intent, or purpose. This book covers three main groups of patterns:

* **Creational patterns** provide object creation mechanisms that increase flexibility and reuse of existing code.
* **Structural patterns** explain how to assemble objects and classes into larger structures, while keeping these structures flexible and efficient.
* **Behavioral patterns** take care of effective communication and the assignment of responsibilities between objects.

# **Singleton Design Pattern**

## **Intent**

**Singleton** is a creational design pattern that lets you ensure that a class has only one instance, while providing a global access point to this instance.

It comes under Creational design pattern. It is a way of creating a single object that is shared amongst a bunch of different resources throughout your application without having to recreate that object or losing the information.

It ensures a class has only one instance and provide a global point of access to it.

The singleton pattern is one of the simplest design patterns. Sometimes we need to have only one instance of our class for example a single DB connection shared by multiple objects as creating a separate DB connection for every object may be costly. Similarly, there can be a single configuration manager or error manager in an application that handles all problems instead of creating multiple managers.

Let’s see various design options for implementing such a class. If you have a good handle on static class variables and access modifiers this should not be a difficult task.

**Method 1: Classic Implementation**

// Classical Java implementation of singleton

// design pattern

class Singleton

{

private static Singleton obj;

// private constructor to force use of

// getInstance() to create Singleton object

private Singleton() {}

public static Singleton getInstance()

{

if (obj==null)

obj = new Singleton();

return obj;

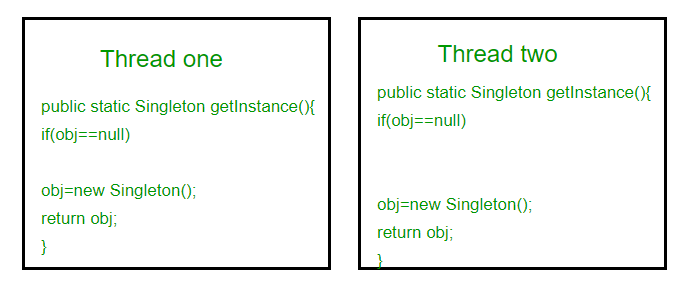
}

}

Here we have declared getInstance () static so that we can call it without instantiating the class. The first time getInstance () is called it creates a new singleton object and after that it just returns the same object.

Note that Singleton obj is not created until we need it and call getInstance () method. This is called lazy instantiation.

The main problem with above method is that it is not thread safe. Consider the following execution sequence.



This execution sequence creates two objects for singleton. Therefore this classic implementation is not thread safe.

   
**Method 2: make getInstance() synchronized**

// Thread Synchronized Java implementation of

// singleton design pattern

class Singleton

{

private static Singleton obj;

private Singleton() {}

// Only one thread can execute this at a time

public static synchronized Singleton getInstance()

{

if (obj==null)

obj = new Singleton();

return obj;

}

}

Here using synchronized makes sure that only one thread at a time can execute getInstance(). 

The main disadvantage of this is method is that using synchronized every time while creating the singleton object is expensive and may decrease the performance of your program.  

However if performance of getInstance() is not critical for your application this method provides a clean and simple solution.

**Method 3: Eager Instantiation**

// Static initializer based Java implementation of

// singleton design pattern

class Singleton

{

private static Singleton obj = new Singleton();

private Singleton() {}

public static Singleton getInstance()

{

return obj;

}

}

**Implementation of Singleton Pattern:**

**Diagram

Description automatically generated**

**Class Diagram**

* Declare a private constructor to prevent instantiation of a singleton class from other classes.
* Create a private static instance of that class so that it gets memory only once.
* Create a public static getInstance () method that returns an instance of the class and offers the global point of access to the Singleton object.

**Advantages:**

* Save memory-because class instance creates only one time, it results in memory saving.
* Flexibility to do all sort of things with that object.

**Disadvantages:**

* The global nature leads to dependency hiding.
* It can be difficult to unit test the code.
* It can lead to tightly coupled code.
* If the single Instance of the object becomes corrupted, the entire system is compromised.

**Applications:**

* It is mostly used in multi-threaded applications, logging, thread pool, configuration settings, caching, etc.

**Example:**

#include<iostream>

#include<string>

using namespace std;

class GameSetting {

static GameSetting\* \_instance;

int \_height;

int \_width;

int \_brightness;

GameSetting():\_width(786),\_height(1300),\_brightness (75) {

}

public:

static GameSetting\* getInstance () {

if(\_instance==NULL)

\_instance=new GameSetting();

return \_instance;

}

void setWidth (int width) {

\_width=width;

}

void setHeight (int height) {

\_height=height;

}

void setBrightness (int brightness) {

\_brightness=brightness;

}

int getWidht () {

return \_width;

}

int getHeight () {

return \_height;

}

int getBrightness () {

return \_brightness;

}

void displaySettings () {

cout<<"brightness:"<<\_brightness<<endl;

cout<<"width:"<<\_width<<endl;

cout<<"height:"<<\_height<<endl;

}

};

GameSetting \*GameSetting::\_instance=NULL;

void some Function () {

GameSetting \*setting=GameSetting::getInstance ();

cout<<” display latest values:\n”;

setting->displaySettings ();

}

int main () {

GameSetting \*setting=GameSetting::getInstance ();

setting->displaySettings ();

setting->setBrightness (100);

someFunction ();

return 0;

}

**OUTPUT:**

brightness:75

width:786

height:1300

display latest values:

brightness:100

width:786

height:1300

* From the above program we can say that wherever you will go, we will get the same instance all the time. We always want same single object to be stored all the places wherever we go in our program.

# **Factory Method**

It is also known asVirtual Constructor.

## **Intent**

**Factory Method** is a creational design pattern that provides an interface for creating objects in a superclass but allows subclasses to alter the type of objects that will be created.

## **Implementation of Factory Design Pattern:**



**CLASS DIAGRAM**

* The **Product** declares the interface, which is common to all objects that can be produced by the creator and its subclasses.
* **Concrete Products** are different implementations of the product interface.
* The **Creator** class declares the factory method that returns new product objects. It’s important that the return type of this method matches the product interface.
* You can declare the factory method as abstract to force all subclasses to implement their own versions of the method. As an alternative, the base factory method can return some default product type.

Note, despite its name, product creation is **not** the primary responsibility of the creator. Usually, the creator class already has some core business logic related to products. The factory method helps to decouple this logic from the concrete product classes. Here is an analogy: a large software development company can have a training department for programmers. However, the primary function of the company is still writing code, not producing programmers.

**Concrete Creators** override the base factory method, so it returns a different type of product.

Note that the factory method doesn’t have to **create** new instances all the time. It can also return existing objects from a cache, an object pool, or another source.

**Example code:**

#include <iostream>

using namespace std;

enum toolType

{net, bolt, cutter, screw\_driver, rinch };

class Tool

{

public:

virtual void TOOLS () = 0;

};

class Net:public Tool

{

public:

void TOOLS ()

{

cout << "net" << endl;

}

};

class Bolt:public Tool

{

public:

void TOOLS ()

{

cout << "bolt" << endl;

}

};

class Cutter:public Tool

{

public:

void TOOLS ()

{

cout << "cutter" << endl;

}

};

class Screw\_driver:public Tool

{

public:

void TOOLS ()

{

cout << "screw\_driver" << endl;

}

};

class Rinch:public Tool

{

public:

void TOOLS ()

{

cout << "rinch" << endl;

}

};

class factory

{

private:

enum toolType TypeOfTool;

static Tool \*obj;

public:

factory ()

{

cout << "Const : " << endl;

}

static Tool \*getIns (toolType type)

{

if (type == net)

{

obj = new Net ();

}

else if (type == bolt)

{

obj = new Bolt ();

}

else if (type == cutter)

{

obj = new Cutter ();

}

else if (type == screw\_driver)

{

obj = new Screw\_driver ();

}

else if (type == rinch)

{

obj = new Rinch ();

}

return obj;

}

};

Tool \*factory::obj = nullptr;

int main ()

{

Tool \*

Cobj = factory::getIns (rinch);

Cobj->TOOLS ();

return 0;

}

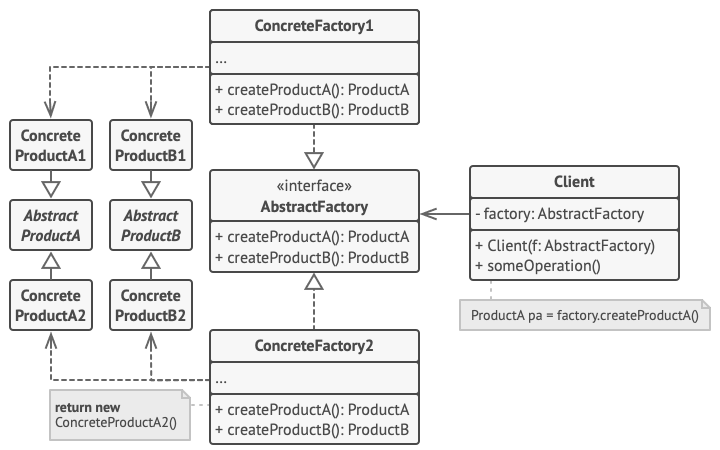
**OUTPUT:**rinch

# **Abstract Factory**

## **Intent**

**Abstract Factory** is a creational design pattern that lets you produce families of related objects without specifying their concrete classes.

**Implemetation of Abstract Factory Design Pattern**



**CLASS DIAGRAM**

* **Abstract Products** declare interfaces for a set of distinct but related products which make up a product family.
* **Concrete Products** are various implementations of abstract products, grouped by variants. Each abstract product (chair/sofa) must be implemented in all given variants (Victorian/Modern).
* The **Abstract Factory** interface declares a set of methods for creating each of the abstract products.
* **Concrete Factories** implement creation methods of the abstract factory. Each concrete factory corresponds to a specific variant of products and creates only those product variants.
* Although concrete factories instantiate concrete products, signatures of their creation methods must return corresponding abstract products. This way the client code that uses a factory doesn’t get coupled to the specific variant of the product it gets from a factory. The **Client** can work with any concrete factory/product variant, if it communicates with their objects via abstract interfaces.

**Example :**

**Client Code:**

#ifndef \_\_io\_\_

#define \_\_io\_\_

#include <iostream>

using namespace std;

#endif

#include "CarFactory.cpp"

#define SIMPLE\_CAR 1

//#define LUXURY\_CAR 1

int main() {

#ifdef SIMPLE\_CAR

CarFactory\* factory = new SimpleCarFactory;

#elif LUXURY\_CAR

CarFactory\* factory = new LuxuryCarFactory;

#endif

Car \*car = factory->buildWholeCar();

car->printDetails();

return 0;

}

**Carfactory Code:**

#ifndef \_\_io\_\_

#define \_\_io\_\_

#include <iostream>

using namespace std;

#endif

#include "CarFactory.cpp"

#define SIMPLE\_CAR 1

//#define LUXURY\_CAR 1

int main() {

#ifdef SIMPLE\_CAR

CarFactory\* factory = new SimpleCarFactory;

#elif LUXURY\_CAR

CarFactory\* factory = new LuxuryCarFactory;

#endif

Car \*car = factory->buildWholeCar();

car->printDetails();

return 0;

}

**Car Code:**

#ifndef \_\_io\_\_

#define \_\_io\_\_

#include <iostream>

using namespace std;

#endif

class Tire {

protected:

string name;

int presure;

public:

Tire(string n, int presure):name(n),presure(presure){};

string getName(){return name;}

int getPresure(){return presure;}

};

class SimpleTire : public Tire{

public:

SimpleTire():Tire("SimpleTire",75) {}

};

class LuxuryTire : public Tire{

public:

LuxuryTire():Tire("LuxuryTire",100) {}

};

class Body {

protected:

string name;

int strength;

public:

Body(string n, int strength):name(n),strength(strength) {}

string getName(){return name;}

int getStrength() {return strength;}

};

class SimpleBody : public Body{

public:

SimpleBody():Body("SimpleBody",75) {}

};

class LuxuryBody : public Body{

public:

LuxuryBody():Body("LuxuryBody",100) {}

};

class Car {

protected:

string name;

Tire \*tire;

Body \*body;

public:

Car(string n):name(n) {}

void setTire(Tire\* t) { tire = t;}

void setBody(Body\* b) { body = b;}

void printDetails(){

cout << endl << "Car: " << name << endl;

cout << "Tire: " << tire->getName() << " Presure: " << tire->getPresure() << endl;

cout << "Body: " << body->getName() << " Strength: " << body->getStrength() << endl << endl;

}

};

**Observer Design pattern:**

It is a behavioral design pattern that allows some objects to notify other objects about changes in their state.

The observer pattern provides a way to subscribe and unsubscribe to and from the events for any object that implements a subscriber interface.

Observers’ patterns intent is to define a one-to-many dependency between objects so that when one object changes state, all its dependents are notified and updated automatically.

**Implementation of observer Pattern:**

**Diagram

Description automatically generated**

**Class Diagram**

* Here Observer and Subject are interfaces
* All observers who need the data need to implement observer interface.
* notify () method in observer interface defines the action to be taken when the subject provides it data.
* The subject maintains an observer Collection which is simply the list of currently registered(subscribed) observers.
* registerObserver (Subscriber Info) and unregisterObserver (Unsubscribe Info) are methods to add and remove observers respectively.
* notifyObservers () is called when the data is changed, and the observers need to be supplied with new data.

**Advantages:**

* There is no need to modify Subject to add or remove observers.
* We can reuse subject and observer classes independently of each other.

**Disadvantages:**

* The main disadvantage of the observer design pattern that subscribers are notified in random order.
* There is also a memory leakage problem in the observer design pattern because of the observer's explicit register and unregistering.

**Applications:**

* Mobile application notification system in which whenever new update comes, we will get notification.
* Social media, email subscription in which you have the option to follow or subscribe and you receive latest notification.
* All users of an app on play store gets notified if there is an update.

**When to Use:**

* When one objects’ state changes and must be reflected in another object without keeping the objects tightly coupled.

**Example:**

#include <iostream>

#include <string>

#include <vector>

#include <list>

#include <algorithm>

using namespace std;

class IObserver

{

public:

virtual void newUpdateShow (float price) = 0;

};

class Shop: public IObserver

{

std:string name;

float price;

public:

Shop (std::string name)

{

this->name = name;

}

void newUpdateShow (float price)

{

this->price = price;

std ::cout << "Price at "<< name << " is now "<< price << "\n";

}

};

class ShopOperationSubject

{

std::vector<Shop\*> list;

std::vector<Shop\*>::iterator itr;

public:

void subscribeInfo (Shop \*shop)

{

list.push\_back(shop);

}

void unSubscribeInfo (Shop \*shop)

{

list.erase(std::remove(list.begin(), list.end(), shop), list.end());

}

void notifyInfo (float price)

{

for (vector<Shop\*>::const\_iterator iter = list.begin(); iter != list.end(); ++iter)

{

if (\*iter != 0)

{

(\*iter)->newUpdateShow(price);

}

}

}

};

class UpdateProductInfo : public ShopOperationSubject

{

public:

void ChangePrice (float price)

{

notifyInfo(price);

}

};

int main()

{

cout<<"Hello World" <<endl;

UpdateProductInfo product;

Shop shop1("Shop 1");

Shop shop2("Shop 2");

product.subscribeInfo(&shop1);

product.subscribeInfo(&shop2);

product.ChangePrice(10);

product.unSubscribeInfo(&shop2);

cout<<"Now shop2 has unsubscribed "<<endl;

product.ChangePrice(20);

return 0;

}

**OUTPUT:**

Hello World

Price at Shop 1 is now 10

Price at Shop 2 is now 10

Now shop2 has unsubscribed

Price at Shop 1 is now 20

* From the above program we can say that whenever product price will change all the shops which are subscribed should get informed.

**LINKED LIST**

It is a non-sequential collection of data items. It is a dynamic data structure. For every data item in a linked list, there is an associated pointer that would give the memory location of the next data item in the linked list.

Linked List can be defined as collection of objects called nodes that are randomly stored in the memory.

Simply it is linear data structure includes a series of connected nodes. Each node stores the data and the address of the next node.

Diagram

Description automatically generated

* The address of first node is called head.
* The last node of the list contains pointer to the null.

**Advantages of Linked lists:**

* These are dynamic data structures. I.e., they can grow or shrink during the execution of a program.
* They have efficient memory utilization. Here, memory is not preallocated. Memory is allocated whenever it is required, and it is de-allocated when it is no longer needed.
* Insertion and deletion are easier and efficient. Linked lists provide flexibility in inserting a data item at a specified position and deletion of the data item from the given position.
* Many complex applications can be easily carried out with linked lists.

**Disadvantages:**

* It consumes more space because every node requires an additional pointer to store address of the next node.
* Searching a particular element in list is difficult and time consuming.

**Why use linked list over arrays:**

* Array is the most common data structure used to store collections of elements.
* Arrays are convenient to declare and provide the easy syntax to access any element by its index number.
* Once the array is set up, access to any element is convenient and fast.
* As of now, we were using array data structure to organize the group of elements that are to be stored individually in the memory. However, Array has several advantages and disadvantages which must be known to decide the data structure which will be used throughout the program.

**Array contains following limitations:**

* The size of array must be known in advance before using it in the program.
* Increasing size of the array is a time taking process. It is almost impossible to expand the size of the array at run time.
* All the elements in the array need to be contiguously stored in the memory. Inserting any element in the array needs shifting of all its predecessors.

**Linked list is the data structure which can overcome all the limitations of an array. Using linked list is useful because,**

* It allocates the memory dynamically. All the nodes of linked list are non-contiguously stored in the memory and linked together with the help of pointers.
* Sizing is no longer a problem since we do not need to define its size at the time of declaration. List grows as per the program's demand and limited to the available memory space.

**Types of linked lists**

**Single Linked List:**

* It is the most common. Each node has data and a pointer to the next node.

**Diagram

Description automatically generated**

**Double linked List**

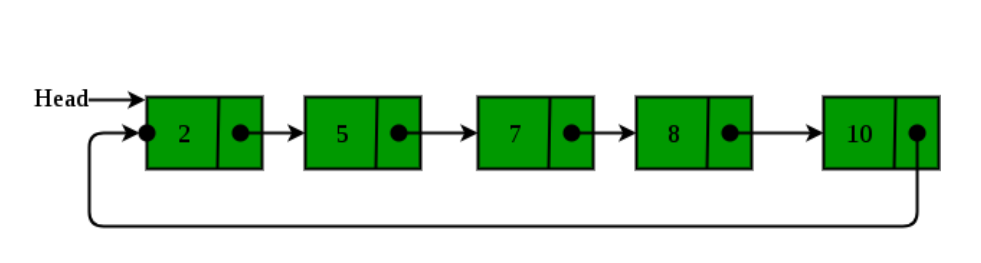
* It is the one in which all nodes are linked together by multiple links which help in accessing both the successor node (next node) and predecessor node (previous node) within the list. This helps to traverse the list in the forward and backward directions.
* The first node must be always pointed by head.
* The previous field of the first node must be NULL.
* The next field of the last node must be NULL.

Chart, diagram

Description automatically generated

**Circular linked List**

* It is a sequence of elements in which every element has link to its next element in the sequence and the last element has a link to the first element in the sequence.
* All nodes are connected to form a circle.
* There is no NULL at the end.



**Circular Double linked list:**

* It has properties of both doubly linked list and circular linked list.
* It doesn’t contain NULL values in any of the nodes. The heads’ previous pointer points to last node and last nodes’ next pointer points to head.

Diagram

Description automatically generated

**Single linked List**

**Single Linked List Definition**

Struct node {

int data;

node\* next;}

A node can be added in three ways

* Insertion at beginning of list
* Insertion at specified position of list
* Insertion at end of list

**Insertion at beginning of list**

**Algorithm:**

**Step 1:** IF PTR = NULL

Write Overflow

Go to step 6

[End of If]

**Step 2:** Else Allocate memory for new node

**Step 3:** SET PTR → DATA = DATA

**Step 4:** SET PTR → NEXT = HEAD

**Step 5:** SET HEAD = PTR

**Step 6:** EXIT

Chart, diagram, waterfall chart

Description automatically generated

**Insertion At Specified Position of List:**

Algorithm:

**Step 1:** IF PTR = NULL

Write Overflow

Go to step 11

[End of If]

**Step 2:** Else Allocate memory for new node

**Step 3:** SET PTR → DATA = DATA

**STEP 4:** SET TEMP = HEAD

**STEP 5:** SET I = 0

**STEP 6:** REPEAT STEPS 7 AND 8 UNTIL I<POS-1

**STEP 7:** SET TEMP = TEMP → NEXT

**STEP 8:** SET I=I+1

**STEP 9:** SET PTR->NEXT=TEMP->NEXT

**STEP 10:** SET TEMP->NEXT=PTR

**STEP 11:** EXIT

Chart, box and whisker chart

Description automatically generated

**Insertion At End of List:**

**Algorithm:**

**Step 1:** IF PTR = NULL

Write Overflow

Go to step 10

[End of If]

**Step 2:** Else Allocate memory for new node

**Step 3:** SET PTR → DATA = DATA

**Step 4:** SET PTR->NEXT=NULL

**Step 5:** IF HEAD==NULL then SET HEAD=PTR

**Step-6**: Else SET TEMP=HEAD

**Step 7:** Repeat Step 8 while TEMP - > NEXT! = NULL

**Step 8:** SET TEMP=TEMP->NEXT  
 [END OF LOOP]

**Step 9:** SET TEMP - > NEXT = PTR

**Step 10:** EXIT

Chart, waterfall chart

Description automatically generated

**Program:**

#include <iostream>

#include <stdlib.h>

using namespace std;

struct node {

int data;

struct node \*next;

}\*head;

void createList(int n);

void insertNodeAtBeginning(int data);

void insertNodeAtEnd(int data);

void insertNodeAtSpecificLocation(int data);

void displayList();

int main()

{

int n, data,choice;

cout<<"Enter the total number of nodes: ";

cin>>n;

createList(n);

cout<<"\n Data in the list:"<<" ";

displayList();

while(1){

cout<<"\n\nSelect an Option";

cout<<"\n1.Insertion At Beginning\n"<< "2.Insertion At End\n"<<"3.Insertion At Specific Location\n"<<"4.Exit\n";

cout<<"\nEnter Your choice:";

cin>>choice;

switch(choice){

case 1:{

cout<<"\nEnter data to insert at beginning of the list: ";

cin>>data;

insertNodeAtBeginning(data);

cout<<"\n Data in the list: "<<" ";

displayList();

break;

}

case 2:{

cout<<"\nEnter data to insert at End of the list: ";

cin>>data;

insertNodeAtEnd(data);

cout<<"\nData in the list: "<<" ";

displayList();

break;

}

case 3:{

cout<<"\nEnter data to insert at specific location of the list: ";

cin>>data;

insertNodeAtSpecificLocation( data);

cout<<"\nData in the list: "<<" ";

displayList();

break;

}

case 4:{

cout<<”exit”;

exit(1);

}

default:{

cout<<"\n Incorrect choice.";

break;

}} }

return 0;

}

void createList(int n)

{

struct node \*ptr, \*temp;

int data, i;

head = (struct node \*)malloc(sizeof(struct node));

if(head == NULL)

{

cout<<"Overflow\n";

}

else

{

cout<<"Enter the data of node 1: ";

cin>>data;

head->data = data;

head->next = NULL;

temp = head;

for(i=1; i<n; i++)

{

ptr = (struct node \*)malloc(sizeof(struct node));

if(ptr == NULL)

{

cout<<"Unable to allocate memory.";

break;

}

else

{

cout<<"Enter the data of node: "<<i+1<<":";

cin>>data;

ptr->data = data;

ptr->next = NULL;

temp->next = ptr;

temp = temp->next;

}

}

cout<<"SINGLY LINKED LIST CREATED SUCCESSFULLY\n";

}

}

void insertNodeAtBeginning(int data)

{

struct node \*ptr;

ptr = (struct node\*)malloc(sizeof(struct node));

if(ptr == NULL)

{

cout<<"Overflow";

}

else

{

ptr->data = data;

ptr->next = head;

head = ptr;

cout<<"DATA INSERTED SUCCESSFULLY\n";

}

}

void insertNodeAtEnd(int data)

{

struct node \* ptr ,\*temp;

ptr = (struct node\*)malloc(sizeof(struct node));

if(ptr == NULL)

{

cout<<"Overflow";

}

else

{

ptr ->data = data;

ptr ->next = NULL;

if(head==NULL){

head= ptr;

}

else{

temp=head;

}

while(temp->next!=NULL){

temp=temp->next;

}

temp->next= ptr;

cout<<"DATA INSERTED SUCCESSFULLY\n";

}

}

void insertNodeAtSpecificLocation(int data)

{

struct node \* ptr ,\*temp;

int i,pos;

if(ptr == NULL) {

cout<<"Overflow";

} else{

ptr = (struct node\*)malloc(sizeof(struct node));

ptr ->data = data;

if(head==NULL){

head= ptr;

ptr ->next=NULL;

}

else{

temp=head;

cout<<"\n Enter the location of the element to be inserted:";

cin>>pos;

for(i=0;i<pos-1;i++){

temp=temp->next;

ptr ->next=temp->next;

}}

temp->next= ptr;

cout<<"DATA INSERTED SUCCESSFULLY\n";

}}

void displayList()

{

struct node \*temp;

temp = head;

while(temp != NULL)

{

cout<<temp->data<<" ";

temp = temp->next;

} }

**OUTPUT:**

Enter the total number of nodes: 3

Enter the data of node 1: 1

Enter the data of node: 2:2

Enter the data of node: 3:3

SINGLY LINKED LIST CREATED SUCCESSFULLY

Data in the list: 1 2 3

Select an Option

1.Insertion At Beginning

2.Insertion At End

3.Insertion At Specific Location

4.Exit

Enter Your choice:1

Enter data to insert at beginning of the list: 23

DATA INSERTED SUCCESSFULLY

Data in the list: 23 1 2 3

Select an Option

1.Insertion At Beginning

2.Insertion At End

3.Insertion At Specific Location

4.Exit

Enter Your choice:2

Enter data to insert at End of the list: 44

DATA INSERTED SUCCESSFULLY

Data in the list: 23 1 2 3 44

Select an Option

1.Insertion At Beginning

2.Insertion At End

3.Insertion At Specific Location

4.Exit

Enter Your choice:3

Enter data to insert at specific location of the list: 55

Enter the location of the element to be inserted:2

DATA INSERTED SUCCESSFULLY

Data in the list: 23 1 55 2 3 44

Select an Option

1.Insertion At Beginning

2.Insertion At End

3.Insertion At Specific Location

4.Exit

Enter Your choice:4

exit

**Deletion of singly linked List**

A node can be deleted in three ways

* Deletion at beginning of list
* Deletion at specified position of list
* Deletion at end of list

**Deletion at beginning of List:**

**Algorithm:**

**Step 1:** IF HEAD = NULL

Write Underflow

Go to step 5

[End of If]

**Step 2:** SET PTR = HEAD

**Step 3:** SET HEAD = HEAD -> NEXT

**Step 4:** FREE PTR

**Step 5:** EXIT

![Diagram

Description automatically generated](data:image/jpeg;base64,/9j/4AAQSkZJRgABAQEASABIAAD/4R0CRXhpZgAATU0AKgAAAAgABgALAAIAAAAmAAAIYgESAAMAAAABAAEAAAExAAIAAAAmAAAIiAEyAAIAAAAUAAAIrodpAAQAAAABAAAIwuocAAcAAAgMAAAAVgAAEUYc6gAAAAgAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAFdpbmRvd3MgUGhvdG8gRWRpdG9yIDEwLjAuMTAwMTEuMTYzODQAV2luZG93cyBQaG90byBFZGl0b3IgMTAuMC4xMDAxMS4xNjM4NAAyMDIzOjAxOjEyIDExOjA1OjQ0AAAGkAMAAgAAABQAABEckAQAAgAAABQAABEwkpEAAgAAAAMxMgAAkpIAAgAAAAMxMgAAoAEAAwAAAAEAAQAA6hwABwAACAwAAAkQAAAAABzqAAAACAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAMjAyMzowMToxMiAxMTowNDozOAAyMDIzOjAxOjEyIDExOjA0OjM4AAAAAAYBAwADAAAAAQAGAAABGgAFAAAAAQAAEZQBGwAFAAAAAQAAEZwBKAADAAAAAQACAAACAQAEAAAAAQAAEaQCAgAEAAAAAQAAC1UAAAAAAAAAYAAAAAEAAABgAAAAAf/Y/9sAQwAIBgYHBgUIBwcHCQkICgwUDQwLCwwZEhMPFB0aHx4dGhwcICQuJyAiLCMcHCg3KSwwMTQ0NB8nOT04MjwuMzQy/9sAQwEJCQkMCwwYDQ0YMiEcITIyMjIyMjIyMjIyMjIyMjIyMjIyMjIyMjIyMjIyMjIyMjIyMjIyMjIyMjIyMjIyMjIy/8AAEQgAVQEAAwEhAAIRAQMRAf/EAB8AAAEFAQEBAQEBAAAAAAAAAAABAgMEBQYHCAkKC//EALUQAAIBAwMCBAMFBQQEAAABfQECAwAEEQUSITFBBhNRYQcicRQygZGhCCNCscEVUtHwJDNicoIJChYXGBkaJSYnKCkqNDU2Nzg5OkNERUZHSElKU1RVVldYWVpjZGVmZ2hpanN0dXZ3eHl6g4SFhoeIiYqSk5SVlpeYmZqio6Slpqeoqaqys7S1tre4ubrCw8TFxsfIycrS09TV1tfY2drh4uPk5ebn6Onq8fLz9PX29/j5+v/EAB8BAAMBAQEBAQEBAQEAAAAAAAABAgMEBQYHCAkKC//EALURAAIBAgQEAwQHBQQEAAECdwABAgMRBAUhMQYSQVEHYXETIjKBCBRCkaGxwQkjM1LwFWJy0QoWJDThJfEXGBkaJicoKSo1Njc4OTpDREVGR0hJSlNUVVZXWFlaY2RlZmdoaWpzdHV2d3h5eoKDhIWGh4iJipKTlJWWl5iZmqKjpKWmp6ipqrKztLW2t7i5usLDxMXGx8jJytLT1NXW19jZ2uLj5OXm5+jp6vLz9PX29/j5+v/aAAwDAQACEQMRAD8A9/ooAKKACigAooAKKACigAooAKKACigAooAKKACigAooAKKACigAooAoahqJspIY1hMjSBm+9jGMf41V/tyT/n0/8iVz1MTGnLlZvChKcboP7ck/59P/ACJR/bkn/Pp/5ErP65DsX9Vl3L+n30eo2UdzGCFcdD2PcVarsWxysKKACigAooAKKACigAooAKKACigAooAKKACigAooAKKAMTW/+P20/wCucn80rkdbOuR6jBJpYDQrERJGwyGbcuPpxmvMxFvbano0L+y0MddQ8bBId9jEWKnzCE6Ht37c/Wuk0H7f/ZMZ1Ld9qJYtuABxnj9KxkoJe6aRc2/eOi8Kf8gCH6t/M1t17C2R5b3CimIKKACigAooAKKACigAooAKKACigAooAKKACigAooAy9Y02e/EL21wIZY9wyV3Ag4z/ACFYg0XWDcNF/aEXCg/6kd8/4VhPDwm7s2hXlBWRL/YOsf8AQQi/79Cq9rpOs3AlzqEQ2Ssn+qHY4qfqlMr61M6LRbB9M0yO1d97JnLYxmtCuk5wooAKzNW8QabojQrqE5i84nadhI/HHSgDKT4h+GJFLJqSsASDhD2z7f7J/T1qS88baTYam9lctIhQD5wpO7Pl4wBz1lUUWAsaL4s0rX7u4trGUu8G0njqrKrA/ju/Q1uUAFFABRQAUUAFFABRQAUUAFFABRQAUUAFVl/5CEn+4v8AWgCzWUjz6fNOn2OadHkaRXiZO5zg7mFAEv8Aacv/AEC7z84v/i6P7Tl/6Bd5+cX/AMXQAf2nL/0C7z84v/i6P7Tl/wCgXefnF/8AF0AH9py/9Au8/OL/AOLqhqMVtqoT7bodzLsBA3GLj/x+gDM/4RvRNoX/AIRqfAzj/VcZIP8Af9QKs3GlabdXIuZtAuWmGPnJizwVI/j/ANhfyFO7Adp2m2GlXP2iy0G5hkK7SVMQyMAY+/8A7I/Ktf8AtOX/AKBd5+cX/wAXQAf2nL/0C7z84v8A4uj+05f+gXefnF/8XSAP7Tk76ZeAeuYz/J6t288dzCssRJVumRigCWigAooAKKACigAooAKKACigAqsv/IQk/wBxf60AWaKACoLy4FpaSTld2wZwO9A0rsz/AO2Jv+fUf990f2xN/wA+o/77qOdHR9WkIdZlUEm1GB/t1H/wkUZH+ob8xXnY7M44RxTW5cMHKQv/AAkUf/PBvzFJ/wAJFH/zwb8xXB/rDT7F/UJdzUsruO+tUnjztYdD2qxX0MXzRTOFqzsFU9Qv1sI0ZkLbjjissTXVCk6j6FU4c8lEzm8QoykeSwyMZBHFZegX9xpVi0F5cPeOZGYSEBcAnIHHpXh/6w0+x2/UJdzXTxCjyKggbLHA59s1P/a3/TL/AMer28vxCxlL2qOHERdGfKw/tb/pl/49TX1pIl3yR4XIBIOepxXd7JmHtDUB3AEd6WsTUKo3GoiGdovLztxzmqjHmdiZSsiL+1v+mQ/76o/tb/pkP++q09iyPaET64Ek2eTk4z96qy+K7Z72SzVV+0RqHaPdzg9DWbjZ2OiEOaNwfxZax3sVmwUXEqlkj38kDqauWutx3F4ts0ZVnBKnOc4pWG6dlc1arTQzeeJoWXOMMrDg0jMTdff3Yf1pN19/dg/WgBd17n7sOPxqlqpvP7Ln3LDjbzjPqKHsOO6KVFc56wyX/VP9KwRNwK+X4i+Kn8/0OrDK9w86jzuK+bT1OrkOo8OG5/sWHaI8c9c+prVzd+kX61+o0v4cfRHy0/iYZus9IsfjXKay2spezm+NubAuv2YIDvB2fNn2znFcObf7nP0NsJrWiZvnUedX55c+j5AFxsljbPRv6Gp/t/8AtV+i8LRvgfmz5jN3y4i3kH2//aqG6vGkg2IRuLLjPT7wr6OUNGebGWp2cban5S4S16f7VO3ap/ctf/Hq8s7hsjat5T7Ftd207fvda5M3WqQtt1hoTeYG8wghT6Vvh1eZlWdoifb/APao+3/7Vd3szk5itPffvwc/w/1rmPFN9GTb/ZC39tA5tWj6r67v9j1FcNZWmz3cPC9BMZ4YuFW4uW1Ak60Tmcv3HYp/s12mh3Hna/b85wrVmi6lO1Js72kbO07cbscZqDzzze4tfHz3DOJ1+RmC7XAyD6cY+makutN8Yf2dprWs5F1DalJd0mcyYbB9+StPQRa0u38YQ+JYp76cS6ft8txkZI3SFTgDHAZR+XpXaXlv9rspYN5TeuNw7UMadjnzoWsZ41OP/vyKT+wtZ/6CUf8A35FTyRN/rExDoOsMpB1KPB/6Yiq3/CIXmP8Aj8T/AL4/+vXn43K6WLac21Y2pY6pTvZC/wDCI3n/AD+J/wB8f/Xo/wCEQvP+fxP++P8A69cH+ruG/mZr/adXsdJpNk2n6fHbM24p3A61dr34rlSS6HnN3dwrM1rS21S1WNJfLdG3A4yKxxFCNek6UtmVSqOnNTXQwf8AhEbz/n8T/vj/AOvR/wAIjef8/if98f8A168b/VzDfzM7/wC06vYa/g68YD/TUGDn7n/16b/whl5/z/L/AN8f/Xr3cupxwFH2VPVXuedim8TPnkH/AAhl5/z/AC/98f8A16cng26EiM16pCsCRs64Nd7xbatY5lQSdzsUG1FHoKdXGdAVzut+HJdSvftENz5ZKgMpXPStKVR05cyInDnVjM/4Qy8/5/l/74/+vR/whl5/z/L/AN8f/Xro+tvsZfV0MfwPdu2Tfr0x9yq4+Hbi6a5F2nnMoUv5fOBWE6nM7s76WIlTgoJbCv8ADyR7iOdrtDLGCFfy+QD1FamjeFJ9M1JLqS6EgUEbduKm6KnipTi4tbnV0VBzBRQAUUAFFAHI+MLbxBdahp8Wi3EkEZWTznWQqoOVxnAOeN3HFTeENL1rTIpU1e8e46CMGZpAAPcjqafQDqKKQBRQAUUAFFABRQAUUAFFABRQAUUAFFABRQAUUAFFABRQAUUAFFABRQAUUAFFABRQAUUAFFABRQAUUAFFABRQAUUAFFABRQAUUAFFABRQAUUAFFABRQAUUAFFABRQAUUAFFABRQAUUAFFAH//2QD/4THoaHR0cDovL25zLmFkb2JlLmNvbS94YXAvMS4wLwA8P3hwYWNrZXQgYmVnaW49J++7vycgaWQ9J1c1TTBNcENlaGlIenJlU3pOVGN6a2M5ZCc/Pg0KPHg6eG1wbWV0YSB4bWxuczp4PSJhZG9iZTpuczptZXRhLyI+PHJkZjpSREYgeG1sbnM6cmRmPSJodHRwOi8vd3d3LnczLm9yZy8xOTk5LzAyLzIyLXJkZi1zeW50YXgtbnMjIj48cmRmOkRlc2NyaXB0aW9uIHJkZjphYm91dD0idXVpZDpmYWY1YmRkNS1iYTNkLTExZGEtYWQzMS1kMzNkNzUxODJmMWIiIHhtbG5zOnhtcD0iaHR0cDovL25zLmFkb2JlLmNvbS94YXAvMS4wLyI+PHhtcDpDcmVhdGVEYXRlPjIwMjMtMDEtMTJUMTE6MDQ6MzguMTE5PC94bXA6Q3JlYXRlRGF0ZT48eG1wOkNyZWF0b3JUb29sPldpbmRvd3MgUGhvdG8gRWRpdG9yIDEwLjAuMTAwMTEuMTYzODQ8L3htcDpDcmVhdG9yVG9vbD48L3JkZjpEZXNjcmlwdGlvbj48L3JkZjpSREY+PC94OnhtcG1ldGE+DQogICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgCiAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAKICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgIAogICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgCiAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAKICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgIAogICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgCiAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAKICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgIAogICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgCiAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAKICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgIAogICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgCiAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAKICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgIAogICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgCiAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAKICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgIAogICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgCiAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAKICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgIAogICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgCiAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAKICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgIAogICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgCiAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAKICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgIAogICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgCiAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAKICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgIAogICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgCiAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAKICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgIAogICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgCiAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAKICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgIAogICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgCiAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAKICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgIAogICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgCiAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAKICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgIAogICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgCiAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAKICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgIAogICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgCiAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAKICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgIAogICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgCiAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAKICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgIAogICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgCiAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAKICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgIAogICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgCiAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAKICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgIAogICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgCiAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAKICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgIAogICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgCiAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAKICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgIAogICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgCiAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAKICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgIAogICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgCiAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAKICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgIAogICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgCiAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAKICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgIAogICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgCiAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAKICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgIAogICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgCiAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAKICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgIAogICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgCiAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAKICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgIAogICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgCiAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAKICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgIAogICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgCiAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAKICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgIAogICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgCiAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAKICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgIAogICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgCiAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAKICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgIAogICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgCiAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAKICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgIAogICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgCiAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAKICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgIAogICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgCiAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAKICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgIAogICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgCiAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAKICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgIAogICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgCiAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAKICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgIAogICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgCiAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAKICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgIAogICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgCiAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAKICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgIAogICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgCiAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAKICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgIAogICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgCiAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAKICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgIAogICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgCiAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICA8P3hwYWNrZXQgZW5kPSd3Jz8+/9sAQwADAgIDAgIDAwMDBAMDBAUIBQUEBAUKBwcGCAwKDAwLCgsLDQ4SEA0OEQ4LCxAWEBETFBUVFQwPFxgWFBgSFBUU/9sAQwEDBAQFBAUJBQUJFA0LDRQUFBQUFBQUFBQUFBQUFBQUFBQUFBQUFBQUFBQUFBQUFBQUFBQUFBQUFBQUFBQUFBQU/8AAEQgBXAQjAwEiAAIRAQMRAf/EAB8AAAEFAQEBAQEBAAAAAAAAAAABAgMEBQYHCAkKC//EALUQAAIBAwMCBAMFBQQEAAABfQECAwAEEQUSITFBBhNRYQcicRQygZGhCCNCscEVUtHwJDNicoIJChYXGBkaJSYnKCkqNDU2Nzg5OkNERUZHSElKU1RVVldYWVpjZGVmZ2hpanN0dXZ3eHl6g4SFhoeIiYqSk5SVlpeYmZqio6Slpqeoqaqys7S1tre4ubrCw8TFxsfIycrS09TV1tfY2drh4uPk5ebn6Onq8fLz9PX29/j5+v/EAB8BAAMBAQEBAQEBAQEAAAAAAAABAgMEBQYHCAkKC//EALURAAIBAgQEAwQHBQQEAAECdwABAgMRBAUhMQYSQVEHYXETIjKBCBRCkaGxwQkjM1LwFWJy0QoWJDThJfEXGBkaJicoKSo1Njc4OTpDREVGR0hJSlNUVVZXWFlaY2RlZmdoaWpzdHV2d3h5eoKDhIWGh4iJipKTlJWWl5iZmqKjpKWmp6ipqrKztLW2t7i5usLDxMXGx8jJytLT1NXW19jZ2uLj5OXm5+jp6vLz9PX29/j5+v/aAAwDAQACEQMRAD8A/VOiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACim7qdQAUUUUAFFFFABRRRQAUUUUAFFFFABRRRQAUU3dTqACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigApKWkoA+RdD8RfFPxzrXjO5t/ipfaJZWHibVNLtbG30XT5VihgunjjG94SxwoAySScc1s/2Z8V/wDotWqf+E/pf/xiqHwh/wBb8Qv+x313/wBLZK9Br4LF5hiqdecYzskz7TC4HD1KMZSjdtHG/wBmfFf/AKLVqn/hP6X/APGKP7M+K/8A0WrVP/Cf0v8A+MV2VFcn9p4v+c6v7Pw38hxv9mfFf/otWqf+E/pf/wAYo/sz4r/9Fq1T/wAJ/S//AIxXZUUf2ni/5w/s/DfyHG/2Z8V/+i1ap/4T+l//ABij+zPiv/0WrVP/AAn9L/8AjFdlRR/aeL/nD+z8N/Icb/ZnxX/6LVqn/hP6X/8AGKP7M+K//RatU/8ACf0v/wCMV2VFH9p4v+cP7Pw38hxv9mfFf/otWqf+E/pf/wAYo/sz4r/9Fq1T/wAJ/S//AIxXZUUf2ni/5w/s/DfyHG/2Z8V/+i1ap/4T+l//ABij+zfiv/0WrVP/AAn9L/8AjFdlRR/aeL/nF/Z+G/kPKfiHr/xX+H3hG88Qf8Le1DUBZPCz202haaqyK0yIwJWDI4Y9K+ovDvjKy8SW4msrhbiLJG5a+Zf2lv8AkiPib/cg/wDSiOuy/Z3vG/sVVJOPObj8q+xyerUxVCU6ju0z5jNKNOhVjGmraH0UrbhmnVFbtujBqWvX6nkhRRRQAUUUUAFFFFABRRRQAUUUUAFFFFABRRRQAUUUUAFFFFABRRRQAUUUUAFFFFABRRRQAUUUUAFFFFABRRRQAUUUUAFFFFABRRRQAUUUUAFFFFABRRRQAUUUUAFFFFABRRRQAUUUUAFFFFABRRRQAUUUUAFFFFABRRRQAUUUUAFFFFABRRRQAUUUUAFFFFABRRRQAUUUUAFFFFABRRRQAUUUUAFFFFABRRRQAUUUUAFFFFABRRRQAUUUUAFFFFABRRRQAUUUUAFFFFABRRRQAUUUUAFFFFABRRRQAUUUUAFFFFABSdxS0npSA+UPhD/rfiF/2O+u/wDpbJXoFef/AAh/1vxC/wCx313/ANLZK9Ar8zx3+81PU/Q8H/u8PQKjmmS3heWRgkaKWZj0AAyakrO8Rf8AIvap/wBesv8A6Aa4ktUjrlsYXgX4t+DviZJex+FvENnrb2RAuVtXJMROevHtXXV+Uv7IvxL8ceEL3x9p3w68Np4i8RXMwkYXDbYIEVmOWJIySM4APavrP9kn9r3UPjRN4j0TxfpkGk+JdDDvcLbgiMohwxAJPQkDrXqYjL50+Zw1St6nmUMdGpZS0b+4+p6K+Fbj9tL4r/FLxNrw+D3gyx1bw7o5bzrnUPvSKM5KfMvp05Nez/BH9q62+KXwj1/xPcaPcRa14dLQ6jpMCl5fNBIwigEnO08cmsKmCq0480v+Gv3NoYulN2R9B0lfA2tftmfHfStAu/G8vw6tLDwLbXIik+2RtHdhSQB8pcHnPXbivcfGH7YGj+Hf2abX4qQWjTC82wQWm7n7QQflJ9AVIzTnga0bK179hRxlGV9bW1Pc/FHirSfBWh3Os67fw6ZplsAZrqc4RMnHNReD/Guh/EDQodZ8O6nBrGlzEiO6tmyjEdQK/O74tftIfGDxv+z3rF94z8E2tn4R1uJRZ6hZcPHhwQZFLk4JGOg619Nf8E9/+TYfD3/Xef8A9CrStgvYUeeT1vYzpYv21b2cVpa59I0tJ6UteYeieY/tLf8AJEfE3/XOD/0ojrqf2eP+QQP+urVy37S3/JEfE3/XOD/0ojrqf2d/+QQP+urV9/w//us/U+Ozr+PH0Po60/1IqeoLT/Uip691nhhRRRSAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigApPSlpPSkB8ofCH/W/EL/sd9d/9LZK9Arz/wCEP+t+IX/Y767/AOlslegV+Z47/eanqfoeC/3eHoFZ3iL/AJF7VP8Ar1l/9ANaNRzQpcQyQyqHjkUo6nuD1FccXytM65aqx8Ef8Ew4U/tz4lyBF3meMFscn534rnP2ctLk1T9qv41WFr+7km0u5VNvHzGRPSvuzwD8H/Bvwtmv5PCvh+10V74hrlrct+9Iz1yT6mm+Gfg34K8HeKtR8TaN4etdP13UFKXV7Hu3yqSCQcnHUDtXryxseepJL4kvwPJjg5KME38LZ+S/wN8HaJYzeJtK8XfGHWvhNqdhMwksLcsizkdv9auW46Y9K94+FFzp/wAE/wBnv4oeNvhf4l1nxNO0mJL7WNPSALJyDKuJH39Dycda+2PGf7Nfwx+ImrHVPEXg3TtUv85M8qsGJ9flYV1tj4D8PaZ4ZPh610e1h0Rk8s2Sp+7K+hFdFXMoVEtHra60sY0svnTe/e3zPyG8QTr8QPgzPr+pfGfWdY8WXY8x/BsErSpgEMxdfM4AGT93tXY69cQ+MP8Agnjo0GkXcd7c6DrXmalaRtl7dWMpVnHYECv0e8Lfs3/DPwVcXlxong3TdPmu0aOd41Yl1YEMOSeoJ6etXfDfwL8A+D9H1XStG8L2Nhp2qHN7bxhik556gk+p6etXLNKenLF6O6Ijl9Tq1qrHxZ8WP2o/h94o/Yvh8M6XqMN74iuLOKE6THgyQlJASWGeMAE179/wT3/5Nh8Pf9d5v/QhXc6b+yp8JdHt7+Gz8C6ZBFfLsuFUP+8Gc4+9x+Fdz4J8C6B8OfD8Oh+GtMh0jSYWLR2sGSqknJPJJ5rgxGKoypOnTT1d9Tto4erGqqlRraxvUtFFeSekeY/tLf8AJEfE3/XOD/0ojrqf2d/+QQP+urVy37S3/JEfE3/XOD/0ojrqf2d/+QQP+urV+gcP/wC6z9f8j43Ov48fT9T6OtP9SKnqC0/1IqevdZ4YUUUUgCiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKSlpPWgD5P8AhD/rPiF/2O+u/wDpbJXoNeITeKvFPwX1zxna6l8O9avrXUPFGq6laX1tPbiOWGa6d0YBpA33SDyB1qr/AMNSXIOD8OfEH/f+2/8AjlfCYrLcXVrznCDabPs8LjsPToRjKaTSPeaK8F/4amuf+ic6/wD9/wC2/wDjlVNV/a6XRNPlvb74f67b20WN8jT2+FycD+P1Irl/snG/8+2dX9o4X+dH0LRXgv8Aw1Nc/wDROdf/AO/9t/8AHKP+Gprn/onOv/8Af+2/+OUf2Tjf+fbD+0cL/Oj3qivBf+Gprn/onOv/APf+2/8AjlH/AA1Nc/8AROdf/wC/9t/8cp/2Tjf+fbD+0cL/ADo96orwX/hqa5/6Jzr/AP3/ALb/AOOUf8NTXP8A0TnX/wDv/bf/AByj+ycb/wA+2H9o4X+dHvVFeC/8NTXP/ROdf/7/ANt/8crI8N/tmR+KrO5ubH4eeIHjt7qW0fM1sP3kbbW/5adjS/srG/8APth/aOF/nR9I0V4L/wANTXP/AETnX/8Av/bf/HKP+Gprn/onOv8A/f8Atv8A45S/snG/8+2L+0cL/OjrP2lv+SI+Jv8ArnB/6UR11P7O/wDyCB/11avn/wCKXxx1D4ieA9U8PW3gPWrKa+ESLcTzW5RMSq5JAfPRT0r6B/Z5UjSFyP8Alq3avs8mw9XDYacasbNs+XzSvTxFaMqbvofRtp/qRU9QWn+pFT16fU8sKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKTFLRQBxnjv4f2HjNY/tqu3lbtuxtvXrXnE/7POi72Ijm/77/wDrV7yyhhg1GbdG7VpGbjohHgv/AAz3o3/POb/vv/61cZ8Yv2X9B8SfDfWdOn1P+w4ZhFuv7hg0cOJUPIJHXG3r1NfVn2WP0rzn9ob4e3PxL+DviHw1p97b6bd3ywCO6u3Kxx7J45DkjkcKR+NV7RhY4CP4A6R523y5f++//rVcH7Pej4/1c3/ff/1q9ljVfPxjmtNbdGGcU/atBY8H/wCGe9G/55zf99//AFqP+Ge9G/55zf8Aff8A9avefssfpR9lj9KPbMLHg3/DPejf885v++//AK1H/DPejf8APOb/AL7/APrV7z9lj9KPssfpR7ZhY8G/4Z70b/nnN/33/wDWrwj9l74O6Xq3gvxPJLHKWj8XaxAMPjhbpgK+7zbIB0r5m/ZDjX/hA/GBx/zPGvf+lr0e0bYmjdh/Z90d4wfLm/77/wDrVJ/wz3o3/POb/vv/AOtXuFlCrRAYqz9lj9KPasLI8G/4Z60bP+rm/wC/ldz4J8A2/hO3FvbBhGCW+Y5PNegfZY/SlFui9BUuo3uOyC3XbHipaQDbS1kMKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKTdS1+dH7Xn7VnxV+FX7Ymg+DPB1+13peoWsCw6I0UflyzvjBZ8bgCeuDxQB+i2aK/LfUf2pf2gf2cf2gPCumfE/W7HX/DXiKQEWtpEFjtgxICqdisWBHfIr0r9r79srxxp/xj0f4P8AwtubTSteu4457rWrwrst1fgfeyMDnPH0p2Yj7+zRmvzf+Ev7VnxY+GHxutvhZ8VfEWneLJ9ftpDpWuaWUKxy7SRnaF4GD1Gc1r/s1/taeNV8O/HHS/HviFtX17wPHPLDcSxRxtwr7FwqgHlPTvT5WM/QnNJur81v2Ff2xPiX448f+JtI+JWsS3cE2mzarpYnhjiCRhlAUbVGfvd69T/YL+PHjr43ah4+8TeJddfUfDH29rTSLNoURbfYctgqoJ4YdSaOViPpX44fHXwt+z14KPirxhNdQaQJlty9pAZn3MCR8o7cGuP8e/tlfDj4b/DXwv461m51BPD/AIkYLYSQ2bPIc4+8mfl614r/AMFYLgXH7KrsP+grD/6BJXgnxy+LXiT4a/se/BOTw/dW9tNKUhlNxZw3AK5HQSKwB56jmhRuFz9RPD/iK08SaLp+q2bMbO+t47mEuMMUdQy5HY4PStLdmvzP/a//AGrfiD8GPDvwdvvDOtSWkepadHLqFvDBEftJ2x8AFTt6nhcVznxi/aU/ab+CF34W8feIdX0//hEtWuEWTQLaIHyEIBIkYrnJBzwx5p8rC598fHb9qTwL+znNokXjK4vYH1h2jtfsdqZskEA5weOor1LT9Th1Kzt7qE5injWRM9cMAR/Ovy1/4KaePLbx5oPwT8QIRBDfL9swT9wP5Tf1rrtH/ac+IP7RHxh0TwR8JNdbQPC2gQwtrWupFHJ5wCDKoHDfLkgdAc5oUQufpNnvS1iaLfM1pFE87XUiKFaVhhnPqcVtKdwzUvQYtFFFIAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACvMf2kfBuufED4K+JNA8NhTrV4sAt90ojHyzxu3zHp8qtXp1eW/tM2HiPVPgj4ltvCUVzP4hkWD7Klo22UkXEZbByP4d3fpmgDvI2XzsY5rUj+6Ky49vnY71qR/dFMB1FFFIAooooARvumvmL9kL/kQvGH/Y8a9/6WPX0633TXzL+yfH9h8KeNLGTi4TxlrU7J3CyXblT+Iqo7gfSFj/AKv8Kt1Usf8AV/hVuk9wCiiikAUUUUAFFFFABRRRQAUUUUAFFFFABRRRQAUUUUAFFFFABRRRQAUUUUAFfl1+1Yob/gpp4ByM4htGH1+Wv1FrlPEfgjRtY1BdSn0awuNUjAEd7Lao0yY6YcjcPwNNbiZ+Y3/BSSM/8LM+EpY5Jv4x/wCRGrz79ur4T2OlftOWfi3xppmqXnw+v7WOK4utLJEkTrncSQrYxnoetfrBdfDjSNY8uTU9IsdQlhOYZLu2SVoz6qWBx+FN1D4eWGuQyQalYW9/A3/LK6hWVPyYEVs5LqI/KL9nHwH8FfE3xv0Ob4b+GfGOpRWMn2g63eXvl28DYIwyGH5uv94ViftZf2t8Mf2j/GXh7Soni/4WFHaBpVB2s0kkgYe5+YV+sWlfCrTvDcrrpOlWemRsfmWzt0iB/BQK1dS+Fuia15N1f6Lp97fQ4MVzcWqSSIQcgqzAkU7qwj8wv2vLO5+AEnw08VaRAYY00RdFvNgwWdlB/PKV9SfsB+E/+EO/Zz8LQiPyZtTX+0ZkIwd7gDn/AL5r6PufhfpWuQiHV9LstTiRtyR3lukqqfUBgcGr+j+C49LuESCGO3t4+EjjUKqgdAAKd1cLHy7/AMFUIyn7J+Dyf7Ug/wDQJK+W/wBq7/kz34Jnt5sf81r9bda8Nab4h0v7Fq2nWmp2uQ3kXkCzJkdDtYEZrj9S+Fui6pHDa3Wi6fc2Nuf3NtNao0cX+6pGB+FZRauM/Mf9uQeZofwFDcj7FBxnj7sdek/8FIFZv2dNEY9p4cfklfdV58K9H1RoBfaNY3i2wxALi1SQRf7uQdvQdKuT/DzTdWt/s+qaZZ6jbL0hvLdJUH0DAiteZIR+U/7aOmxap8GfgDZz7jDPpsCNtODgxxDirl5oz/sJfHTwr4g0Tzo/APiiGK3vYmOQjsvTP1Ct0r9QdR+Fejat9nju9F0+6htgBbxz2qOsQGMBAR8vQdKTUfhZpOuxpDqmlWeoQxndHHdW6SqhHcBgcUXQzX+H2pW2saXBeWs63EE6LIkiMCGU9wa7hcbeK5Lw7oI0VY4IY1gt4xtSKNQFUDoABwBXWR/dFYy3Gh9FFFQMKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAry39prXPEPhz4IeJtR8Ki6/t+BYPsv2NGeXJuIlbaF5PylunbNepV5l+0l461X4afBXxJ4k0QL/alisBh3IHGXuI0bgg54Y0AdvGq+d15rUj+6Ky44x52c1qR/dFNgOooopAFFFFACHkV8r+MtQn/AGcvjHqGt3FlPN4D8VNH9puYEL/2fdL/ABMqjIR9zEsRgbRk19U1m6zo9prFlLa3ltFd20oxJDNGHRh6FSMGmgMzwn4w0XxNZxz6TqtlqMTjIa1uEkH6Guk3D1r508Sfsf8AgLV7ySdIdZ0gOdxj0PWLmwjz/uwuorF/4Yp8Ar11bx5/4Wepf/HqpoR9Sbh6/rRuHr+tfLf/AAxV8P8A/oLeO/8Aws9S/wDj1H/DFXw//wCgt47/APCz1L/49SsM+pNw9f1o3D1/Wvlv/hir4f8A/QW8d/8AhZ6l/wDHqP8Ahir4f/8AQW8d/wDhZ6l/8epWYH1JuHr+tG4ev618t/8ADFXw/wD+gt47/wDCz1L/AOPUf8MVfD//AKC3jv8A8LPUv/j1FmB9Sbh6/rRuHr+tfLf/AAxV8P8A/oLeO/8Aws9S/wDj1H/DFXw//wCgt47/APCz1L/49RZgfUm4ev60bh6/rXy3/wAMVfD/AP6C3jv/AMLPUv8A49R/wxV8P/8AoLeO/wDws9S/+PUWYH1JuHr+tG4ev618t/8ADFXw/wD+gt47/wDCz1L/AOPUf8MVfD//AKC3jv8A8LPUv/j1FmB9Sbh6/rRuHr+tfLf/AAxV8P8A/oLeO/8Aws9S/wDj1H/DFXw//wCgt47/APCz1L/49RZgfUm4ev60bh6/rXy3/wAMVfD/AP6C3jv/AMLPUv8A49R/wxV8P/8AoLeO/wDws9S/+PUWYH1JuHr+tG4ev618t/8ADFXw/wD+gt47/wDCz1L/AOPUf8MVfD//AKC3jv8A8LPUv/j1FmB9Sbh6/rRuHr+tfLf/AAxV8P8A/oLeO/8Aws9S/wDj1H/DFXw//wCgt47/APCz1L/49RZgfUm4ev60bh6/rXy3/wAMVfD/AP6C3jv/AMLPUv8A49R/wxV8P/8AoLeO/wDws9S/+PUWYH1JuHr+tIdrda+XP+GKvh//ANBbx3/4Wepf/HqP+GKvh/8A9Bbx3/4Wepf/AB6izA+oti+tARR3FfLv/DFfgD/oLePP/Cz1L/49R/wxX4A/6C3jz/ws9S/+PUWYH1D5SE54pdq9Mivl3/hivwB/0FvHn/hZ6l/8eo/4Yr8Af9Bbx5/4Wepf/HqLMD6iEajuKBGoOcivl3/hivwB/wBBbx5/4Wepf/HqP+GK/AH/AEFvHn/hZ6l/8eoswPqTcvtTfLX1FfLv/DFXw/8A+gt47/8ACz1L/wCPUf8ADFfgD/oLePP/AAs9S/8Aj1HKB9ReWvrR5a+tfLv/AAxX4A/6C3jz/wALPUv/AI9R/wAMV+AP+gt48/8ACz1L/wCPU7MD6i8tfWjy19a+Xf8AhivwB/0FvHn/AIWepf8Ax6j/AIYr8Af9Bbx5/wCFnqX/AMeoswPqIRqPSnbh618t/wDDFfgD/oLePP8Aws9S/wDj1H/DFXw//wCgt47/APCz1L/49SswPqTcPX9aNw9f1r5b/wCGKvh//wBBbx3/AOFnqX/x6j/hir4f/wDQW8d/+FnqX/x6izA+pNw9f1o3D1/Wvlv/AIYq+H//AEFvHf8A4Wepf/HqP+GKvh//ANBbx3/4Wepf/HqLMD6k3D1/WjcPX9a+W/8Ahir4f/8AQW8d/wDhZ6l/8eo/4Yq+H/8A0FvHf/hZ6l/8eoswPqTcPX9aNw9f1r5b/wCGKvh//wBBbx3/AOFnqX/x6j/hir4f/wDQW8d/+FnqX/x6izA+pNw9f1o3D1/Wvlv/AIYq+H//AEFvHf8A4Wepf/HqP+GKvh//ANBbx3/4Wepf/HqLMD6k3D1/WjcPX9a+W/8Ahir4f/8AQW8d/wDhZ6l/8eo/4Yq+H/8A0FvHf/hZ6l/8eoswPqTcPX9aNw9f1r5b/wCGKvh//wBBbx3/AOFnqX/x6j/hir4f/wDQW8d/+FnqX/x6izA+pNw9f1o3D1FfLf8AwxV8P/8AoLeO/wDws9S/+PVLF+yF4Z0r95o3iPxnYXS8rLceJL27UH1KSSkGnygfT+70p1fL9n488dfAXxZo+leLtTXxX4L1KUW0euvAsE9nIeVWRVGCmA3zFieB68fStpeJcRqytlWAIPsaXKBbooopAFFFFABRRRQAUUUUAFFFFABRRRQAUUUUAFFFFABRRRQAUUUUAFFFFABRRRQAUUUUAFFFFABRRRQAUUUUAFFFFABRRRQAUUUUAFFFFABRRRQAUUUUAFFFFABRRRQAUUUUAFFFFABRRRQAUUUUAFFFFABRRRQAUUUUAFFFFABRRRQAV5v+0N4+m+F/wf8AEPieC0hvprBYStvcLuRt88cfIz/t5+oFekV5z+0F4u0/wH8IfEGu6rpFvrthZrCZdPulDRy7po0GQQehYHp2oA7COPE27Nakf3RWVGrednPFasf3RTYDqKKKQBRRRQAUnWlooAia3VuoFRmxQ81ZopgVfsKUfYUq1RRdgVfsKUfYUq1RRdgVfsKUfYUq1RRdgVfsKUfYUq1RRdgVfsKUfYUq1RRdgVfsKUfYUq1RRdgVfsKUfYUq1RRdgVfsKUfYUq1RRdgVfsKUfYUq1RRdgVfsKUfYUq1RRdgVfsKUfYUq1RRdgVfsKUfYUq1RRdgVfsKUfYUq1RRdgVfsKUfYUq1RRdgVfsKUfYUq1RRdgVfsKUfYUq1RRdgVfsKUfYUq1RRdgVfsKUfYUq1RRdgVfsKUfYUq1RRdgVfsKUfYUq1RRdgVfsKUfYUq1RRdgVfsKUfYUq1RRdgVfsKUfYUq1RRdgVfsKUfYUq1RRdgVfsKUfYUq1RRdgVfsKUfYUq1RRdgVfsKUfYUq1RRdgeAfttWKQ/sv+PbpF2z2tj50L90cOgBHvg16j4XkZNOsVY5JgjP/AI6K/OP/AIKF/tUfEz4O6x4y+GniSztNd8JeKbEf2PfLELeSFcoX+6Dvw3y8mvUP2H/2oPiH+0p4k1TX9StrTQfANhClpbWMUayvLcAfN+9IBAAKmrWoH3uhLKDT6p2N2LiNSOlXKzAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigArzn9oLVfDWh/CLxBfeL9M/tnw7CsP2qxyR5gM8YXpjo5U/hXo1ec/tBQeFbn4Q+II/GzXC+F2WH7YbVisnE8ZTBH+2EoA6+MN52e1asf3RWXHu872rUj+6KYDqKKKQBRRRQAUUUUAFFFFABRRRQAUUUUAFFFFABTQTmnU08KaAPILz9rP4ZWOoXlk+uXEk9pPJbTCHTbmRVkRirLuWMg4Ixwai/wCGvPhj/wBBi9/8FN3/APGq83+AP/It+JM8/wDFT6v/AOlclenVyyrcrase9RyxVKcZ826If+GvPhj/ANBi9/8ABTd//GqP+GvPhj/0GL3/AMFN3/8AGqmoqfrHkbf2Sv5yH/hrz4Y/9Bi9/wDBTd//ABqj/hrz4Y/9Bi9/8FN3/wDGqmoo+seQf2Sv5yH/AIa8+GP/AEGL3/wU3f8A8ao/4a8+GP8A0GL3/wAFN3/8aqaij6x5B/ZK/nIf+GvPhj/0GL3/AMFN3/8AGqP+GvPhj/0GL3/wU3f/AMaqaij6x5B/ZK/nIf8Ahrz4Y/8AQYvf/BTd/wDxqj/hrz4Y/wDQYvf/AAUXf/xqpqSl9Y8hf2Sv5xkX7XHwzmlSNNYvNzMFG7SroDJ9T5fFdSvxs8IsoI1JiD0xbyc/+O1xus/8gi9/64P/AOgmuJ0WT/iT2f8A1xX+Vfm3FvFmKyGpSjh4J819/Kx6GFyOlVT55M9p/wCF1+Ev+gi//gPJ/wDE0f8AC6/CX/QRf/wHk/8Aia8i836Ueb9K/P8A/iJeZ/8APuP4nd/q9h/5meu/8Lr8Jf8AQRf/AMB5P/iaP+F1+Ev+gi//AIDyf/E15F5v0o836Uf8RMzP/n3H8Q/1ew/8zPXf+F1+Ev8AoIv/AOA8n/xNH/C6/CX/AEEX/wDAeT/4mvIvN+lHm/Sj/iJmZ/8APuP4h/q9h/5meu/8Lr8Jf9BF/wDwHk/+Jo/4XX4S/wCgi/8A4Dyf/E15F5v0o836Uf8AETMz/wCfcfxD/V7D/wAzPXf+F1+Ev+gi/wD4Dyf/ABNH/C6/CX/QRf8A8B5P/ia8i836Ueb9KP8AiJmZ/wDPuP4h/q9h/wCZnrv/AAuvwl/0EX/8B5P/AImlj+NHhOWaONdRYNIwRd0EgBJOB/D615D5v0rG8W3jWuhyzoQHidHH1DAit8P4kZlVqwpunHVpdSJ8P0Ixb5mfVa3CydDmpa8j+D3ju88VaOk96waZnYZUYHBr1mNty1/R9trnwA+iiigYUUUUAFFFFABRRRQAUUUUAJRS02kBzniT4haJ4RvI7XVLvyJ5E8xUEbMSucZ4B71lf8Lr8Jf9BF//AAHk/wDia4r4utt8f2n/AGDv/aprl/N+lfhWf8d47Ksxq4OlTi4xfW/Y+ywOS0cTh41ZN3Z67/wuvwl/0EX/APAeT/4mj/hdfhL/AKCL/wDgPJ/8TXkXm/SjzfpXz/8AxEzM/wDn3H8Tv/1ew/8AMz13/hdfhL/oIv8A+A8n/wATR/wuvwl/0EX/APAeT/4mvIvN+lHm/Sl/xEzMv+fcfxD/AFew/wDMzy3/AIKE/D7w1+0p8Jdmhzef4r0uQT2WIHDyryGiBK4Gc55x92tL9nvwzoPwU+HuheGYLhv9EhX7TOsDgyyHqx+Xr0/KvQPMHtR5o9quPidma/5dx/EP9XsP/Mz0zS/i54Us7dFfUX3Af8+8n/xNaP8Awuvwl/0EX/8AAeT/AOJryLzfpR5v0qX4mZn/AM+4/iH+r2H/AJmeu/8AC6/CX/QRf/wHk/8AiaP+F1+Ev+gi/wD4Dyf/ABNeReb9KPN+lH/ETMz/AOfcfxD/AFew/wDMz13/AIXX4S/6CL/+A8n/AMTUP/C9vBrSOialJIyHDhLSZtvGcHC+leUeb9Km+F823UPFXPW+X/0Utfo3A/FGJ4lxtTDYqCSjHm09Uv1PnM7y+GW0Y1KTd27anqf/AAvHwj/z/XH/AIBTf/EUf8Lx8If8/wDcf+AU3/xFZP2r3/Sj7V7/AKV+1fU6fmfG/WJmt/wvHwh/z/3H/gFN/wDEUf8AC8fCH/P/AHH/AIBTf/EVk/avf9KPtXv+lP6nS8x/WJmt/wALx8I/8/0//gFN/wDEUf8AC8fCH/P/AHH/AIBTf/EVk/avf9KPtXv+lH1Ol5i+sTNb/hePhD/n/uP/AACm/wDiKP8AhePhD/n/ALj/AMApv/iKyftXv+lH2r3/AEo+p0/Mf1iZrf8AC8fCH/P/AHH/AIBTf/EUf8Lx8If8/wDcf+AU3/xFZP2r3/Sj7V7/AKUfU6fmH1iZrf8AC8fCP/P9P/4BTf8AxFH/AAvHwh/z/XH/AIBTf/EVk/avf9KPtXv+lH1On5i+sTO+0HxNp/ibS4dR06fz7SbJSTaVzgkHg89Qa1Qc18x/A/4gXzalPosjj7HbtKY1xyMyknn8TX0nYzeZEp65ry61P2c3FHfTlzxTZaooorA1CiiigAooooAKKKKACiiigApKWkpAZPiTxTpvhHS21DVbj7LaK6x79jN8zHAGACeTXMf8Lx8If8/9x/4BTf8AxFVPjx8vg6yP/UVs/wD0ctVPtXv+lenh8PCpT55HBVrShPlRrf8AC8fCH/P/AHH/AIBTf/EUf8Lx8If8/wDcf+AU3/xFZP2r3/Sj7V7/AKV0/U6fmZ/WJmt/wvHwh/z/ANx/4BTf/EUf8Lx8I/8AP9P/AOAU3/xFZP2r3/Sj7V7/AKUfU6fmL6xM1v8AhePhD/n/ALj/AMApv/iKP+F4+EP+f+4/8Apv/iKyftXv+lH2r3/Sj6nT8x/WJmt/wvHwh/z/ANx/4BTf/EUf8Lx8If8AP/cf+AU3/wARWT9q9/0o+1e/6UfU6fmH1iZrf8Lx8I/8/wBP/wCAU3/xFH/C8fCH/P8A3H/gFN/8RWT9q9/0o+1e/wClH1On5i+sTNb/AIXj4Q/5/rj/AMApv/iKr3X7QPgmzaJZtUljeTOxTZzZbHJx8lUftXv+lcL44uc+N/B5B/5/P/Ra1lUwtOMW1c7MJOVasoS2PRP+GiPA/wD0E7j/AMAJ/wD4il/4aG8D/wDQSuP/AABn/wDiK4/7T70faPpXn8q7H0H1WB2H/DQ3gf8A6CVx/wCAM/8A8RSf8NEeB/8AoJ3H/gBP/wDEVyH2kDJJ4rxa1/bA+H918YLj4ef2oI9ViATznAETTd4w2euMfnS5US8NSjuz6Y/4aI8Df9BK4/8AACf/AOIo/wCGiPA3/QSuP/ACf/4iuQ+1e9H2r3p8qK+qw7M6/wD4aI8Df9BK4/8AACf/AOIpf+GhvA//AEErj/wAn/8AiK4/7T70faPpRyoX1WB1/wDw0R4H/wCgncf+AE//AMRR/wANEeBv+glcf+AE/wD8RXzR4s/a98AeD/itp3gK91MHVrpvKd4wGjhkJAVGbPBJzn0r2aO8EqK6MrKwBDDoRS5UJYak3ZHY/wDDRHgb/oJXH/gBP/8AEUf8NEeB/wDoJ3H/AIAT/wDxFch9p96PtPvT5UV9VpnpXhH4qeG/HN5dWujX7XFxaosksbwvGyqxIBwwHoa61XDDivj+68bX3hX4zTCzZQLy0tYpMrnK+Y/T8zX1N4d1H7dapIT2pShy6nnV4KlU5EblFFFZGAUUUUAFFFFABXnnx88KaJ42+E2vaJ4i1R9F0a6WIXF9GATFtmR1xkEcsoHTvXodcV8ZPh7F8VvhrrnhSe6ayj1KJVNwnJTbIrg/mooA3YZS8oxyp6GtaP7orz34deJNMvLAaJbeI7LxDqukKLa/e1lVnWUcEOoJKnPY16BG2VFUwJKKKKkAooooAKKKKACiiigAooooAKKKKACiiigAprdDTqRuhoA+UvgB/wAi34k/7GfV/wD0rkr06vMfgB/yLfiT/sZ9X/8ASuSvTq8up8bPusJ/Ah6BRRRWZ1hRRRQAUUUUAFFFFABSUtFAFLWv+QRe/wDXF/8A0E15/o8n/Eqs/wDrkv8AIV6BrX/IIvf+uL/+gmvNdKkxplqP+mS/yFfhHiT/ABMP6P8AQ9nL48yka3m0ebVTzaPNr8WuexyFvzaPNqp5tHm0XDkLfm0ebVTzaPNouHIW/No82qnm0ebRcOQt+bR5tVPNo82i4chb82sPxnJu8OXf0X+YrR82sbxdJu8P3Q9h/MV2YJ/7TT9V+ZjWhanL0O3/AGd2K6HHj/no/wDOvoi1/wBSK+dv2ef+QHH/ANdG/nX0Ta/6lfpX94z2R+HE1FFFZDCiiigAooooAKKKKACiiigApveq+qalBo+m3d/dPstrWF55Wx0RVLMfyBrmvh58WPCfxW0SLVvC2uWesWbAEm2mVmjJGdrgH5T7GgDzv4yPt8fWf/YN/wDaprkfNrqfjQ//ABXdkRyP7O/9qmuL82v484z/AOR7ifVfkj9byaN8DT/rqW/No82qnm0ebXxVz2+Qt+bR5tVPNo82i4chb82jzaqebR5tK4chb82jzaqebR5tO4chb82jzaqebR5tFw5C35tHw7uPL1DxL73q/wDopaqebUHgefbqPiLn/l7X/wBFrX7v4Q+9m9df9O//AG6J8BxjHlwdN/3v0Z6P9s96PtnvWH9q96PtXvX9Z+zPyPnNz7Z70fax61h/avej7V70ezDnNz7Z70fbPesT7V70fav9qj2Yc5t/ax60fbPesP7V70favej2Yc5ufbPej7Z71h/avej7V70ezDnNz7Z70fbPesP7V70faven7MOY83+C5/4rPUMf3n/9GV9e6Kc26/SvkD4K8+ML76v/AOjK+vtF/wCPdPpXzmM/iM9zD/w0alFFFeedIUUUUAFFFFABRRRQAUUUUAFNp1cPp/xo8G6l481XwbFr1mniXTXVJ9NllVZvmQOCqk5YYYc0AZnx+48F2h/6iln/AOjRXPfbPetz9oGQHwLAQc41K0/9GiuH+1e9fSYGPNQ+Z4uKdqpufbPej7Z71h/avej7V713ezOTnNz7Z70fbPesP7V70v2r3o9mHMbf2z3o+2e9Yf2r3o+1e9Hsx8xufbPej7Z71h/avej7V70ezFzm59s96PtnvWH9q96PtXvR7MOY3PtnvXEeNLnd4y8JnPT7X/6LWtv7V71x/jC4z4s8LnPT7V/6LWufEQtSketlb5sXBev5HT/ahR9qFY32w+tL9s9/1r54/RPZGx9qH4V8T/tt/smt4qZvH/giH7J4htT5t1Fb5UzY53D0Yf0r7A+2e/601rrcpB5GMYocbmdTDKpHlZ8ifsYftoN4mWDwD49mNn4itv3NteXHy+dj5Qjf7XT0719qi8DAEHIPT0r4M/bC/ZC/tqSbxz4Gi+za1CfOnt4BgyHruXHO7/GtD9j/APbMbxJ5PgXx3N9k8Q258m3upzt87H8LZ/i69+ahO2jOKnenP2NX5M+4/tQr5F/bI/bNh+G1nJ4O8GyrqHiu9XypJYfm+yZ4x7sR9etZ/wC19+2Qnw1tZvCXg+b7Z4suRsaaA7hbZGOg/i/UYryD9nH9nWe11AeNPGifb/EN43n29vL8zRk/xNnuefwob6IK0nKXsqW/Vl39mf8AZxu9PvofHPjPzL/xNdt51vbXB3NEx5Dt/tZ7e1foV4dursaRbi+2ibbg444rifB/hn+z8Xt6d90/IU/w/Wuv+2H1/Wqijqo4VU42Rs/ahR9qFY32s+tH2w+tUb+yPOvFsnmfF6A/9MLb/wBGNX174HYmxjB9K+Otfk8z4rwHr+6t/wD0Y1fYngf/AI8YvpV1PhR8tjly4hr0OzoooriOMKKKKACiiigAprIGUginUUAeMWv7Pfh/wr8Y734i6K1xp2o38TJf2cLHyLpjnDsueGGT0AqLw1qPxqt/GMUOq2vg6XwmblvMuI7m5F8ICTjauwJuHHU4r2iSESVWlsQRwKq4HlGqa58bF8aSLp2meB38Ii7ULNcXl2L37NuG4lQmzfjOBnGaf431v40ReKJl8Iab4JufDgCeXLrF3dpdHgb8rGhXrnHNeofYeMUgsCARRoB5t8Qta+MEOrwDwPp3g260v7OpmfXLq6jlE2TuCiNCNuMYzzT/ABtrXxdjtdE/4RDTvB9xdNbZ1RdYurlESfC8QmNSSmd3LYPAr0dbAr70Lp+2jQDzjX9a+Li+DdDfRNP8Hy+K2Lf2rDe3VytmnHHkMqlz2+8BRJrXxc/4V3FImn+D/wDhOzc4e3a6uf7O8jJ5DbPM34xxjHWvRhp+Dmj+z+SaNAPONI1r4vN4F1aTVNO8HR+MVlUadDa3VybF0+XJlYrvB+990HtS+E9a+Lsmha8fE2neD4NaWEHSI9Murl4JJMNxOXUMFzt+6CetejHT8tmlbT91GgHnHw/1r4uy3WoDxvp/g+1txbE2R0O6uZGafPAk8xBhMZ5HNQfD/W/jNL4iiXxvpvgq10Extuk0O7upLnfj5cCRAuM9efpXpzafux2pDp5IAo0A8s0HXPja3jKBNa0zwPF4U+0OJZrK8u2vBDztIVkCb/u5ycdcUuoa58bF8bSJZaZ4Hfwh9sCrPPeXYvvs27klQmzzNueM4zXqX9nnGKX+z/lxRoB5f4z1v40x+Kpl8J6Z4JuPDYKeXLq13dJdEYG/KohXrnHPpUvxC1r4wxa3EvgfTvBl1pPkKZH126uo5xLzuAEaEbcYx3616V/Z/GKVbAjPOaNAPOPHGtfF2K30X/hD9O8H3MzWudTGs3VzGqT4XIh8tDlM7uWwelHiDWvi4vg/Q20PT/B83ilt39qw311crZp6eQyruPP94CvR10/bR/Z/U5o0EfKH7L0mqyfD/V31uO0i1VvEeqm4SxZmhD/anzsLAHGfWvYK8w/Z+GPDPiMf9TPq3/pXJXp9eTU+Nn3mE/gQ9AooorM6gooooAKKKKACiiigAooooApa1/yCL3/ri/8A6Ca8u0xv+Jda8/8ALNf5CvUda/5BF7/1xf8A9BNeR6dIRYWwz/yzX+Qr8G8S3aph/R/oe/lUebmNTfRv96qeYfWjzD61+Jc3kfQchb3+9G/3qp5h9aPMPrTv5ByFvf70b/eqnmH1o8w+tF/IOQt7/ejf71U8w+tHmH1ov5ByFvf70b/eqnmH1o8w+tF/IOQt7/eud+IEl+PCN+dMW3kvto8pbliIycjrjmtfzD61l+JpP+JLcD2H8xXfgJcuLpO32l+Zz4mH7mduzNv9mK88X/8ACFXB1a30aPVg7/ZktJZWgJycbyRkdule4fDnUviTJBqA8a2Xhm1IB+w/2HcTybumPM8xRj8K8z/Z9gEmixnP/LRv519CRWu+Ne3Ff3nLZH4F1OH8D6r8TZvEF8vjCx8L22iAn7LJo9xcPcH03iRQo7dDTNB1T4pN42uE1qw8KxeEc/uZrO4uDekdtysoT8jXffYyepz+FH2M+v6VAzgZtV+KI+IAjisPCp8FZ/173FwNQx/u7dnT3o8Uar8UY/GFonh6w8LTeFiR9om1G4uFvAMc7FVSv5mu++x9s8fSj7H7/pRoBwvj7VPiZDq1gPBdl4YudNYr9rbW7ieOUDIzsEakHj1p/wAQNU+JENnp58FWXhq5uyF+2LrlxPGinv5flqSefWu3FoV6HFH2MjocUaAcX4u1T4ixeE7J/DNn4cn8SnH2mLVJ50tB0zsZFLHv1FOvtT+Ia+A0ks7Pw6/jPHzQTTziwzjswXf+ldl9jIzg4+lH2Pvn9KNAON0nU/iE3gWV9Ss/DqeMcHy4Laec2JPuxXf+lN8Gap8RZfDF6/iuz8OW/iHn7JHpM872rdcb2dQw7dBXafY++f0o+x8g55o0A878M6h8QrvQdeX4gWHha2iMEot10m4uJI3Xaf8AXeYoIGM5254r8OLDXvih4b/aK8SL8KXuIb1NVuPs9roTs9lxIcbBJ8rL0xuFf0AaxoMOtaTeafdDfbXcLwSgcEqylT+hryjwt+z34R+FOgro/g7QrbSLXA8xoYwHlYfxOR1b3ppK5LPDPAfiT4jeItM0uf4mQafDr62IUfY2Yuyb85lBAAfJP3eMYrrt/vWn8QtGbQvE9rE3Bey3f+RDWD5h9a/jfjV2z7E27/oj9oyKN8vpv+ty3v8Aejf71U8w+tHmH1r4e/ke9yFvf70b/eqnmH1o8w+tF/IOQt7/AHo3+9VPMPrR5h9aL+Qchb3+9G/3qp5h9aPMPrRfyDkLe/3o3+9VPMPrR5h9aL+Qchb3+9UPCk3l6lr/AL3S/wDotak8w+tZfh+fy9S1vnrcL/6Atfv3g372c11/07/9uR+c8cLlwNN/3v0Z2n2r3o+1e9Y32uj7XX9h+zPxHmNn7V70favesb7XR9r96PZsOY2ftdH2usb7XR9ro9mHMbP2r3o+1e9Y32v3o+10ezDmNn7V70fa/esb7XR9ro9mHMbP2uj7V71jfavej7XS9mHMeM/D+++LVv411I+HtO8IzW+9vLOpXNyjFd/fYpGc19FaZ4j/AGjVgXZo3w0xj+K/vh/7Trzv4Kx+b4qu+3Df+h19ZaTpYa2T6V8Zi1+9dz6fD/w0eP8A/CTftI/9AX4Z/wDgff8A/wAbo/4Sb9pH/oC/DP8A8D7/AP8Ajde3/wBjj0o/scelcOh0niH/AAk37SP/AEBfhn/4H3//AMbo/wCEm/aR/wCgL8M//A+//wDjde3/ANjj0o/scelPQDxD/hJv2kf+gL8M/wDwPv8A/wCN0f8ACTftI/8AQF+Gf/gff/8Axuvb/wCxx6Uf2OPSjQDxD/hJv2kf+gL8M/8AwPv/AP43R/wk37SP/QF+Gf8A4H3/AP8AG69v/scelH9jj0o0A8Q/4Sb9pH/oC/DP/wAD7/8A+N0f8JN+0j/0Bfhn/wCB9/8A/G69v/scelH9jj0o0A8Q/wCEm/aR/wCgL8M//A+//wDjdfmH/wAFFtS8dW/x40O9uYdD0vxi9nmdvBd1cu7NvwvmlwpDY24A4xiv2r/sceleWXX7Nvgq08fav44j0K3m8VaiVMuoTKGkXaoUBDjgYUcUaCPij9nTxx8ftZ+H9rY/EqKJvDi3VuYJr/5b0AOCm0IMFSepY5r6i+1e9Xfiz4VfSfCK3bcBb63x/wB/BXM/a6+ty6CdHTufPY2VqvyNn7V70favesb7X70fa69T2Zw8xs/a6PtdY32uj7XR7MOY2ftdH2r3rG+10fa6PZhzGz9q96PtXvWN9ro+10ezYcxs/a6PtfvWL9q96X7XR7MOY2ftXvXJeLbjPibw4c9PtP8A6Atan2uuX8U3W7xBoB9PtH6oK4sZDloSZ7mSPmzCmv62N/7UfWj7WfWsj7V9fyo+1/5xXyR+x+xNf7WfWj7UfWsj7X/nFJ9roD2JsNdbgVOGHp61+Z/7dmm+DNH+IlpN4SnMXiZm3XkVjwqHnByOjdOle8/tV/taRfDuzm8L+FZftXim4XaZYTvFvntx/F04+vFeS/s//AWa3vh4v8Yx/b9cvD5sFvMNxjJ43MPXr6dqzl72iPBxlsRL6vSV2t32PNf2U5vDt58SJpvFkktx4jZx9iF8dytISc53Hls4xmv0t8I+GV09UvLoK85GUXsvp+lfHX7R37KV5qWnnxp4Yj8jW4czT20Iwz453D/a6/nXTfso/tcf8JEsXgzxrN9l163/AHNvcz/L52P4Wz0br+VKOjsyMLD6rUVGurX2fc+zftZ9aPtR9ax/tdL9q+v5VqfQ+x8jX+1H1o+1H1rI+1fX8qPtX1/KmHsTmNRk8z4oQH/pnb/+jDX2b4H/AOPGL6V8VNJ5nxKhP+xbj/x819q+B/8Ajxi+lVU+BHwOZK2Lkjs6KKK4jzgooooAKKKKACiiigAooooAKK4X4sfG7wb8EdB/tfxjrUGkWZOFDndI/rtQcnGewrzv4U/t0/Br40a8ujeF/FaT6i5wkN1A9uXPou8DJ+lAHvuaK/Pn9tD48eO/Af7XHwy8O+HfFN/pWgagLc3lhbOBHNulYHcMc5FfT/xd/aw+HHwGht5PGuvpphn/ANXFHG00hHrsUE4/DvTsB7Pupa/Mv9pD9tPUPEvx5+FB+Fnj66TwlqhhF7a2bhFlJlYESIRkHAHBxX6O2OsJdttQ5osK5rZpN1fntD8fPHkf/BSi98Dy+KtQPg2NjjRy48gfud3THrzXX/Bf4p6jq37X3jLSJ/jBc+I7CD7Rs8HyWMscdnhjjErDYdvseaOVhc+3KK+dbP8Ab0+DU0PiKV/FX2caA5jvluLaSNgwIBVAwBc8j7ue9dn8Ff2oPh3+0JZ3Nx4I11dUFscSxyRtDIue5RgDj8KLAerZpa/PH9i39obxr4u/aN+LOl+LfFl9qugaT5xtbW7kHlW4WVQMccYBNfV/wo/ak8A/GbxRrGg+FNTuNUvdKbZcyraSrbg88LMV2N0PQmizGev0U1XDdKdSAKax+U06mHvQB8qfAD/kW/En/Yz6t/6VyV6dXmPwB/5FvxJ/2M+r/wDpW9em7fevLqX52fc4R/uIegtFJt96NvvWWp13FopNvvRt96NQuLRSbfejb70ahcWik2+9G33o1C4tFJt96XHvT1Fcpa1/yCL3/ri//oJrxyxf/QoP9xf5V7HrXGj3vP8Ayxf/ANBNeIWch+yw/wC4P5V+CeJjtUw3o/0Pp8lXNzml5lHme/61T3mjea/Eed9j6n2Zc8z3/WjzPf8AWqe80bzRzsOQueZ7/rR5nv8ArVPeaN5o52HIXPM9/wBaPM9/1qnvNG80c7DkLnme/wCtHme/61T3mjeaOdhyFzzPf9azvED50mcew/mKl3mqGtsTps30/rXdgZf7VS/xL8zlxUbUJ+jPVv2ef+QHH/10b+dfRNr/AKlfpXzt+zz/AMgOP/ro386+ibX/AFK/Sv74lsj+derJqKKKyGFFFFABRRRQAUUUUAFFFFACVE1ujdRzU1J9KQHz38fAI/G2n44/0D/2oa868z3/AFr0D9oY7fGmnH/pwP8A6MNeZ7z71/G/GsrZ9iV5r8kfuHD8b5bT+f5lzzPf9aPM9/1qnvNG818Pzs+j5C55nv8ArR5nv+tU95o3mjnYchc8z3/WjzPf9ap7zRvNHOw5C55nv+tHme/61T3mjeaOdhyFzzPf9aPM9/1qnvNG80c7DkLnme/61iaXceXqmr89Z1P/AI4Kv7zXPW023U9UGf8Alqv/AKCK/oTwVfNnddf9O3/6VE/L/EBcuXU3/e/RnS/ah60fah61jfaff9aPtPv+tf2j7M/AOc2PtXvR9q461j/aff8AWj7T7/rT9mPnNj7V70favesf7T7/AK0faff9aPZhzmx9q460favesf7T7/rR9p9/1o9mHObP2oetH2oetY32n3/Wj7T7/rR7MOc2PtXvS/ah61jfaff9aPtPv+tHsxc5q/Aw/wDFUXJ9j/6HX2Fov/Hun0r47+BP/IzT+6k/+P19iaL/AMe6fSvzzHfxZep9nh/4UTUooorzTpCiiigAooooAKKKKACiiigAqN4lfqKkpvYmkB5R+0hGsPw2YgY/0+2/9GCvBvtQ9a95/aYbHwxkPf7bbf8AowV82/avf9a+3yaPNh36nzOZStVXobP2rjrSfavesf7T7/rR9p9/1r3/AGZ5XObP2oetH2oetY32n3/Wj7T7/rR7MOc2PtXvR9q461j/AGn3/Wj7T7/rR7MOc2ftQ9aT7V71j/aff9aPtPv+tHsw5zY+1e9H2r3rH+0+/wCtH2n3/Wj2Yc5s/ah61zfia43a3ohz087/ANBFW/tPv+tc/wCIrj/ibaQc9PO/9BFefj4Ww0z6Lh182Z0V5/obP2r3o+1e9Y32s+po+2e9fCH9B+yRs/avc/nXzV+1F+1PH8P7eTwx4ZlF14muR5TsnP2cNx/30a98+1+9eIeKPgB4aTx9J4yhsDNfy4zESCgk/v7fXp37UpXPNx1Gs6VqGl9/Q8c+AvwPltbweLvFyyXuvXDedDbz/Pt3HO5s9z2r7B8J+H0sVW6ulBnblV/uj6Vk+GNBSw23NyA1weVHHy//AF66f7Z70RjYWDy+NCCRsNchlKnlT1B6flXyH+1N+zD/AGlM/jLwZGbXVYP3s0EGVLkfxLjoenT0r6k+2e9I10JFKsdykYIPQ0Sjc3xOCp4mm4zR82/sr/tWHxGsPg3xhMbbXYMQwXE3HnY4Cn/a6V9Wfa/f9a+cfGn7Ovhy48eW/jK1tWjvIX81oIyArODkMR6/jXtOk6hNLYQmZPKk28jNKN9jmwVGtTTpV9bbPyOl+1e9H2r3rG+2e9H2z3qz1FRIrOTzPiJCevywf+hmvt3wP/x4xfSvhnRZPM8exH/rh/6HX3N4H/48YvpVVPgR+V5srY6aOzoooriPKCiiigAooooAKKKKACiiigD8pf2wo4/id/wUX8I+D/Fczt4XtbOGeKzkY+U7hnOcdDu2gH16Vg/8FJPAfhf4a3ngfxP4O0218Na5BdpDH/Zca2+5cn5iEAyeBz15r7H/AGyv2KbD9oq+0zxNpWqP4b8aaVt+zanH3VTlQ4AJIByRj1NfPvhX9gTxTrvjzSte+MPxAfx1HpL5trNEKoQD918r0+nNbR2Ezzz9q7WrvW/2pvgreXgK3DWlpvDdc72qDxppOnfEz/gplBpniyJb3S7K0glgsrj5omYJHxtPBB5yO9fRfxq/ZC1r4sfGfwh430/WLPTrDQRGHs54nLyBGJ+Ujgde9ZP7Sn7EGqfF/wCIGn+PvB3iUeEfFtmFVrt0LK+0AKeB1AFXYR86ftbeAvDHgf8AbB+HL+G7K30k3bQyzWNogjjDeYw3hRwvAA/Cv1O8ESvJI24nNfn5c/8ABPvxdfePvC/i/VfiAut63psyS302oxsTLtOdse0DaOnWv0E8FxssjkggGk7WEfnhd/8AKVTWG7gn/wBEVQ/ZrnMH/BQb4juDg4uv/QmzX0jq/wCx1rlr+15ffGX+37F9LuOmmCJ/PH7vb977vvWD8Mf2Sdb8BftHeKPibca1Z3Vhq4lCWMcTiWPeSRknjvSiDPl/9iv4Z+H/AB3+0p8U7jxFYRavbWEkkkFndKJIfMMijcUPBO1j1roP2cbO3+Gf/BRnxbo2gR/YNIuLe5mazh+WMYDYAUcADsK+gv2b/wBkvWvgf8RvG3ii/wBZs9Tt/EDMYoLaJ1aLLq3zE8H7vap/C/7IOteHP2qtU+L8muWU2mXltNAumpE4mUuDglvu45qtBH5xXmt/EO08efGiHwVazPazvK2q3NvnzI7fzlyFIGQc7emK/Sn/AIJ86p4Qi/Z98Pf8IdCkRdduoPj94bnA8wMep7d/Wue+CX7HOq/C/wCIXjzxDq2r2Wraf4lEiLaQwuHjDMD8xbg8DtVv9mf9lHxL+zr8TPEl9Za/aXPgbVJDJb6P5T+bbdcYP3c8jP0osij7h01maNSav1l6LP5kCcY4rUrne5QU1h1p1FID5D+L37LWmWGoTXnhlvElu99PNeXMdlrFykXmu+5iEDgDJJ6V5LJ8A/ECMR9s8Wf+Dm5/+Lr9Dbm0WZSCoNZknh+Fz/qxWsZRW6HzT6M+A/8AhQuv/wDP74s/8HNz/wDF0f8AChdf/wCf3xZ/4Obn/wCLr74/4RyH/nnR/wAI5D/zzq+aH8qDmn/Mz4H/AOFC6/8A8/viz/wc3P8A8XR/woXX/wDn98Wf+Dm5/wDi6++P+Ech/wCedH/COQ/886OaH8qDmn/Mz4H/AOFC6/8A8/viz/wc3P8A8XR/woXX/wDn98Wf+Dm5/wDi6++P+Ech/wCedH/COQ/886OaH8qDmn/Mz4H/AOFC6/8A8/viz/wc3P8A8XR/woXX/wDn98Wf+Dm5/wDi6++P+Ech/wCedH/COQ/886OaH8qDmn/Mz4H/AOFC6/8A8/viz/wc3P8A8XR/woXX/wDn98Wf+Dm5/wDi6++P+Ech/wCedH/COQ/886OaP8qFzT/mZ8FQ/AnX4ZUcXXiptpB2vq9yynHYgvyPat+Hwb4u+13CNYXQhUL5fy+o5r7W/wCEch/550N4chIx5debjMuwGYNPFUYzttdHRSxWIoX9nUav5nxn/wAIX4p/58rv8jR/whfin/nyuvyNfZn/AAjkP/POj/hHIf8AnnXn/wCr2Tf9AsPuOj+0cb/z+f3nxn/whfin/nyuvyNH/CF+Kf8AnyuvyNfZn/COQ/8APOj/AIRyH/nnR/q/k3/QLD7g/tLG/wDP5/efGf8Awhfin/nyuvyNH/CF+Kf+fK6/I19mf8I5D/zzo/4RyH/nnR/q/k3/AECw+4P7Sxv/AD+f3nxn/wAIX4p/58rr8jR/whfin/nyuvyNfZn/AAjkP/POj/hHIf8AnnR/q/k3/QLD7g/tLG/8/n958Z/8IX4p/wCfK6/I0f8ACF+Kf+fK6/I19mf8I5D/AM86P+Ech/550f6v5N/0Cw+4P7Sxv/P5/efGf/CF+Kf+fK6/I0jeB/FEilWsLplPYg19m/8ACOQ/886P+Edh/wCedOOQ5NCSksLC68hPMMa1Z1X955L8D/Dt3o2jxR3cLwSb2JRxg9a90tf9StZ9rpS25yqYNaca7VxXuSld6HnLQfRRRUFBRRRQAUUUUAFFFFABRRRQAUlLRQB5n8Yvh7Z+KNPk1EQTT6pbw+XB5MhXjdnoOvU180X3gPxTbzELY3YXPoa+35IxIuDWXPokUrZKAmvHq5LleIqOtXw8ZSe7aO2njsVSioU6jSXmfFf/AAhfin/nyuvyNH/CF+Kf+fK6/I19mf8ACOQ/886P+Ech/wCedR/q/k3/AECw+4v+0sb/AM/n958Z/wDCF+Kf+fK6/I0f8IX4p/58rr8jX2Z/wjkP/POj/hHIf+edP/V/Jv8AoFh9wf2ljf8An8/vPjP/AIQvxT/z5XX5Gj/hC/FP/Pldfka+zP8AhHIf+edH/COQ/wDPOj/V/Jv+gWH3B/aWN/5/P7z4z/4QvxT/AM+V1+Ro/wCEL8U/8+V1+Rr7M/4RyH/nnR/wjkP/ADzo/wBX8m/6BYfcH9pY3/n8/vPjP/hC/FP/AD5XX5Gj/hC/FP8Az5XX5Gvsz/hHIf8AnnR/wjkP/POj/V/Jv+gWH3B/aWN/5/P7z4z/AOEL8U/8+V1+RrK1D4deKGvIGj068PmyfvioPTaeTX3D/wAI5D/zzo/4R2Hp5delgMDgsrqOrgqKpyel0raHJiK9bFxUK83JLufEv/CtfEP/AD4335ml/wCFZ+If+fG9/M19sL4bhH/LMUv/AAjkP/POvd+v1v5med9VpfynxN/wrPxD/wA+N7+Zo/4Vn4h/58b38zX2z/wjkP8Azzo/4RyH/nnT/tCv/Mx/VaX8p8Tf8Kz8Q/8APje/maP+FZ+If+fG9/M19s/8I5D/AM86P+Ech/550f2hX/mYfVaX8p8Tf8Kz8Q/8+N7+Zo/4Vn4h/wCfG9/M19s/8I5D/wA86P8AhHIf+edH9oV/5mH1Wl/KfE3/AArPxD/z43v5mj/hWfiH/nxvfzNfbP8AwjkP/POj/hHIf+edH9oV/wCZh9VpfynxN/wrPxD/AM+N7+Zo/wCFZ+If+fG9/M19s/8ACOQ/886P+Ech/wCedL6/W6yYfVaX8qPmf4M+B9T0XXZZLuzmt08vhpFxn5ga+ptHUrAg9qrQ6HHGwITFalvD5Y6YrhqVHUd2dEYqKsieiiisCwooooAKKKKACiiigAooooAKSlooA57xx4Q03xxoMml6pHJLaM6yFY5ChypyORXyL4q+D+rWOo3CWOn3626yMEy7HKgnHevtplDcGs660uOY/MoNddHFVaK5YSsjCdGFR3kj4Y/4Vn4i/wCfG+/M0f8ACs/EP/Pje/ma+2m8Owsf9XSf8I5D/wA866/7QrfzMz+q0v5T4m/4Vn4h/wCfG9/M0f8ACs/EP/Pje/ma+2f+Ech/550f8I5D/wA86P7Qr/zMPqtL+U+Jv+FZ+If+fG9/M0f8Kz8Q/wDPje/ma+2f+Ech/wCedH/COQ/886P7Qr/zMPqtL+U+Jv8AhWfiH/nxvfzNH/Cs/EP/AD43v5mvtn/hHIf+edH/AAjkP/POj+0K/wDMw+q0v5T4m/4Vn4h/58b38zR/wrPxD/z43v5mvtn/AIRyH/nnR/wjkP8Azzo/tCv/ADMPqtL+U+Jv+FZ+If8AnxvfzNU9U+FGv3Nuz/2ffNLGrFDls5x9a+5f+Ech/wCedH/COw/886mWNqzXLKWhpToxozVSno11PhWD4Pa01vGX0/UN5UE/O3XH1qT/AIU7rP8Az4ah/wB/G/xr7l/4RuHdny6X/hHYf+edc/tvI9H63if+fj+8+Gf+FO6z/wA+Gof9/G/xpD8G9YOB9g1D/v43+Nfc/wDwjkP/ADzo/wCEch/550e28hfW8T/z8f3nwz/wp3Wf+fDUP+/jf40f8Kd1n/nw1D/v43+Nfc3/AAjkP/POj/hHIf8AnnR7byD63if+fj+8+Gf+FO6z/wA+Gof9/G/xo/4U7rP/AD4ah/38b/Gvub/hHIf+edH/AAjkP/POj23kP63if+fj+8+GP+FN6x0+wahj/ro3+NL/AMKc1n/nw1D/AL+N/jX3N/wjkP8Azzo/4RyH/nnR7byF9bxP/Px/efDP/CndZ/58NQ/7+N/jR/wp3Wf+fDUP+/jf419zf8I5D/zzo/4RyH/nnR7byH9bxP8Az8f3nxb4X+FGr6brltP/AGddj96m55CW4DCvsXwfbNb2kat1q+vh+Jf+WfFaNraCDgLgVlOpzI5pSlOXNN3ZdooorAQUUUUAFFFFABRRRQAUUUUARTQiVcHmsqbQIZZNxQZ+lbVFO4GXHpEUce0KMULpMSqQF61qUU7isYTeHYWbPlirlnpcdr9xcVo0UrgVLqyWdcMMiqv9jQ7cbOK1aKLsZlf2PDjG2ntpcTJtKgj6VpUUXYGUdGhK42UHR4toXbwORWrRRdisVrW1FuuFGBVmiikMKKKKACiiigAooooAKKKKACiiigAooooAKKK+MNW/4Kh+AtD8YeLPDd54b1uPUtBmkt1jjCSteujFcRquTzg9aAPs+kzXyp8Bf+Ch3gj483GvWFnpGqaFrmlW0lz/AGTqIUTzqiliEA74B4NfKnhv/gqH4k1X9q68abQvEb+B3tjZReFoYYzLFONo89+MgZB796dmB+q9FfHvi7/gpP4J8C/EjWvB2q+H9XjutMgWY3CFGWUsoIRVB3buQOle9fAz41Wnxw+H1l4rttIv9BiuZHjFjqahZ129yB60WYHpFFNVgwyKdSAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooARvumvyB/Zql0GD/goH8Rn1PyTqQmvfsXn4+9545XP8Xpiv1+b7pr8YPh38G9H+Nf7WXxe0XVZbizZbq7lgvbORklhcSkBlIPuaqO4nsdHcXFn/wAPOtKn8MCPzlSY3xt8bfM8qTO7HfGOtXPgdeD/AIeNeNm3rHLJp90Bk4J+Za+i/wBnT9jPwt8BdWvtasru+8Q67djadR1MhpEX0BB/+vVf4mf8E9fDfxA+Ji/ECPXtY8P6m20zW+nMFSbH94nnnvXRsSfN8Hh+y17/AIKaym+gS5S2dpRHIAVLeScZH15r9O/C0lxcMDnCZ6CvnLRP2PdJ0n47P8U01nUX1UoIzYuieT9zZnON1fUXhGze3h5GOal7AdRbqVj5qakA28Utc5YUUUUAFFFFABRRRQAUUUUAFFFFABRRRQAUUUUAFFFFABRRRQAUUUUAFFFFABRRRQAUUUUAFFFFABRRRQAUUUUAFFFFABRRRQAUUUUAFFFFABRRRQAUUUUAFFFFABRRRQAUUUUAFFFFABRRRQAUUUUAFFFFABRRRQAUUUUAFFFFABRRRQAUUUUAFFFFABRRRQAUUUUAFFFFABRRRQAUUUUAFFFFABRRRQAUUUUAFFFFABRRRQAUUUUAFFFFABRRRQAUUUUAFFFFABRRRQAUUUUAFFFFABRRRQAUUUUAFFFFABRRRQAUUUUAFFFFABRRRQAUUUUAFFFFABRRRQAjdK8pX4E+C/DfiO/8QaD4W0zTNc1AsbvULW3VJp9xydzDrzXq9MaNW6imnYDm9E0FLeIB0G6tw2CeXtxx6dqsrGq9BTqG2xGeukxDPyCrENqsPQYqxRQMKKKKQBRRRQAUUUUAFFFFABRRRQAUUUUAFFFFABRRRQAUUUUAFFFFABRRRQAUUUUAFFFFABRRRQAUUUUAFFFFABRRRQAUUUUAFFFFABRRRQAUUUUAFFFFABRRRQAUUUUAFFFFABRRRQAUUUUAFFFFABRRRQAUUUUAFFFFABRRRQAUUUUAFFFFABRRRQAUUUUAFFFFABRRRQAUUUUAFFFFABRRRQAUUUUAFFFFABRRRQAUUUUAFFFFABRRRQAUUUUAFFFFABRRRQAUUUUAFFFFABRRRQAUUUUAFFFFABRRRQAUUUUAFFFFABRRRQAUUUUAFFFFABRRRQAUUUUAFFFFABRRRQAUUUUAFFFFABRRRQAUUUUAFFFFABRRRQAUUUUAFFFFABRRRQAUUUUAFFFFABRRRQAUUUUAFFFFABRRRQAUUUUAFFFFABRRRQAUUUUAFFFFABRRRQAUUUUAFFFFABRRRQAUUUUAFFFFABRRRQAUUUUAFFFFABRRRQAUUUUAFFFFABRRRQAUUUUAFFFFABRRRQAUUUUAFFFFABRRRQAUUUUAFFFFABRRRQAUUUUAFFFFABRRRQAUUUUAFFFFABRRRQAUUUUAFFFFABRRRQAUUUUAFFFFABRRRQAUUUUAFFFFABRRRQAUUUUAFFFFABRRRQAUUUUAFFFFABRRRQAUUUUAFFFFABRRRQAUUUUAFFFFABRRRQAUUUUAFFFFABRRRQAUUUUAFFFFABRRRQAUUUUAFFFFABRRRQAUUUUAFFFFABRRRQAUUUUAFFFFABRRRQAUUUUAFFFFABRRRQAUUUUAFFFFABRRRQAUUUUAFFFFABRRRQAUUUUAFFFFABRRRQAUUUUAFFFFABRRRQAUUUUAFFFFABRRRQAUUUUAFFFFABRRRQAUUUUAFFFFABRRRQAUUUUAFFFFABRRRQAUUUUAFFFFABRRRQAUUUUAFFFFABRRRQAUUUUAFFFFABRRRQAUUUUAFFFFABRRRQAUUUUAFFFFABRRRQAUUUUAFFFFABRRRQAUUUUAFFFFABRRRQAUUUUAFFFFABRRRQAUUUUAFFFFABRRRQAUUUUAFFFFABRRRQAUUUUAFFFFABRRRQAUUUUAFFFFABRRRQAUUUUAFFFFABRRRQAUUUUAFFFFABRRRQAUUUUAFFFFABRRRQAUUUUAFFFFABRRRQAUUUUAFFFFABRRRQAUUUUAFFFFABRRRQAUUUUAFFFFABRRRQAUUUUAFFFFABRRRQAUUUUAFFFFABRRRQAUUUUAFFFFABRRRQAUUUUAFFFFABRRRQAUUUUAFFFFABRRRQB//9k=)

**Deletion at End of List:**

**Algorithm:**

**Step 1:** IF HEAD = NULL

Write Underflow

Go to step 8

[End of If]

**Step 2:** SET PTR = HEAD

**Step 3:** Repeat Steps 4 and 5 while PTR -> NEXT! = NULL

**Step 4:** SET TEMP = PTR

**Step 5:** SET PTR = PTR -> NEXT

[END OF LOOP]

**Step 6:** SET TEMP -> NEXT = NULL

**Step 7:** FREE PTR

**Step 8:** EXIT

Diagram

Description automatically generated

**Deletion at Specific Location of List:**

**Algorithm:**

**Step 1:** IF PTR = NULL

Write Underflow

Go to step 10

[End of If]

**STEP 2:** SET PTR = HEAD

**STEP 3:** SET I = 0

**STEP 4:** REPEAT STEP 5 TO 7 UNTIL I<POS

**STEP 5:**  TEMP=PTR

**STEP 6:** PTR=PTR->NEXT

**STEP 7:** I = I+1

[END OF LOOP]

**STEP 8:** TEMP → NEXT = PTR → NEXT

**STEP 9:** FREE TEMP

**STEP 10:** EXIT

**![Diagram

Description automatically generated](data:image/jpeg;base64,/9j/4AAQSkZJRgABAQEASABIAAD/4RyyRXhpZgAATU0AKgAAAAgABgALAAIAAAAmAAAIYgESAAMAAAABAAEAAAExAAIAAAAmAAAIiAEyAAIAAAAUAAAIrodpAAQAAAABAAAIwuocAAcAAAgMAAAAVgAAEUYc6gAAAAgAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAFdpbmRvd3MgUGhvdG8gRWRpdG9yIDEwLjAuMTAwMTEuMTYzODQAV2luZG93cyBQaG90byBFZGl0b3IgMTAuMC4xMDAxMS4xNjM4NAAyMDIzOjAxOjEyIDExOjA5OjIyAAAGkAMAAgAAABQAABEckAQAAgAAABQAABEwkpEAAgAAAAMwNQAAkpIAAgAAAAMwNQAAoAEAAwAAAAEAAQAA6hwABwAACAwAAAkQAAAAABzqAAAACAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAMjAyMzowMToxMiAxMTowODoyOQAyMDIzOjAxOjEyIDExOjA4OjI5AAAAAAYBAwADAAAAAQAGAAABGgAFAAAAAQAAEZQBGwAFAAAAAQAAEZwBKAADAAAAAQACAAACAQAEAAAAAQAAEaQCAgAEAAAAAQAACwUAAAAAAAAAYAAAAAEAAABgAAAAAf/Y/9sAQwAIBgYHBgUIBwcHCQkICgwUDQwLCwwZEhMPFB0aHx4dGhwcICQuJyAiLCMcHCg3KSwwMTQ0NB8nOT04MjwuMzQy/9sAQwEJCQkMCwwYDQ0YMiEcITIyMjIyMjIyMjIyMjIyMjIyMjIyMjIyMjIyMjIyMjIyMjIyMjIyMjIyMjIyMjIyMjIy/8AAEQgATgEAAwEhAAIRAQMRAf/EAB8AAAEFAQEBAQEBAAAAAAAAAAABAgMEBQYHCAkKC//EALUQAAIBAwMCBAMFBQQEAAABfQECAwAEEQUSITFBBhNRYQcicRQygZGhCCNCscEVUtHwJDNicoIJChYXGBkaJSYnKCkqNDU2Nzg5OkNERUZHSElKU1RVVldYWVpjZGVmZ2hpanN0dXZ3eHl6g4SFhoeIiYqSk5SVlpeYmZqio6Slpqeoqaqys7S1tre4ubrCw8TFxsfIycrS09TV1tfY2drh4uPk5ebn6Onq8fLz9PX29/j5+v/EAB8BAAMBAQEBAQEBAQEAAAAAAAABAgMEBQYHCAkKC//EALURAAIBAgQEAwQHBQQEAAECdwABAgMRBAUhMQYSQVEHYXETIjKBCBRCkaGxwQkjM1LwFWJy0QoWJDThJfEXGBkaJicoKSo1Njc4OTpDREVGR0hJSlNUVVZXWFlaY2RlZmdoaWpzdHV2d3h5eoKDhIWGh4iJipKTlJWWl5iZmqKjpKWmp6ipqrKztLW2t7i5usLDxMXGx8jJytLT1NXW19jZ2uLj5OXm5+jp6vLz9PX29/j5+v/aAAwDAQACEQMRAD8A9/ooAKKACigAooAKKACigAooAKKACigAooAKKACigAooAKKACigAooApXuopZSRxmJ5GcFhtxwBj1+tV/wC3F/59ZvzX/GsJ14wdmawoykroP7cX/n1m/Nf8aP7cX/n1m/Nf8an61Ar6vMu2V5FfWkdxCTscZGe3tViukwCigAooAKKACigAooAKKACigAooAKKACigAooAKKACigDD1r/j/ALf/AK5P/Na5DW9R1ix1KP7DafabfycumOS2exry66TrO56NFtUtDJHirxHth36AwLoSxGSFOeP0rpdBuLy60iKa/TZcMW3Ltxjk44+lZSjFLRmkZSb1Oh8Kf8gCH6n+Zrbr2Fsjy3uFFMQUUAFFABRQAUUAQ3dx9ksp7kqX8qNpNo6nAziuGb4nxJPHC+i3gZlZnIwQoGT1+goAsL4+a60Se/tdOlRobiKFkl6ncwyfwBqjJ8TpEghk/sef5mG49dw/ix6ckdadgPQYJRPAkq9HUEVJSAKKACigAooAKKACigDH1uyvLh4ZbLyi6AqVkzjBx6fSsn+z9f8A+eVn+bVzVMMqkuZs6KeIcI8pBFb67LPNF5NmDEQDy3pU/wDZ+v8A/PKz/Nqj6lHuX9afY29BsptP0mK3n2+Yuc7enWtOuzY5AooAKKAKl5dSxSRwQKplkyfn6ADvUWdW/wCnT8m/xoAM6t/06fk3+NGdW/6dPyb/ABoAM6t/06fk3+NGdW/6dPyb/GgBD/apBBFmQeo2t/jUBsrwnJt7Akf7B/xpgKLS+VWUQ2IVuo2Nz+tJ9ivMY+z2H/fB/wAaAJ0XVEUKoswo6AK3+NLnVv8Ap0/Jv8aQBnVv+nT8m/xozq3/AE6fk3+NABnVv+nT8m/xozq3/Tp+Tf40ABbVgCdtq2OwDDP61atLhbq2SZRgMM49KAJqKACigAooAz7P/kI33++P5VoUAFFABRQAUUAZ9x/yGrb/AK5P/MVoUAFFABRQAUUAFFABRQAUUAFFABVDRv8AkFw/T+tAF+igAooAKKAM+z/5CN9/vj+VaFABUVzOtrbSTuCVRckDrQBR/tpP+fS4/wDHP/iqP7aT/n0uP/HP/iqnnRt7CfYP7aQf8ulx/wCOf/FU3+37QjO2T8h/jXHiswpYa3O9yo4WpIqS6vA+pQzBJNiRspOB1JHv7Vb/ALftP7sn5D/GuP8AtzC9y/qdQrX/AIw0fToo2uJ3BkUsqrGTjtyQMLk8c1Sg+IGk3SXLQR3D/Z7N7p/kwMJ1UE8E/TivYhLnipLqcrVnYil+JOgw208kjTCSJZCY/KPOwyDG77oJ8p8ZPat+LWrOfTI9QhZnt5QCjBcZB6HmprVFSg5y2QRi5OyI/wC37T+7J+Q/xo/t+0/uyfkP8a8n+3ML3Or6nVA+ILMdRJ+Q/wAan/tSP/nlJ/47/jXo4LFQxkXKn0OetCVJ2kJ/akf/ADyk/wDHf8aDq0KqWaOQKBknjj9a7fZSMedFyKRJolkQ5VhkGn1mWFV7i8S2ZVZWYsM/Lj+tNK7sJuyKs2phoZBEjrIVOxiFIB7cZrN8OXd9Z6PDDqqq90udzQABevbmtPZMj2iNOTWIYwu6KX5jgfd/x9qibxDaJIkbK4kfO1SVy2OuOahxaNYxcldBL4hs4U3yq6LnGWKgfzp41213KHWRAxA3NjH86VivZs06KRBnNHc2t7NNDEsscuCRuwVIGKf9qvP+fL/yIKAD7Vef8+X/AJEFYuv6lq8aRwxaUHtZVYTymUAxDsff6UMcd0SUVznrCN90/SsATfKOe1fM8Q/8u/n+h04dXuL53vR53vXzVzq5DR0vSLPVtOje8sI52jdtjMenNaFv4Y0u2kmeLS4086MxSYPDIeCPyzX6dhv4EPRfkfMVfjfqMbwrpLK4bSYiHyWBPUndn9Xf/vo1n6g13p8jadDpixaWqh0nEnRyxyu39fxrDMf90qeheHX72JR873o873r83ufSchFcTYgY56Y/nV/7f7193wmr0J+p8/nHu1I+gfb/AHqOe+zbyjPVD/KvreQ8fmOp0mWb+y7b9zxsH8VXPOnx/qP/AB6vMe52rYXzpv8Anh/499f/AK1cneapqT38yX1iLaONysEgfPmrxzjtWlFXmiKrtEh+3+9H2/3rv5Dk5iC6vsiPno/9DWJr8lncac0t3MbcwHzIrhThomHQj/DvXHXVpHs4OHNRuc7pOo3Ov6lENdPlmAB7e3xtWb/pp9fbtXaPd+YYkz1kX+dZI3VP3Gz0mioPOCigAqtfwNc2E0KEBnQgE0DTs7nOG117P/Hpa/8Afw0fZte/59LX/v4an2a7nT9afYQ2uvEEfZLX/v4aoDw7q2P9XH/31XlZllaxnL71rG9HHunfQP8AhHdW/wCecf8A31R/wjurf884/wDvqvM/1bX85v8A2p/dOn0CymsNOEM4AfcScGtWvp6cOSCh2SR5Mpc0m+4Vma7ZzX2mPFb4MuQQCcZrPE0fbUpU+5VKfJNS7HIf2Drn/PvF/wB/KP7B1z/n3i/7+V85/q2v5z1P7U/ujJfD2uSRMn2eIZH9+pf+Ea1j+7F/31X0OTYVZdTlC97s8vH1PrUlLawv/CNax/di/wC+qQ+GdYKkbYuRj71ey8Wuxw/V33O20+F7ewhif7yKAatVwPc6kFYPiTSrrUUha1K7kzkNV0p8klIiceaNjnv+Ea1j+7F/31R/wjWsf3Yv++q7Pra7HP8AV33I5fC+suF+WLg5+9VK88BalftCbhY2ETb1XfwT7iuarUU5XPSw1b2NPkauF34D1G8WPzEiDRMGjdWwVPtV2Dwpq63ETSeXtVwxw3pWd0byxl01bc9DoqDhCigAooAKKACigAooAKKACigAooAKKACigAooAKKACigAooAKKACigAooAKKACigAooAKKACigAooAKKACigAooAKKACigAooAKKACigD/9kA/+Ex6Gh0dHA6Ly9ucy5hZG9iZS5jb20veGFwLzEuMC8APD94cGFja2V0IGJlZ2luPSfvu78nIGlkPSdXNU0wTXBDZWhpSHpyZVN6TlRjemtjOWQnPz4NCjx4OnhtcG1ldGEgeG1sbnM6eD0iYWRvYmU6bnM6bWV0YS8iPjxyZGY6UkRGIHhtbG5zOnJkZj0iaHR0cDovL3d3dy53My5vcmcvMTk5OS8wMi8yMi1yZGYtc3ludGF4LW5zIyI+PHJkZjpEZXNjcmlwdGlvbiByZGY6YWJvdXQ9InV1aWQ6ZmFmNWJkZDUtYmEzZC0xMWRhLWFkMzEtZDMzZDc1MTgyZjFiIiB4bWxuczp4bXA9Imh0dHA6Ly9ucy5hZG9iZS5jb20veGFwLzEuMC8iPjx4bXA6Q3JlYXRlRGF0ZT4yMDIzLTAxLTEyVDExOjA4OjI5LjA0OTwveG1wOkNyZWF0ZURhdGU+PHhtcDpDcmVhdG9yVG9vbD5XaW5kb3dzIFBob3RvIEVkaXRvciAxMC4wLjEwMDExLjE2Mzg0PC94bXA6Q3JlYXRvclRvb2w+PC9yZGY6RGVzY3JpcHRpb24+PC9yZGY6UkRGPjwveDp4bXBtZXRhPg0KICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgIAogICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgCiAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAKICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgIAogICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgCiAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAKICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgIAogICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgCiAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAKICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgIAogICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgCiAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAKICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgIAogICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgCiAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAKICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgIAogICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgCiAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAKICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgIAogICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgCiAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAKICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgIAogICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgCiAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAKICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgIAogICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgCiAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAKICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgIAogICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgCiAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAKICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgIAogICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgCiAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAKICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgIAogICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgCiAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAKICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgIAogICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgCiAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAKICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgIAogICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgCiAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAKICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgIAogICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgCiAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAKICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgIAogICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgCiAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAKICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgIAogICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgCiAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAKICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgIAogICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgCiAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAKICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgIAogICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgCiAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAKICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgIAogICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgCiAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAKICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgIAogICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgCiAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAKICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgIAogICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgCiAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAKICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgIAogICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgCiAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAKICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgIAogICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgCiAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAKICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgIAogICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgCiAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAKICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgIAogICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgCiAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAKICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgIAogICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgCiAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAKICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgIAogICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgCiAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAKICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgIAogICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgCiAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAKICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgIAogICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgCiAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAKICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgIAogICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgCiAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAKICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgIAogICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgCiAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAKICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgIAogICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgCiAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAKICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgIAogICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgCiAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAKICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgIAogICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgCiAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAKICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgIAogICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgCiAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAKICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgIAogICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgCiAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAKICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgIAogICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgCiAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAKICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgIAogICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgCiAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAKICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgIAogICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgCiAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAKICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgIAogICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgPD94cGFja2V0IGVuZD0ndyc/Pv/bAEMAAwICAwICAwMDAwQDAwQFCAUFBAQFCgcHBggMCgwMCwoLCw0OEhANDhEOCwsQFhARExQVFRUMDxcYFhQYEhQVFP/bAEMBAwQEBQQFCQUFCRQNCw0UFBQUFBQUFBQUFBQUFBQUFBQUFBQUFBQUFBQUFBQUFBQUFBQUFBQUFBQUFBQUFBQUFP/AABEIAUAEGgMBIgACEQEDEQH/xAAfAAABBQEBAQEBAQAAAAAAAAAAAQIDBAUGBwgJCgv/xAC1EAACAQMDAgQDBQUEBAAAAX0BAgMABBEFEiExQQYTUWEHInEUMoGRoQgjQrHBFVLR8CQzYnKCCQoWFxgZGiUmJygpKjQ1Njc4OTpDREVGR0hJSlNUVVZXWFlaY2RlZmdoaWpzdHV2d3h5eoOEhYaHiImKkpOUlZaXmJmaoqOkpaanqKmqsrO0tba3uLm6wsPExcbHyMnK0tPU1dbX2Nna4eLj5OXm5+jp6vHy8/T19vf4+fr/xAAfAQADAQEBAQEBAQEBAAAAAAAAAQIDBAUGBwgJCgv/xAC1EQACAQIEBAMEBwUEBAABAncAAQIDEQQFITEGEkFRB2FxEyIygQgUQpGhscEJIzNS8BVictEKFiQ04SXxFxgZGiYnKCkqNTY3ODk6Q0RFRkdISUpTVFVWV1hZWmNkZWZnaGlqc3R1dnd4eXqCg4SFhoeIiYqSk5SVlpeYmZqio6Slpqeoqaqys7S1tre4ubrCw8TFxsfIycrS09TV1tfY2dri4+Tl5ufo6ery8/T19vf4+fr/2gAMAwEAAhEDEQA/AP1TooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAoopM0ALRSUZ9qQC0UmfajPtTAWikz7UZ9qAFopM+1GfagBaKTPtRn2oAWikz7UZ9qAFopKM0ALRSZpaACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACkopKQHiNp+2h8Ir63juLfxDqNxBINySxeHdTZWHqCLbmp/+GxvhT/0G9V/8JrVP/kavKv2Zh/xYXwT/ANg9f/QjXptfKVM7lTm4cm3mfS08nVSClz7+RY/4bG+FP/Qb1X/wmtU/+RqP+GxvhT/0G9V/8JrVP/kaq9FZ/wBvS/59/iaf2Kv5/wACx/w2N8Kf+g3qv/hNap/8jUf8NjfCn/oN6r/4TWqf/I1V6KP7el/z7/EP7FX8/wCBY/4bG+FP/Qb1X/wmtU/+RqP+GxvhT/0G9V/8JrVP/kaq9FH9vS/59/iH9ir+f8Cx/wANjfCn/oN6r/4TWqf/ACNR/wANjfCn/oN6r/4TWqf/ACNVej8KX9vS/wCff4h/Yq/n/Asf8NjfCn/oN6r/AOE1qn/yNR/w2N8Kf+g3qv8A4TWqf/I1V6Kf9vS/59/iH9ir+f8AAsf8NjfCn/oN6r/4TWqf/I1J/wANjfCn/oN6r/4TWqf/ACNUFL+FL+3pf8+/xD+xY/z/AIGjon7WHww8Qa5pukWeuX32/UZ1trVLjQ7+3WSVvurvkgVQTjuRXraTJJ905r4+/aD1abQtI8I6hbnE9v4lspFJHcCTrXvfwp8WXHiTQ7S4um3TyJlivA619JgcQ8ZQ9ta2p4WMw6wtX2d7npVFNQ5UU6u04wooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACk/wpaT/AApAfIX7M3/JBPBH/YPX+Zr06vMf2Zv+SCeCP+wev8zXp1fluI/jT9Wfo+H/AIMfQKTjrS1x/wAYPE194L+F3ijXtNaNb/T7CW4gMiblDqCRkdxWMY8zSRvKSim2df16UtfLX7Kf7VT+OvhVd+JPiX4g0fSZ11BrWGaTbaxsoUHHJ5PJr6V0LxBpvifTYtR0i+t9RsJhmO4tpA6N9CK2q0J0ZOMlt9xjSrQqxvFmhRXFap8avAei6wNKv/F+j2mpltotJrxFkJ9NpOc11xvYFtDdGZPswXeZc/Lt9c+lZOMo7o1U4y2ZPSVxVj8bPAOqawNKtPGGjXOpFtgtI7xDJn025zXUazrmn+HdOlv9UvIbCyi/1lxcOERfqT0o9nJOzWolOMldMvUlfMH7Un7VkXgv4SHxH8NPEei6xfR3yW8rxMl0igqxIIB4PAr2r4K+K7/x18KPC/iDVGjfUdQskuJ2iTYpc9cDtW0sPUp01UkYwrQnUdOJ21FFFcp0Hjn7T/8AyKPhv/sYrP8A9Bkr1/4DsR4bsf8Ac/rXj/7T3/Io+G/+xis//QZK9f8AgT/yLdj/ALn9a/Rcj/3L5s+Izj/efkj3GD/VLUlRwf6pfpUlev1PICiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiikzQAtFJuozQAtFJmjNAC0UlG6gBaKyfE3ivR/Bmi3Gr67qVrpOl2+PNvLuURxpk4GWPA5rBn+NHgS18Gx+LZfFukR+GJH8tdXa7T7Mzc/KHzjPB/KgDtKKztD8Qad4m0m21TSb2HUdOuk3wXVs4eORc4yrDgjNaGaAFornPGvxE8MfDiwhvvFOvaf4fs5pPKjn1C4WFHf+6C3U1r6Xq9nrWn299YXEd3Z3EYlhuIW3JIpGQQR1FAFyikpaACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACk/wpaT/CkB8hfszf8kE8Ef8AYPX+Zr06vMf2Zv8Akgngj/sHr/M16dX5biP40/Vn6Ph/4MfQK85/aM/5IT46/wCwTP8A+gmvRq474w+Gb7xp8LfFOhaYqPqGoWEtvAsj7VLsMDJ7CopNRqRb7mlVNwkl2Pz5/Y3/AGafDfxh+CPi3V/FYl1A20s0dhAJWVLciMMHwpGTnPWpP2VfiHrngD9nX43WlpfTSyeHUcWMjNu8pisnzDP+6Pavp39j/wCBfiv4NfBnxB4c8S29vBqd5PNJEsFwJVIaPaMsOnNch+zP+yX4j8I6F8WND8dWttBp3ixvLhNrcrKTGfMBPH3ThhX0M8VTk6inK6umv+AeDDDziqbgrOzufDHw/wDFHw+1D4fau/ivwT4s8S+M75mkg16zyyW7k5G35xnn1B6mvqH4R+PotW/Ya1Gx+I/iHVvDMNldG2W9hQi6kiAASJd45zyMn161d8K/s9ftN/BfT7/wh4D1jR5fCdxMTDeXEqrPboT1QF8g16P8Yf2TPGXxK/Zrs/COoeK5vEPjK3uBdG+1CUkSNwTHkngZHHJxmujEYihOUY8ys2nvdpenQwo0a0U2k7pdt/8AM+CPiZ4m+GNj4O0hPAnhfxVpXiO1uw7eJdSO0XS8kAkPjPToB0r6B/a28fa54y+DHwJ0Oe/mgXxEsZ1CZWwZS3lgZPtk10+p/sj/AB2+KHwnPhbxjquk6XbaPEDpOnaeRtmlBwDI4bH3S3atjxV+x78S/iP+zj4U0HW5tNsvG3hKYrpf2aQbGhAQKGcH73ynmtJYihzQbkrp977kRo1rSSi7Ndjzz9sj9ljwj8H/AIE+Htc8MGWwu3mgivF85mS63RlixViQGyO3avuP9mT/AJN/8Cf9gyL+tfHfxE/Zh/aU+Mnw5stH8XahpEh0pkWz063uFUShRgO8m4gkDjp3r7c+CfhXUPA3wn8LaBqqJHqOn2SQTpG4dQ464I615WMqJ0FBzUmmz0sJC1dyUOVNHbUUUV4J7J43+09/yKPhv/sYrP8A9Bkr1/4E/wDIt2P+5/WvIP2nv+RR8N/9jFZ/+gyV6/8AAn/kW7H/AHP61+i5H/uXzZ8RnH+9fJHuMH+qX6VJUcH+qX6VJXr9TyAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAr4/8A20P23tW/ZX+IngvRLXQLDVtM1qIyXM1w0gmjO5lAjCkAk7R19a+wK/MP/gqTax3v7SXwUilXfGwXIPtNIaAOm1b/AIKa+PfA/wAQtCg8a/Cn/hG/BGtXCx2WoXEhNyYyQN7ASFRgMOK9s/ap/bm039n/AEXw7Fo2knxP4o8Rosmm6cpwGRgcMxyOM8cc818wf8FU98vgDwtK4yyahhT3GWSvG/24tJ1az+I3wX1k61ceHLNtEt4Y9cjVj9jfaOcgjnHuOta8isK59gfC3/goJ43g+KmkeC/jR8OY/h9NrR2adcwyF0dyflVjvbkkgcevOK7j4C/tla38SPi18RPA/ijRNN0a48NCSe2ezeQmeBcbWfcepBHT1r8/tU8Ep4q+KngWDxD+0Zc/ELUbXUIrnT7eO0e9EbB1JBYSnZnAyT6V0/7T3iq4/Zz/AGkda8TQtIk/irw59lVlGN7qYk/DO2jk0Fc+of2X/wDgoxqvx5+Pl94G1Lw7p2laM6zNpuoW7SGScRqzHO47ei9vWvQvgR+15rvxn+P/AI38Hx6Jp8HhPw7uhGqQvIZ3nBA2sCduPvdB2r4S8T6DJ+zX8Lfgj8RLWE2+o28cseqSIMFfNQKM/i1e+f8ABNTSyngPxd4xwTL4q1y4uhIepCyyL/WnyBc9/wD+CiFwJv2RfGpBz8sP/o1a+O9N8cab4P8A+CZuhzax4XsfF1iuoEHTNRlljiJJk+bMbK2R9e9fWf7fGf8Ahj3xnuOSUh/9GrXw74y5/wCCY2jj/p+H/tSpjvYD6P8AGf7Yl9+zr+yv8L/E/hzwrpMVrqWIG0uSWYw2sW4n5Du3HHJ+Ymuc8bf8FPPiT4U0bTPFsPwjB+H0/lo+sXLkNIxHJjAk6HnBYCvF/wBqBRcfsRfByNxlPOUY+rHNex/tSQgfsD2cCKFhh0qz2KOi4hIGKvlQXY//AIKW/EjTvit+yx8O/EumH/Q9S1NJ0BPT5SCPzBre/wCG0tU+H3hb4WfDn4d6NaeL/G97ptoktvcM3kWse1RucqQe/v0Oa+Zvjqst1/wT0+D0ZcqTMdp9MySDNY/gPRdT/Yv+J3w88d3V42q+G/E1hFFeXUqkGAuFDkkngDccUcoj9mvC2q395odi+rJbxam0StcR2hYxq5HIUnnA9631bcoNeffD/VItd0+2v4JRNDPGsqOOhVhkH8q9AjG1QKyasykOoooqRhRRRQAUUUUAFFFFABRRRQAUUUUAFFFFABRRRQAUUUUAFFFFABRRRQAUUUUAFFFFABRRRQAUUUUAFFFFABRRRQAUUUUAFFFFABRRRQAUUUUAFFFFABRRRQAUUUUAFJ/hS0lID5C/Zm/5IJ4I/wCwev8AM16dXmP7M/8AyQTwT/2D1/ma9Or8uxCftp+rP0fDtexj6BSfypaK57M30E+lFLRRysWglLRSfzpWfYegtFFFHKwEo96WijlfYAoooosxaHjf7T3/ACKPhv8A7GKz/wDQZK9f+BP/ACLdj/uf1ryH9p7/AJFHw57eIrP/ANBkr174E/8AIt2P+5/Wv0XI/wDcn6s+Izf/AHr5I9xg/wBUv0qSo4P9Uv0qSvWPJCiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKSgBaKQsBRuHrQAtFJuHrRuHrQAtFJuHrRuHrQAtFJuHrRuHrQAtFJuHrRuHrQAtFJuHrRuHrQAtFJuHrRuHrQAtFJuHrRuHrQAtFJuHrRuHrQAtFJuHrRuHrQAtFJuHrRuHrQAtFJuHrRuHrQAteSfGL9nPwJ8YPEGjeIPE+i/wBo6vov/HhcefInlck9FIB5J6g161uHrTXVX64oA8H+JP7OfhD42abDp/jPR/7WtLaXzYo/OePa2Qc5Ug9hUni74C+GPHvh1PD2u6LDqGkQxrFFDMCSiqoVcN1HA65zXt6wovYflQLdPQVpzE2Plf4d/sdfDz4Paw194W8LwWN03SZ3eVl+m8nH4V03xO/ZT8B/G7+zrrxjoA1a500f6K5meMx87sfKRnn1r6Ba1jY5wPyp3koFxijnY7Hz/wCNv2dPC/xU8K/8Iv4o0gX+hIUK2wkaPG0grgqQR0Hetf4e/B3SPhlpdnoPh2wGn6Pak+VAGLYycnknJ5r2gW8YOcCgW8e7OBT52FjkvHXwz0H4oeB7zwx4msf7Q0a8VRNb+YybtpyOVIPUV5LrP7KPgK++HcXw9fQd3g+GXzUsPtEnDc87927ue9fRm1duOMVE0CN2FQmB85eJv2U/AvjHwdo3hLVtC+1eH9IYNZ2vnyL5ZBz94MCfxNdBr3wD8NeNfBP/AAhmt6X9r8NiJIfsnmuvyqu0DcCD0969s+zp6D8qVYEBzgVfMFj5y8Rfsl+AfEXgXRvBF5oHneGNHbdZWf2iRfKOSfvBtx5J6mp/GX7Mfg/4h+DLXwhr2ire+H7RUWC2MjqUCDC4cEN+tfQ3kJnOBR9nT2/KjnCx5z8OPA9r8PdIstE0uF4dNtE8uCOSRpCq56bmJJ/GvSocmMZqNbdFI6VMML3qJO4DqKTcPWjcPWpGLRSbh60bh60ALRSbh60bh60ALRSbh60bh60ALRSbh60bh60ALRSbh60bh60ALRSbh60bh60ALRSbh60bh60ALRSbh60bh60ALRSbh60bh60ALRSbh60bh60ALRSbh60bh60ALRSbh60m4etADqKTdS0AFFFFABRRRQAUUUUAFFFFABRRRQAUUUUAFFFFABRRRQAUUUUAFFFFACUnanUm2kB8XfEb9kvSfDc0Nr4Tt9et9PjiAWO31e52IcngDfgfhXnbfs9awrEZ8R/+Da4/+Lr9DLmzSYcjNZzeH4GzlM/hWkVT6xX3D5p9z4D/AOGe9X9fEf8A4Nrj/wCLo/4Z71f18R/+Da4/+Lr77/4R6D/nn+lH/CPQf88/0qv3X8i+4XNP+ZnwJ/wz3q/r4j/8G1x/8XR/wz3q/r4j/wDBtcf/ABdfff8Awj0H/PP9KP8AhHoP+ef6UWpfyL7g5p/zM+BP+Ge9X9fEf/g2uP8A4uvONA+FWu3nxw8ZaA914jkgsNM0+4S3/ta4+QyGXJ+/32j8q/UL/hHoP+ef6V85eFdHi/4bI+LUewBV8PaGRx6m5pctJ/YX3BzT/mZ4en7P+rOOD4j/APBtcf8AxdO/4Z71f18R/wDg2uP/AIuvuzTdBgaMZjH5Ve/4R+D/AJ51VqX8i+4Oaf8AMz4F/wCGe9X9fEf/AINrj/4uj/hnvV/XxH/4Nrj/AOLr77/4R6D/AJ5/pR/wj0H/ADz/AEpWpfyL7g5p/wAzPgT/AIZ71f18R/8Ag2uP/i6P+Ge9X9fEf/g2uP8A4uvvv/hHoP8Ann+lH/CPQf8APP8ASi1L+RBzT/mPgaH9nnUlvLaeW21q7a3lWaNbq/mmQOOh2sxHGTX1j8HdDudJ0GyhuY2ilVMMjdRzXpH/AAj8H/PP9KsW+lrbt8oxT5klyxVhat3ZoQf6pakpqDaoFOrEYUUUUAFFFFABRRRQAUUUUAFFFFABRRRQAUUUUAFFFFABRRRQAUUUUAFFFFABXjPx2+MWseEdQ0jwj4M09NW8Z60cwrIcRWkQzmaU4OF+VgODzgd817NXzV8JLr/hJ/2gvjJrc58yfRL+Pw9Azc7IvJhuMD/gUh/OmgGTfszyeKh9o8feONe16+l+aSPTruTTo4yf4V8h1yB61WP7F/w6zxdeMj/3Nuo//Hq+gbGzNw29+ea1Y7CNR0qtAPmj/hi74df8/XjH/wAK7Uf/AI9R/wAMXfDr/n68Y/8AhXaj/wDHq+mvsaeg/Kj7GnoPyo0A+Zf+GLvh1/z9eMf/AArtR/8Aj1H/AAxd8Ov+frxj/wCFdqP/AMer6a+xp6D8qPsaeg/KjQD5l/4Yu+HX/P14x/8ACu1H/wCPUf8ADF3w6/5+vGP/AIV2o/8Ax6vpr7GnoPyo+xp6D8qNAPmX/hi74df8/XjH/wAK7Uf/AI9R/wAMXfDr/n68Y/8AhXaj/wDHq+mvsaeg/Kj7GnoPyo0A+Zf+GLvh1/z9eMf/AArtR/8Aj1H/AAxd8Ov+frxj/wCFdqP/AMer6a+xp6D8qPsaeg/KjQD5l/4Yu+HX/P14x/8ACu1H/wCPUf8ADF3w6/5+vGP/AIV2o/8Ax6vpr7GnoPyo+xp6D8qNAPmX/hi74df8/XjH/wAK7Uf/AI9R/wAMXfDr/n68Y/8AhXaj/wDHq+mvsaeg/Kj7GnoPyo0A+Zf+GLvh1/z9eMf/AArtR/8Aj1H/AAxd8Ov+frxj/wCFdqP/AMer6a+xp6D8qPsaeg/KjQD5l/4Yu+HX/P14x/8ACu1H/wCPUf8ADF3w6/5+vGP/AIV2o/8Ax6vpr7GnoPyo+xp6D8qNAPmX/hi74df8/XjH/wAK7Uf/AI9R/wAMXfDr/n68Y/8AhXaj/wDHq+mvsaeg/Kj7GnoPyo0A+Zf+GLvh1/z9eMf/AArtR/8Aj1H/AAxd8Ov+frxj/wCFdqP/AMer6a+xp6D8qPsaeg/KjQD5l/4Yu+HX/P14x/8ACu1H/wCPUf8ADF3w6/5+vGP/AIV2o/8Ax6vpr7GnoPyo+xp6D8qNAPmX/hi74df8/XjH/wAK7Uf/AI9R/wAMXfDr/n68Y/8AhXaj/wDHq+mvsaeg/Kj7GnoPyo0A+Zf+GLvh1/z9eMf/AArtR/8Aj1H/AAxd8Ov+frxj/wCFdqP/AMer6a+xp6D8qPsaeg/KjQD5l/4Yu+HX/P14x/8ACu1H/wCPUf8ADF3w6/5+vGP/AIV2o/8Ax6vpr7GnoPyo+xp6D8qNAPmX/hi74df8/XjH/wAK7Uf/AI9R/wAMXfDr/n68Y/8AhXaj/wDHq+mvsaeg/Kj7GnoPyo0A+Zf+GLvh1/z9eMf/AArtR/8Aj1H/AAxd8Ov+frxj/wCFdqP/AMer6a+xp6D8qPsaeg/KjQD5l/4Yu+HX/P14x/8ACu1H/wCPUf8ADF3w6/5+vGP/AIV2o/8Ax6vpr7GnoPyo+xp6D8qNAPmX/hi74df8/XjH/wAK7Uf/AI9R/wAMXfDr/n68Y/8AhXaj/wDHq+mvsaeg/Kj7GnoPyo0A+Zf+GLvh1/z9eMf/AArtR/8Aj1H/AAxd8Ov+frxj/wCFdqP/AMer6a+xp6D8qPsaeg/KjQD5l/4Yu+HX/P14x/8ACu1H/wCPUf8ADF3w6/5+vGP/AIV2o/8Ax6vpr7GnoPyo+xp6D8qNAPmX/hi74df8/XjH/wAK7Uf/AI9R/wAMXfDr/n68Y/8AhXaj/wDHq+mvsaeg/Kj7GnoPyo0A+Zf+GLvh1/z9eMf/AArtR/8Aj1H/AAxd8Ov+frxj/wCFdqP/AMer6a+xp6D8qPsaeg/KjQD5l/4Yu+HX/P14x/8ACu1H/wCPUf8ADF3w6/5+vGP/AIV2o/8Ax6vpr7GnoPyo+xp6D8qNAPmX/hi74df8/XjH/wAK7Uf/AI9R/wAMXfDr/n68Y/8AhXaj/wDHq+mvsaeg/Kj7GnoPyo0A+Zf+GLvh1/z9eMf/AArtR/8Aj1H/AAxd8Ov+frxj/wCFdqP/AMer6a+xp6D8qPsaeg/KjQD5l/4Yu+HX/P14x/8ACu1H/wCPUf8ADF3w6/5+vGP/AIV2o/8Ax6vpr7GnoPyo+xp6D8qNAPmX/hi74df8/XjH/wAK7Uf/AI9R/wAMXfDr/n68Y/8AhXaj/wDHq+mvsaeg/Kj7GnoPyo0A+Zf+GLvh1/z9eMf/AArtR/8Aj1H/AAxd8Ov+frxj/wCFdqP/AMer6a+xp6D8qPsaeg/KjQD5l/4Yu+HX/P14x/8ACu1H/wCPUf8ADF3w6/5+vGP/AIV2o/8Ax6vpr7GnoPyo+xp6D8qNAPmX/hi74df8/XjH/wAK7Uf/AI9R/wAMXfDr/n68Y/8AhXaj/wDHq+mvsaeg/Kj7GnoPyo0A+Zf+GLvh1/z9eMf/AArtR/8Aj1PT9j7wdp+ZNJ1bxbYXY+5NL4kvbgKf915SK+l/saeg/Kj7Enp+lF0I+YI/HXjj9nfxFpdn4q1F/FngK9k8j+2pIViuLBzyvmhRgpgY3Ek5I45r6bsr6O6jR0bcrDII7iuL+MXgW08afDXxDo9zH5kNxaNuH+78381rkP2XfFlz4u+BvgnWrlzJNfaekrMe5yR/SjR7DPb6KZE25BT6gAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAK+afC/wDyed8Xv+xd0L+d1X0tXzT4X/5PO+Lv/Yu6F/O6qo7gfQekj90taVZ2k/6la0qT3AKKKKQBRRRQAUUUUAFFFFABRRRQAUUUUAFFFFABRRRQAUUUUAFFFFABRRRQAUUUUAFFFFABRRRQAUUUUAFfLv7PPHxZ/aKA6f8ACaR/+m61r6ir5d/Z6/5K1+0V/wBjpH/6brWmtwPpPTF/dir9UtN/1K/SrtD3AKKKKQBRRRQAUUUUAFFFFABRRRQAUUUUAFFFFABRRRQAUUUUAFFFFABRRRQAUUUUAFFFFABRRRQAUUUUAFFFFABRRRQAUUUUAFFFFABRRRQAUUUUAFFFFABRRRQAUUUUAFFFFABRRRQAUUUUAFFFFABRRRQAUUUUAFFFFABRRRQBmeJv+Rd1T/r1l/8AQDXgf7GPy/su/DD/ALBKf+htXvnib/kXNU/69Zf/AEA14F+xn/ya78MP+wSn/obVUdwPoq3/ANWKmqG3/wBXU1SAUUUUAFFFFABRRRQAUUUUAFFFFABRRRQAUUUUAFFFFABRRRQAUUUUAFFFFABRRRQAUUUUAFfNPhf/AJPO+Lv/AGLuhfzuq+lq+afC/wDyed8Xf+xd0L+d1TjuB9CaT/qVrSrN0n/UrWlTluAUUUVIBRRRQAUUUUAFFFFABRRRQAUUUUAFFFFABRRRQAUUUUAFFFFABRRRQAUUUUAFFFFABRRRQAUUUUAFfLv7PX/JWv2iv+x0j/8ATda19RV8u/s9f8la/aK/7HSP/wBN1rQtwPpXTf8AVL9Ku1S03/VL9Ku03uAUUUUgCiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigDM8Tf8AIuap/wBesv8A6Aa8C/Yz/wCTXfhh/wBglP8A0Nq998Tf8i5qn/XrL/6Aa8C/Yz/5Nd+GH/YJT/0NqqO4H0Vb/wCrqaobf/V1NUgFFFFABRRRQAUUUUAFFFFABRRRQAUUUUAFFFFABRRRQAUUUUAFFFFABRRRQAUUUUAFFFFABXzT4X/5PO+Lv/Yu6F/O6r6Wr5p8L/8AJ53xd/7F3Qv53VOO4H0JpP8AqVrSrN0n/UrWlTluAUUUVIBRRRQAUUUUAFFFFABRRRQAUUUUAFFFFABRRRQAUUUUAFFFFABRRRQAUUlFAC0UUUAFFFFABRRRQAV8u/s9f8la/aK/7HSP/wBN1rX1FXy7+z1/yVr9or/sdI//AE3WtC3A+ldN/wBUv0q7VLTf9Uv0q7Te4BRRRSAKKKTdQAtFJmloAKKKKACiiigAooooAKKKKACiiigAopKM0ALRSUtABRRRQAUUUUAFFFFABRRRQAUUUUAFFFJQAtFJS0AFFFFABRRRQAUUUUAFFFFABRRSZ9KAFopAaWgAooooAKKKKACiiigAooooAKKKKACikooAWikzS0AFFFFABRRSc0AZvib/AJFzVP8Ar1l/9ANeBfsZ/wDJrvww/wCwSn/obV6J49+OHgXQNYvfBupeJLSw8Sz2MssVhcbkZ12dQxG3uO/evBf2T/jV4I8N/Aj4Q+GdQ8R2kOv32mIlvYJukkchmyMKDj8cVUdwPsC3/wBXU1V7WQNGMVYqQCiiigAoopKAFopOaTdQA6im7qN1ADqKbuo3UAOopu6jdQA6im7qN1ADqKbupc0ALRSUtABRRRQAUUUUAFFFFABRRRQAV80+F/8Ak874u/8AYu6F/O6r6Wr5p8L/APJ53xd/7F3Qv53VOO4H0JpP+pWtKs3Sf9StaVOW4BRRRUgFFFFABRRRQAUUUUAFFFFABRRVHXLmSz0W/niO2WK3kdG9CFJBoAuZzRj2r4/+GNx8TvH/AMP9B8R3HxX1O0m1K1W4aCLTbQrGTngEpn866f8A4Rj4k/8ARYNX/wDBXZ//ABFZe2gtLnoxy/ESSkon0zj2ox7V8zf8Ix8Sf+iwav8A+Cuz/wDiKP8AhGPiT/0WDV//AAV2f/xFL20O4/7NxH8v4n0zj2ox7V8zf8Ix8Sf+iwav/wCCuz/+Io/4Rj4k/wDRYNX/APBXZ/8AxFHtodw/s3Efy/ifTOPajHtXzN/wjHxJ/wCiwav/AOCuz/8AiKP+EY+JP/RYNX/8Fdn/APEUe2h3D+zcR/L+J9M49qMe1fM3/CMfEn/osGr/APgrs/8A4ij/AIRj4k/9Fg1f/wAFdn/8RR7aHcP7NxH8v4n0z+FLur5l/wCEY+JP/RYNX/8ABXZ//EUf8Ix8SP8Aor+r/wDgrs//AIij20O4f2dif5fxPpjIp2a+Xb61+Ivhm1fU5filqmox2/ztay6daqsnIGCQmR17Vtr4t8WMoP8Awks3I/594v8A4mvjM64uwOR1o0cRGTclfQ7aOS4msr3SPofIoyK+ef8AhLPFn/QzTf8AgPF/8TR/wlniz/oZpv8AwHi/+Jr5z/iJWVfyT+5f5nT/AKv4n+Zfj/kfQ2RRkV88/wDCWeLP+hmm/wDAeL/4mj/hLPFn/QzTf+A8X/xNH/ESsp/kn9y/zD/V/E/zL8f8j6GyK+Xv2e/+StftE/8AY6R/+m61rb/4SzxZ/wBDNN/4Dxf/ABNcT4M8Jar4J8QeMtXtPE91JceKNSXVLsPbxYWUQpEAOOm2MVcfEnKbNuE/uX+Yv9X8T3X9fI+r9N/1K1dr52j8TeKYuE8STAf9e8X/AMTT/wDhLPFn/QzT/wDgPF/hU/8AESsp/kn9y/zH/q/if5l+P+R9DUZFfPP/AAlniz/oZp//AAHi/wDiaP8AhLPFn/QzTf8AgPF/8TS/4iVlP8k/uX+Yf6v4n+Zfj/kfQ2aTOeO9fPX/AAlniz/oZpv/AAHi/wAKIPiNr+h6xpcuo+IHm097gJOskMajbg9wK6cN4hZZiq0KEISvJpLRdfmZ1MixFODm5LT1PoWlzWLofiK31q1jubeUSwuNwcdCK+L/AIwf8FF9dt/ivf8Aw/8Ag34B/wCFi6xpu4XbNIUjVlIDAEMPuk4PuO9fqlmtD5q591VzOj/E7wl4g8QXWhaZ4k0vUNZtd3n2Ftdo88WDg7kByMGvl79l79v5vjF8QL74feOPDJ8E+N7YEixLFkkwCWVSSTkAZ/rXzp+yRqH9n/8ABQD4sXQPy7r0Z+stNK7sM/VHcD0rm9L+JnhPXPEdz4f0/wASaXfa5a7vP063u0eeLBwdyA5GPpXwnef8FLvGfib4naxoHwv+GR8baNo7f6derKVkCA/MyjcF456+leQfsR/ENfF/7c3j3xMLWXTjeRXTvaTjDwszZKHkjI+tHKK5+tstxHBDJNI6xxRqXd2OAoAySfauY0P4r+DfE1jqF7pHijSdTtNPUteT2t5HIluBnJcg/L0PX0qtrmo/bvAuvPnrp9x/6LavzF/YI1yHQ/hD8eprmxTUbXypDJZyuypMoEuUJU5APTI5o5dbBc/U/wAJ+PvDfjyzlu/DeuWGu2sT+W82n3CzKrehKk4Nb26vzf8A2Uv2gNP8D/sofEPxr4T8Dab4ebSboyjS4ryeWK4fIG5mdiwPPaodM/4KgfETxR8L4/E3hX4TJrktojS6vPHK4s7VQAcK24MSPmzx6UcrC5+jWsa1YeH9LuNR1O8gsLC3XfNdXEgSONfVmPAFU/C/jLQvG2lrqfh7V7LW9OZii3dhOs0RYAEjcpIzyK+N9e/ak039qD9hP4g+Ira1/s6/j09obyyJ/wBVJvXgc9MCvDf2Wf2q9P8A2Z/2J9KuBCNX8QX2pzRabpKH555GWIDOMEAEjPtRyhc/VHORRXlnwE8eeMPG3w907WPHOjWHh/WLxRKLCxkdwiHldxYdcY/OvUUbzFBFS0Aop2RXifjrxd4ig8banY2OsyWNpbpCUjSJG5ZMnkisX/hLPFn/AEM03/gPF/8AE1+Y47j/AC3AYmeFqwk5Qdnov8z6KjkeIr041IyVn6/5H0NkUZFfPP8Awlniz/oZpv8AwHi/+Jo/4SzxZ/0M03/gPF/8TXD/AMRKyn+Sf3L/ADN/9X8T/Mvx/wAj6GyKMivnn/hLPFn/AEM03/gPF/8AE0f8JZ4s/wChmm/8B4v/AImj/iJWU/yT+5f5h/q/if5l+P8AkfQ2RRkV88/8JZ4s/wChmm/8B4v/AImj/hLPFn/QzTf+A8X/AMTR/wARKyn+Sf3L/MP9X8T/ADL8f8j6GyKMivnn/hLPFn/QzTf+A8X/AMTR/wAJZ4s/6Gab/wAB4v8A4mj/AIiVlP8AJP7l/mH+r+J/mX4/5H0NkUZFfPP/AAlniz/oZpv/AAHi/wDiaP8AhLPFn/QzTf8AgPF/8TR/xErKf5J/cv8AMP8AV/E/zL8f8j6GJpPxr56/4SzxZ/0M03/gPF/8TTPCvizxD4m8yN/G89tfK8im2jtYj8qtjPK96+m4f4rwXEWKeEw0ZKSV9dvzZ5mPy2tl9L21RpryPogY+tOrxj7N4q/6He9/8BIf/iaPs/ir/od73/wEh/8Aia/R/qMv5l+J899ZXY9norxj7P4q/wCh3vf/AAEh/wDiaPs/ir/od73/AMBIf/iaf1KX8yD6yuzPZ6K8Y+z+Kv8Aod73/wABIf8A4mj7P4q/6He9/wDASH/4ml9Sl/MvxD6yuzPZ6K8Y+z+Kv+h3vf8AwEh/+Jo+z+Kv+h3vf/ASH/4mn9Sl/Mg+srsz2eivGPs/ir/od73/AMBIf/iaPs/ir/od73/wEh/+JpfUpfzL8Q+srsz2ak/CvGvs/ir/AKHe9/8AASH/AOJqrqjeLLHTLu5TxreO0MLyBTaQ4JCk4+77UfUZfzL8RfWl2Z7f1p1cB8OfiFb+KtBsH+1rdXggjNwV4+cqM5H1zXextvUGuCUHBtM64y5lcdRRRUlBRRRQAUUUUAFFFFABRRRQAU3vTqSkAgPpTq83+L2tatpt14Zs9K1J9LN/ePDLNHGrttETMAAwI6isD7P4q/6He9/8BIf/AImu6nhZVIKd1qcssQoycbHs9FeMfZ/FX/Q73v8A4CQ//E0fZ/FX/Q73v/gJD/8AE1r9Sl/MifrK7M9mpMe9eNfZ/FX/AEO97/4CQ/8AxNH2bxV/0O97/wCAkH/xNL6jL+ZfiH1ldmfPX/BVj9nc/FD4Pjxnpdvv1/wwpkZ0GXa2Od6KOpO4r+Ar5f8A+CWfwbksL+8+JWqQFmXdZaZ5i4KHGJGGeoKuBX6M6roOv63p89lfeMbq5tZ0MckUlnAQynqPu1yfg/4Gw+A/D9jomh6/c2Om2SbIIVt4jtHuSMn6mtI4Np6yX4k/WF2PfPDV3JdWyu3FdEOleI2uj+I7NAkPjS8RR0H2SH/4mp/s3ir/AKHi9/8AASD/AOJqXgnfSSGsSux7RRXjH2fxV/0O97/4CQ//ABNH2fxV/wBDve/+AkP/AMTU/UpfzL8R/WV2Z7NTa8a+zeKv+h3vf/ASD/4muV1TWvGkPjVdHj8b3aQnT/tZk+xwZ3eYVx93pionhZQXNzI6MPL6xP2aVj6QzS188eb40/6H2+/8Arf/AOJo8zxp/wBD9ff+AVv/APE1y+zPS+qS/mR9D0V88+Z4z/6H6+/8Arf/AOJpon8ZNuA8f3pK9QLO34+vy0ez8w+qS7o+iKK+ePM8af8AQ/X3/gFb/wDxNL5njP8A6H6+/wDAK3/+Jo9n5j+qS7o+hqK+ePN8af8AQ+33/gFb/wDxNL5njT/ofr7/AMArf/4mj2fmL6pLuj6Gor548/xkW2/8J/e7sZx9jt//AIml8zxn/wBD9ff+AVv/APE0ez8w+qS7o+hsUh9MV88+b40/6H2+/wDAK3/+Jo83xp/0P19/4BW//wATR7MPqkv5kfQ1Orwn4SfFC9bWPEGh+IdcOpXltfLFavLGsbshiVjwoA6k17MuoKyg5qXFo5KkHTk4s0KKKKkgKKKKACiiigAooooAK+afC3/J53xd/wCxd0P+d1X0tXzXpoGj/tkePWnO0614f0xLf/aMHnl8fTeKqO4H0FpP+pWtKszSWHlqK06HuAUUUVIBRRRQAUUUUAFFFFABRRRQAVmeJv8AkXNU/wCvWX/0A1pZrN8Tf8i5qn/XrL/6AaBHzV+zj/yQvwT/ANg2P+tekV5v+zj/AMkL8E/9g2P+tekV5MviZ+gUP4UfQKKKKk2CiiigAooooAKKKKACiiigRz/jz/kU9R/65/8AswrBWT5V6dK3vHn/ACKeo/8AXP8AqK5dZPlX6V/PfiN/v1H/AA/qe3gI80GXPM+lHmfSqm+jfX5Hc9PkLfmfSjzPpVTfRvouHIW/M+lHmfSqm+jfRcOQt+Z9KPM+lVN9G+i4chb8z6UeZ9Kqb6N9Fw5C35n0rlPiRKRocZHGJR0+hroN9ct8Q23aIv8A11H8jXtZI/8AhTw/+KP5nHjIWw9R+TPYvg7NLdeCoIY22yPalUbPQkHBr87/ANjfx1pHwM/aS+LWh+Np4tA1e+1K5nt7rUmEKtGZWP3mwPmDAiv0P+Bv/ItWH/XIVlfHj9kP4bfHS4S/8UeG4ry+VcfaIGMMjf7zJgnp3r+5paSPxZbHwDouvW3xe/4KPWWt+D3+2afpcMq3eoW43RZ2P824cEHIGc1a/ZoimuP2zPizHC+2aWO7Ab3L19y/C39nfwr8GtPfT/CWgw6YknDy43ysPQueSPxp3hP9lfwb4F8a6n4z0bSpLbxDqm77VcNcSOG3HJwpOBz6UxnwR/wTz+K/hv4S658SdD8U6jb6Dqv2yScm+kEO5Mj5Ruxk8ZwPXpTf2OvENh4w/bV+IGr6UNljdfapImxjzFLkhx9etfa3jj9hz4X+O/FA8Uat4Xjn1ncHaSOVo0Zh3ZFIU/iKt+Cf2VvBfw98b3Pivw9oP9nazcRmGSSOV/L2HsEztH4ChNCPYZs/8K/1zP8A0D7j/wBFtX5hfsY/L8E/jqO3lTf+gyV+rWkaIl1os1ldKWhniaKRc4yrAgj8jXj3h/8AZL8AfDXRfEGi+GdHltNO14MuoRtdSSeYDnOCzHHU9Kz+1cOh+f8A+zqx/wCGDfi4n96ds/8Afa163+xLbRwfsN3sUSALNHevJj+ImJc5r6U8O/sl+BvCPw/1bwVpWkSweHdWO68tjcyMXJOfvE5H4Gt7wH+z74a+G/gn/hDtB097Xw8yuptmmdzhxhvmYk9B61d0B+cf7MLOn7IHx/hVsR/bWXaPrXifgv4S6/4N+C/hf43aZeNqi6RrAM2mlSwSJWjII59zk+1frdoP7IfgDwR4N8QeFNE0aa20TxDJ5uowG7lcyse4Ytlfwqz4b/Zk8I+D/h9N4G0vSPL8NXAkWS0llaXO9drfMxJHHvxTugNT9n34oWHxf8EaP4k06dZILuFWYKc7HAwy/gcj8K91t12xivCvgv8AAHw58BNMfSfCNnPY6bLL5rQyXMkwDcn5dxOOp4Fe5WZPkjd1rKY1ueC+Pm2/EfXf9y3/APRdY/mfStL4jHb8SNb/ANy3/wDRdYW+v4p4mf8AwsYn/Ez9iy2N8HSfkW/M+lHmfSqm+jfXzNz0uQt+Z9KPM+lVN9G+i4chb8z6UeZ9Kqb6N9Fw5C35n0o8z6VU30b6LhyFvzPpR5n0qpvo30XDkLfmfSud+Gcnl+NLs9OJv/QxWvvrlvh/csvje7XPybZT/wCPCv2fwp1zxr+6z4zipcuBv5o9v+2e/wCtH2z3/WsX7UPWj7UPWv7A9mfj3MbX2z/ao+2f7X61i/ah60fah6mj2YcxtfbPf9aPtnv+tYv2oetH2oetHsw5ja+2f7VH2z/a/WsX7UPU0fah60ezDmNr7Z7/AK0fbPf9axftQ9aPtQ9aPZhzG19s/wBr9aoeILzOg6kM/wDLtJ/6AaqfaveqOuXP/El1Dn/l3k/9BNChqJyOU/ZvvHja4XJ6xj9DX1VYyb4V+lfJ37Of+uuf95P5Gvq7S/8AUr9K+cxPxs92l8KLtFFFcZsFFFFABRRRQAUUUUAFFFFABSetLTQeopMDyv42Ps1rwQfTUJf/AEQ9VPtnv+tS/HZtupeDG/6iEn/olqwPtQ9a+lwceahH5ni4iVqrNr7Z/tfrR9s9/wBaxftQ9TR9qHrXX7M5+Y2vtnv+tH2z3/WsX7UPWj7UPWj2YcxtfbP9r9aPtnv+tYv2oetH2oepo9mHMbX2z/a/Wj7Z7/rWL9qHrR9qHrR7MOY2vth/vfrR9s9/1rF+0j1o+1D1o9mHMbX2z3/WuD1e6/4ukrZ/5g2P/I5rovtQ9a4jVrj/AIuUpz/zCAP/ACMa5MTG1Js9rKPexSXkzrftdJ9qrI+1e9H2r3rxT7f2Jp3NzM9vIsDrHOVIRmXcA2OMjvzX5xXX7SPxS/Zd/aG1KH4iXM2seHNUuN5xkxiL+FojkgEDAxz3r9CvtVeXfHz4H6B8ePCU+marEq3iKTbXij54255zUSRz1sPKSvF6o9O8GePdI8f+G7PXdDvI73TrpA8ckbZxnsfQjPQ1t/avevyi+GPxO8a/sN/FCTwz4ljmvPCs8gVlJymwkjzI89PXt0r9K/B/jjSvHegWes6LeR3thdIHSSM55xyD6EdKIu+hNCSq6bPsdj9rrz/42fHbw98C/Btxr2u3C5UYgs1bEs74JAA69utZPxn+OWgfBLwfca5rdyquFIt7YEb5pMcAD0zivzs0vTfF/wC2t8SJfE3iiWSy8I20m1EUkIqg8RoPXA5PtSb6E1pez9yOsj1r9m/46fFf4wfG678a3Wpyad4MyYXtJkJiMfO1EXIBIJByK/Qy31KO6hSaNtyOu4H614H8NPhzZWWm2lnZWyWOj2qhY40GN2P5/WvZIJEtYUijAWNBtC+1OKNKGHajrubf2ul+19Kx/tXvR9q681R0exPONBumj+NupEHGb1T/AOQRX1RBeN5Mfzfwj+VfJnh9t/xmv29bxf8A0UK+qLf/AFEf+6P5VpNbHzGL0ryR6PRRRXEc4UUUUAFFFFABRRRQAV4Z+0V8N9e1K50Xxt4LWNvFvh9n2QSEAXdu+PNhySMFguAegzXudRzR+YpFMDwX4fftU+D9WX7Hrs03hTWIQFurXWImtkRxwdsjgKw9wcV6Kvxy+HO0Z8feGAffWbf/AOLp/if4c6B4mbdq2g6bqvH/AC/Wcc3/AKEDXMf8KH+Hy/8ANPPCp+ui23/xFU0mB03/AAvL4cf9D/4X/wDBzbf/ABdH/C8vhx/0P/hf/wAHNt/8XXNf8KM+H3/ROvCv/gjtv/iKP+FGfD7/AKJ14V/8Edt/8RS5QOl/4Xl8OP8Aof8Awv8A+Dm2/wDi6P8AheXw4/6H/wAL/wDg5tv/AIuua/4UZ8Pv+ideFf8AwR23/wARR/woz4ff9E68K/8Agjtv/iKOUDpf+F5fDj/of/C//g5tv/i6P+F5fDj/AKH/AML/APg5tv8A4uua/wCFGfD7/onXhX/wR23/AMRR/wAKM+H3/ROvCv8A4I7b/wCIo5QOl/4Xl8OP+h/8L/8Ag5tv/i6P+F5fDj/of/C//g5tv/i65r/hRnw+/wCideFf/BHbf/EUf8KM+H3/AETrwr/4I7b/AOIo5QOl/wCF5fDj/of/AAv/AODm2/8Ai6P+F5fDj/of/C//AIObb/4uua/4UZ8Pv+ideFf/AAR23/xFH/CjPh9/0Trwr/4JLb/43RygeGftuftcXnwe8N+HfGXw58X+F9btbG6aLVdIS/iuJLmOQqqlUSTOVwxzggZql8F/+Cm/w0+OGg3Wl6nM3hPxDLayKLW/cCJzsIyJMBRk9AT3rN/bW/ZTvPil4Z0Hwh8O/A/hfRob+6aXVNXi0+G2lto4yrKFdI85bLcZGcVwnwr/AOCfPgf4M2k1/ewHxRrkdu5+0XseYlO0nAjJKnB747VcYks+gf2cDn4FeCSOR/ZsfP516RXmv7N64+BHgfAwP7Mi4r0qvFn8TP0Ch/Cj6BRRRUG4UUUUAFFFFABRRRQAUUUUCOf8e/8AIp6l/wBc/wCorjFf5R9PWuz8ff8AIpal/wBc/wCorg1mO0cV/PHiR/v1H/D+p9HlceaEi1vo31W876Ued9K/Iro9r2ZZ30b6red9KPO+lF0PkLO+jfVbzvpR530oug5Czvo31W876Ued9KLoOQs76N9VvO+lHnfSi6DkLO+uS+Jmq2el+HhPfXcFlAJBmW4lWNRwe5OK6TzvpXKfEjTbLWvD4tr+zt723MgJiuIxIp4PY8V7mRtf2ph7/wAy/M8/Hwthanoz2T4H+JtHXwLZao2rWI01IhuvDcp5I+r5x+texaT4x8O+INIl1HTtd03UdOhH7y7tLuOWJPq6kgfia8R+CPhbR28D2WlHR7E6XJEA9kbVDA31TGD+Vev6P4J8PeH9Kl0vS9A0zTdMnH72zs7OOKF89dyKoB/EV/c9Tc/Co7FnQvFnhfxRHPNo2vaXq0duD5z2N5HMI8ddxVjj8adovjPwv4muZrXR/EGk6tcwHEsNjeRzPHj+8FYkfjVPQfA3h3wnDcRaJ4e0vR4rjImj0+yjgWTPUMFUZ/Gk0PwH4a8K3U13ovhzSdHupzmaewso4Hkz13Mqgn8ayLLen+NvCusapJpVj4i0i91OLiSyt72KSZPqgbI/Kmr4z8KNrn9ir4h0g6znH9ni9i+0f9+9279Kqab8P/DOiatLq+neGtIsNWl5kvrWxijnf6uqhj+JoX4f+GE17+3V8NaQuuZz/aQsYvtOfXzdu79aBF+68deFtH1hNJu/EWk2mqycJYzX0STt9ELbj+VLrnjXwv4evobPVvEGk6XeTnEVve3sUMknptVmBP4Vn33w+8MarrEesXvhrSLzV4zlL+4sYnnQ+okK7h+dLrXgLw14nvob7WfDmk6tewcxXN9YxTSR46bWZSR+FAF7XfGHhjwwtu2s69pWkC4x5P268jh83P8Ad3MM/hTtY8W+G/D2nw3+qa5pemWM3+qury7jiif6MzAH8DVHXvAvhzxatuuueHdK1lbbHkDULKOcRY6bd6nb+FLrHgnw/wCJNOh07VvD+l6pp8H+qtL2zjmij/3UZSB26CgC9d+LPDlnoqaxc63ptvpLcrfS3cawH6OW2/rRF4r8OXGiHWYtc0yXRwMnUEu4zbj/ALaZ2/rVK78FaBqGipot3oOmXWjIMLp81nG9uv0jK7R+VEPgvQLfQzoUWg6bFoh4OmpZxi3P/bLG39KAOY+KXikeIPg14q1LwD4o0c6nDaMbXVY7yOS2hlBByzhsDoR1r4r/AGfv+Cu2halcRaD8UbD+ydRR/KbWLBC1s7ZwMINxGT3zjmvsj4neEf8AhFfg34osPAPhHR2v5bRhbaPDZxxW88hOMOirtPU9jXwB8K/+CWNh4fki1r4k3X9o6lI/mnSbBytvG2cj94MFhnsR2ppczEfXHiDxhpHjjxVqGs6FqFvqem3MVu0dxbSB1P7vkZHcdCOoNVd9ZFl4c07wbeXOk6Vp1vpdjAkIS3toxGo+TrgAcnue5q9530r+KOJ7LOcV/iZ+3ZVD/YqXoWd9G+q3nfSjzvpXy90etyFnfRvqt530o876UXQchZ30b6red9KPO+lF0HIWd9G+q3nfSjzvpRdByFnfRvqt530o876UXQchZ3+9cp4LlEfjS75G7bJ3/wBoV0XnH2rifC8vl+Pbtv8AYk/mK/bvCVKWfS/wM+B4yXLl6fmj137UfWj7UfWsX7XR9rr+zfZn4dzG19qPrS/aj61ifa6PtdP2Ycxtfaj60faj61i/a6PtdHsw5jb+1n1pPtR9axftdH2uj2Ycxtfaj60faj61i/a6PtdHsw5ja+1H1qnrV4F0e/LNhRbyEk/7pqj9sqpq9wsmk3qMAytA4KnoRtNL2dhc5l/syeJNJ1a+vILHVLK9njZN8VvcJIy4BzkA8V9ReHfHvhnVL99LsvEWk3mpx8PZ299FJMn1QNkfiK+VP2a/DOkaPfXlxp+k2NlcSFN8tvbJGzZBzkgc19N+H/h/4Y0W+bV9P8NaRY6tLzJfW1hFHO/1dVDH8TXxuI+N3PqKXwo21+IHhhtc/sUeItJOsZx/Z326L7R/373bv0ovviB4X0vWI9JvPEek2mqyHCWM99Ek7fRC24/lWYvw/wDDC67/AG6vhrSF1zOf7SFhF9pz6+bt3frRf/D/AMMatrEesX3hrSLzV4zmO/uLGJ50PtIVLD865LGxq61498NeG76Gz1bxDpWl3kxxFb3t7FDI+em1WYE/hS65478N+F1t21nX9L0lbj/Um+vI4fMz/d3MM/hWVrXgLw14nvob7WfDmk6tewcxXF9YxTSR46bWZSR+FLr3gXw54tW3TXPDulayltjyF1CyjnEWOm3ep2/hRYDW1jxx4d8P6fDf6rr2maZYTf6q6vLyOKJ/91mYA/gaW68a+H7LRU1i41zTYNJblb+W7jWA/Rydv61lax4J8P8AiTTodO1bw/peqadB/qrS9so5oY/91GUgfgKW68FaBqGipot1oOmXOjJwunzWcb24HtGV2j8qLAakPjbw/caGdZi1zTZdHAydQjvI2tx/20B2/rSaX428Pa5pcupabrum6hp0P+su7W8jlhT6urED86zIfBegW2hnQ4tB02LRCMHTUs4xbH/tmF2/pRpfgnw/oOlzaXpmgaXp2mTf62ytLOOKGT/eRVCn8RRYDR0Px14c8UQzzaNr+l6tFb5M0ljeRzLHjruKscfjSaH498NeJrma30fxBperXEBIlisb2KZ48ddwViR+NZ2heB/DvhSGeHRPD2l6PDcZE0en2ccCyZ6hggGfxpuh+A/DfhW6nutE8OaTpFzcEmaawso4Hkz13Mqgn8aLAaOm/EDwxrOqy6Xp/iLSb7U4uJLK2vopJk+qBiR+Vfnt4y/4KPeI/wBmb9oLxL4I8fxWfirw+t409peaXKDNbQyMSqOQzAlAQCuARivvbTfh/wCGdD1aXVtO8NaRp+qTcyX1rYxRzv8A7zhQx/E1+dPxM/YB8T/tEfHjxL458cR2fhPQDdNBa2umRjzbuKNiFkcbVwXGDuyTzRy3Yj6Xvf2kfA37QWm+ENS8HazFf+Xeu8tu3yTRDyWGWRsMBnjOK3vtR9a8k8N/Avwn8C7Xw9Y+GdHjsVa5dXumG+aT923DSH5iM84zXoP2uvr8DT/cI+exUrVWbf2s+tJ9qPrWL9ro+113+zOTmNr7UfWj7UfWsX7XR9ro9mHMbX2o+tL9rPrWJ9ro+2UezDmNr7UfWj7UfWsX7ZS/a/ej2Ycxtfaj7fnSfaj61i/bKPtdHsw5ja+1H1rjdWuD/wALCU/9QoD/AMimtj7XXJapdf8AFdK3/UNx/wCRTXn46HLQZ9FkD5sfFeTOl+1UfaqxvtYo+2D1r5k/VPZGz9q7UfaTWP8AbBSfbBQHsji/jt8EtC+OXhOXTdThVbtQTb3QHzRtjg/TIFfB/gX4peNv2HvH914d1y3l1Lw7MWZbbdhXHZ0PTODz9a+7fi58ZtC+DvhSfWtZnA4Zbe3B+aZ8cAD0zivz5XRfFv7ZHjq68R63M2naHCSkMirkIvZVBwCeOfpWUt9NzwsdCMaidL+IaelWviz9s74jTeI/EUz2nha1l+WHOEUDny0B6nnn619wfDn4d2dnp9rZ2dstlo9qAiRqMbsfzJ9a+BfAXi3xB+yv8Qk0XxFHLc+GJ5gzddjKTjen5fpX6V+DfGGj+LPDtlqeh3Edzps8YaNoz046H3HeiNhYCnGo3z/H1Out2S1hWKJQiKMACpPtVY32wUfbBWp73sjZ+1fSj7TWP9sFJ9rFAeyOa8KNv+L143reD/0UK+rbf/UR/wC6P5V8meCn3/Fa5Prdj/0UK+s7f/UR/wC6P5VrLZH5/j9MTNHo9FFFcBzBRRRQAUUUUAFFFFABRRRQA1lDdqb5K1JRQBF5C0eQtS0UAReQtHkLUtFAEXkLR5C1LRQBF5C0eQtS0UAReQtHkLUtFAFW4tFkUjGc1zXiDw5AuiarIU5+yy/+gGuvrL8TH/indU/69Zf/AEA002JnzX+zj/yQvwR/2DY/616RXm/7OP8AyQvwT/2DY/616PmvIl8TPv6H8KPoLRSUtSbhRRRQAUUUUAFFFFABRRSUCOf8ff8AIo6l/wBcv6ivOlk+UV6L4+/5FHU/+uX9RXl6z/KOK/nbxJkljqN/5f1PqsnjeEvUt+ZR5lVfP9qPP9q/IOaJ9DyFrzKPMqr5/tR5/tRzoOQteZR5lVfP9qPP9qOdByFrzKPMqr5/tR5/tRzoOQteZR5lVfP9qPP9qOdByFrzKwvGDZ0sD/bH8jWn53tWN4qk36aB/t/0r3MiknmmH/xx/M87MYWwlX0Z7z8DVH/CN2HH/LIV7VFGpjB74rxb4G/8i1Yf9chXtkP+rX6V/dVT4j+f47B5Yo8sU+isihnlijyxT6KAGeWKPKFPooAZ5Yo8oU+igBnlijyxT6KAI2hGKyLrQorqTc6Z/CtukzRewj5f+J8KWfxE1aNBhRHBx/wCua8yum+MD+X8TNW7/u4P/QBXH+f7V/E3E8l/bOK/xM/eMnjfAUX5FrzKPMqr5/tR5/tXy/Mj2OQteZR5lVfP9qPP9qOZByFrzKPMqr5/tR5/tRzIOQteZR5lVfP9qPP9qOZByFrzKPMqr5/tR5/tRzIOQteZXDaFdqnjy7j6MEc/qK6/z/avP9Pk2fEC6b/pk39K/dfB60s/kl/Iz8446XLlq9T0r7Z70n2v3rH+1Ufa/rX9t+zP5+5jY+1+9L9r96xvtVH2qjkDmNn7Z70fbPesb7VR9qo5A5jY+1+9H2v3rH+1UfaqOQOY2ftnvR9s96xvtVH2qjkDmNn7Z71X1G7zp90P+mTfyNZ32qoL66/0O4/65t/Kl7MOY6r9nUDzrjPPMf8AI19W6agaBfpXyl+zn/rrj6p/I19XaX/qR9K/PcV8bPs6PwIt+WKPLFPoriNhnlCjyxT6KYDPKFHlin0UAM8sUeWKfRQAzyxR5Yp9FADPLFZ+oaWl2uCuVx0rTptFxHgf7QmmxaXB4WaNQpN8+f8Av01eb/bPevU/2om2af4WP/T+/wD6KavE/tVfcZUubDJvzPl8c7VmbH2v3pftnvWN9qo+1V6/IefzGz9s96PtnvWN9qo+1Ucgcxs/bPek+1+9Y/2qj7VRyBzGx9r96X7Z71jfa/rR9qo5A5jZ+2e9H2z3rG+1UfaqOQOY2ftnvXK6pdf8Vkp/6cMf+RK0ftVczql1/wAVUpz/AMuWP/IleXmUbYds+r4Y97M4LyZvfbPrR9s+tZH2v3pPtXua+OP3L2JsfbK4v4s/GLRfhF4XuNW1ecB1BENsD88regFbU146xsYxufHyqeAT2r4T8ZfDXx78ZPi7ey+N0k0/Q7CTEarnyynG0R9ASQeT9aiTPMx050IJUo3k9uxh6fZeKf2uPH0uv+IZZLPw1bvlYhwgXOdiA9SeOa+yfh54Gt7OxtrW0gWz0q2UKscYxuxWT8P/AANbWdjb2lrALLSbcAJGgAzj+vvXqlvIltCkUY2oowABRFdTHB5f7P35u8nuzkPjJ8FtB+L3hOTS7y2WK4jUm2uFHzRt2NfGfw9+Ivi79jv4hS+HvECS3XhmaX506qVzw6HscHp719//AGz3rzz4y/CPRfjF4blsNQiVbtV/cXIUb0OOOaTj1RWMy6U2q1B2mvxPRfC/jPTfGWh2uraTdJd2VwgdXjOcZ7Edj9a1ftlfCv7Oul/EL4K/E658MXkLS+GGJMkkzERgdFZDjrkDpX2kt6HUFW3A8giqi7nRg5vEU7yjZrc2Ptn1o+1/Wsj7X7/pR9r96Z3exE+H7bvidMfW6H/osV9cW/8AqI/90fyr5C+GrbviNIev+lD/ANF19e2/+oj/AN0fyrWeyPyjMlbF1F5no9FFFcBxhRRRQAUUUUAFFFFABRRRQAUUUUAFFFFABRRRQAUUUUAFFFFABRRRQAVn69A91omoQxLvlkt5EVfUlSAK0Kayk5oEfCnhD4k+KPgv4H0Dwtrnw01o3em2iQvLHdW218Z+YfPnH1q7/wANWXX/AETjXP8AwKtv/i6+n/HHwt0zxddG5u0kaXYFyr4GBn/GuFk/Z50XccRzY/36ao0ZayPQWYYiKUU9Dxr/AIatuv8AonGuf+BVt/8AF0f8NW3X/RONc/8AAq2/+Lr2L/hnnRv+eU3/AH3R/wAM86N/zym/77p/V6A/7SxXc8d/4atuv+ica5/4FW3/AMXR/wANW3X/AETjXP8AwKtv/i69i/4Z50b/AJ5Tf990f8M86N/zym/77o+r0A/tLFdzx3/hq26/6Jxrn/gVbf8AxdH/AA1bdf8ARONc/wDAq2/+Lr2L/hnnRv8AnlN/33R/wzzo3/PKb/vuj6vQD+0sV3PHf+Grbr/onGuf+BVt/wDF0f8ADVt1/wBE41z/AMCrb/4uvYv+GedG/wCeU3/fdH/DPOjf88pv++6Pq9AP7SxXc8d/4atuv+ica5/4FW3/AMXR/wANWXX/AETjXP8AwKtv/i69i/4Z50b/AJ5Tf990v/DPGjf88pv+/lH1egH9o4rueJ6h+0ZP4os5dLk8D6vpqXQ8s3c9xAyR98kBskfSqUPi60m3BFZvLbY31Fe8/wDDPOj/APPKbP8A10qCP9nLRLXzNkM37xy5/edzXxOe8GZbn1WNbETknFW0t/kerg+Icbg4uMUnc8R/4Si3/wCeb0f8JRb/APPN690H7POj4/1U3/fyl/4Z50b/AJ5Tf99181/xC3Jf+fs/vX+R3/624/8Alj/XzPCv+Eot/wDnm9H/AAlFv/zzevdf+GedG/55Tf8AfdH/AAzzo3/PKb/vuj/iF2S/8/Z/ev8AIP8AW3H/AMsf6+Z4V/wlFv8A883o/wCEot/+eb17r/wzzo3/ADym/wC+6P8AhnnRv+eU3/fdH/ELsl/5+z+9f5B/rbj/AOWP9fM8K/4Si3/55vR/wlFv/wA83r3X/hnnRv8AnlN/33R/wzzo3/PKb/vuj/iF2S/8/Z/ev8g/1tx/8sf6+Z4V/wAJRb/883o/4Si3/wCeb17r/wAM86N/zym/77o/4Z50b/nlN/33R/xC7Jf+fs/vX+Qf624/+WP9fM8K/wCEot/+eb1S1bWotQtfKRGDZzzX0F/wzzo3/PKb/vulX9nnRt2fKm/7+V1YXw3yjB14YiFSd4tPddPkY1uKMdXpypSSs9P61L3wNUr4bsMj/lkK9rh/1a/SuL8I+FYfDlpDawKViiXaoPJrtI/uCv1So+Z3Pj4qyH0UUVkUFFFFABRRRQAUUUUAFFFFACUmeKdSY9KAPl/9oGzvNB8V32u3Vqy6VN5MSXIYY3BOmOvavG3+JujRsQ0rAj2Nfa/xE+H+m/ELRf7N1RZGtvMEo8p9h3AEDn8TXiGofsm+FvPYpBdYz/z3NfmOM8O8qzHE1MXWqTUpu7s1b8j6vD8TYzCUY0aaVor+up4r/wALQ0X/AJ6t+Ro/4Whov/PVvyNewf8ADJ3hj/nhdf8Af80f8MneGP8Anhdf9/zXJ/xC3JP+fk/vX+R0f63Zh/LH+vmeP/8AC0NF/wCerfkaP+FoaL/z1b8jXsH/AAyd4Y/54XX/AH/NH/DJ3hj/AJ4XX/f80f8AELcl/wCfk/vX+Qf63Zh/LH+vmeP/APC0NF/56t+Ro/4Whov/AD1b8jXsH/DJ3hj/AJ4XX/f80f8ADJ3hj/nhdf8Af80f8QtyX/n5P71/kH+t2Yfyx/r5nj//AAtDRf8Anq35Gj/haGi/89W/I17B/wAMneGP+eF1/wB/zR/wyd4Y/wCeF1/3/NH/ABC3Jf8An5P71/kH+t2Yfyx/r5nj/wDwtDRf+erfkaP+FoaL/wA9W/I17B/wyf4Y/wCeF1/3/NH/AAyd4Y/54XX/AH/NH/ELcl/5+T+9f5B/rdmH8sf6+Z4//wALQ0X/AJ6t+RrkrPxdbXHjiY28bzK8LMrLwD0z1r6N/wCGTvDH/PC6/wC/5qSx/ZW8OafqCXkMNz5yoYxumJGDjP8AKvreGuEsv4Xxn13Bzk5NW1tb8EjxM2zfE5zQWHrpJXvoeMf8JE3/AD6yfmKP+Eib/n1k/MV9D2/7PejsOYpf++6l/wCGedG/55zf991+rf2zU7I+M/syn3Z85/8ACRN/z6yfmKP+Eib/AJ9ZPzFfRn/DPOjf885v++6P+GedG/55zf8AfdP+2anZD/syn3Z85/8ACRN/z6yfmKP+Eib/AJ9ZPzFfRn/DPOjf885v++6P+GedG/55zf8AfdH9s1OyD+zKfdnzn/wkTf8APrJ+Yo/4SJv+fWT8xX0Z/wAM86N/zzm/77o/4Z50b/nnN/33R/bNTsg/syn3Z85/8JE3/PrJ+Yo/4SJv+fWT8xX0Z/wzzo3/ADzm/wC+6P8AhnnRv+ec3/fdH9s1OyD+zKfdnzn/AMJE3/PrJ+Ypk2vNLC6fZZBuUjqO4r6P/wCGedG/55zf990f8M86N/zzm/77pf2zU7IP7Mp92cJ+zrGyTXO4Y+ZP5GvqzS/9SPpXm/g34Y2XhGSQ2iOvmEFtxz0r0yzj8uMDHavna0+eTketCPKrFmiiiuc0CiiigAooooAKKKKACiiigAptOpP5UgPG/wBpLwzq/iLRdFfSrFr5rO6aaVUZVKr5ZGeTXzBc6xPZybJbKVG9Ny/4199ahai8tpImyVdSp/EYryDV/gHoV1KWEc3X/npXuYPMqmFp+zS0PNr4GFefPJny9/wkTf8APrJ+Yo/4SJv+fWT8xX0b/wAM86N/zym/77pP+GedG/55zf8Afdd/9s1OyOf+zKfdnzn/AMJE3/PrJ+Yo/wCEib/n1k/MV9Gf8M86N/zzm/77o/4Z50b/AJ5zf990/wC2anZB/ZlPuz5z/wCEib/n1k/MUf8ACRN/z6yfmK+jP+GedG/55zf990f8M86N/wA85v8Avuj+2anZB/ZlPuz5z/4SJv8An1k/MUf8JE3/AD6yfmK+jP8AhnnRv+ec3/fdH/DPOjf885v++6P7ZqdkH9mU+7PnP/hIm/59ZPzFH/CRN/z6yfmK+jP+GedG/wCec3/fdH/DPOjf885v++6P7ZqdkH9mU+7PnP8A4SJv+fWT8xWFrOszR6kL77HKYlgER5GQS+f619U/8M86N/zzm/77qO5/Zx0O7hMUkMxRiCcSehzXNXzKWIh7OS0PRy+l/ZuIWJo7rufLf9sXn/QOn/76X/Gj+2Lz/oHT/wDfS/419TL+z3o+7HlTY/66U7/hnnRv+eU3/fdeXzUz67/WHG9l9x8r/wBtXf8A0DZ/++l/xrM1OBtTmjkk0ycMpydrryPQ819d/wDDPOjf88pv++6P+GedG/55Tf8AfdK8CJZ/i5aSSPlSHVLmCNY00uZVUYADL/jT/wC2Lz/oHT/99L/jX1R/wzzo3/PKb/vuj/hnnRv+eU3/AH3TvAv/AFhxvRL7j5X/ALYvP+gdP/30v+NH9sXf/QNn/wC+l/xr6o/4Z50b/nlN/wB90f8ADPOjf88pv++6LwD/AFhxvZfcfIurQtq21pNLmVwfvqy5+nWr1vqVzbQJEmmzhFAA+df8a+rP+GedG/55Tf8AfdH/AAzzo3/PKb/vui8CVn+LTvZfcfK/9sXn/QOn/wC+l/xo/ti8/wCgdP8A99L/AI19Uf8ADPOjf88pv++6P+GedG/55Tf990c0Cv8AWLG9l9x87fCmOeTxvFPLC0Pm3G4KxGcbMV9g24PkR/7o/lXL6H8DtL0XUIbyCOUSxHIy/FeiJpO1FG3oMVE5J7HzterPEVJVZ7s6qiiiuQgKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigBrIG603yFqSigCL7OtH2dalooAi+zrR9nWpaKdwIvs60fZ1qWii4EX2daPs61LRRcCL7OtKIU/u5qSikBGYU9MUht0NS0UARfZ1o+zrUtFO4EX2daPs61LRRcCL7OtH2daloouBF9nWj7OtS0UXAi+zrR9nWpaKLgRfZ1o8halopARCBV6VJ0paKACiiigAooooAKKKKACiiigAooooAKKKKAEZQ1QPaqx5GasUUAVvscf9wUfY4/7gqzRTuBW+xx/3BR9jj/uCrNFFwK32OP8AuCj7HH/cFWaKLgVvscf9wUv2KL+6KsUUXAr/AGKL+7SfY4/7gqzRRcCt9jj/ALgo+xp/cFWaKLgQLaovanfZ1qWii4EX2daPs61LRRcCL7OtH2daloouBF9nWj7OtS0UXAi+zrR9nWpaKLgRfZ1o+zrUtFICL7OtPVQowKdRQAUUUUAFFFFABRRRQAUUUUAFFFFABRRRQAUxog1PooAi+zrR9nWpaKNQIvs60fZ1qWincCL7OtH2daloouBF9nWj7OtS0UXAi+zrR9nWpaKLgRfZ1o+zrUtFFwIfsqdcUv2daloouBF9nWj7OtS0UXAi+zrR9nWpaKLgRfZ1o+zrUtFFwIvs60fZ1qWii4EX2daPs61LRSAi+zrR9nWpaKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKAEzS01UCLgZ6k8/XNOoAKKKKACiiigAopCaWgAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKAP/2Q==)**

**Program:**

#include <iostream>

#include <stdlib.h>

using namespace std;

struct node {

int data;

struct node \*next;

}\*head;

void createList(int n);

void deleteNodeAtBeginning();

void deleteNodeAtEnd();

void deleteNodeAtLoc();

void displayList();

int main()

{

int n, data,choice;

cout<<"Enter the total number of nodes: ";

cin>>n;

createList(n);

cout<<"\n Data in the list:"<<" ";

displayList();

while(1){

cout<<"\n\nSelect an Option";

cout<<"\n1.Deletion At Beginning\n"<< "2.Deletion At End\n"<<"3.Deletion At Specific Location\n"<<"4.Exit\n";

cout<<"\nEnter Your choice:";

cin>>choice;

switch(choice){

case 1:{

deleteNodeAtBeginning();

cout<<"\n Data In the list: "<<" ";

displayList();

break;

}

case 2:{

deleteNodeAtEnd();

cout<<"\nData in the list: "<<" ";

displayList();

break; }

case 3:{

deleteNodeAtLoc( );

cout<<"\nData in the list: "<<" ";

displayList();

break;

}

case 4:{

cout<<"exit";

exit(1);

}

default:{

cout<<"\n Incorrect choice.";

break;}}}

return 0;

}

void createList(int n){

struct node \*ptr, \*temp;

int data, i;

head = (struct node \*)malloc(sizeof(struct node));

if(head == NULL)

{

cout<<"unable to allocate memory\n"; }

else {

cout<<"Enter the data of node 1: ";

cin>>data;

head->data = data;

head->next = NULL;

temp = head;

for(i=1; i<n; i++)

{

ptr = (struct node \*)malloc(sizeof(struct node));

if(ptr == NULL)

{

cout<<"Unable to allocate memory.";

break;

}

else

{

cout<<"Enter the data of node: "<<i+1<<":";

cin>>data;

ptr->data = data;

ptr->next = NULL;

temp->next = ptr;

temp = temp->next;

}

} cout<<"SINGLY LINKED LIST CREATED SUCCESSFULLY\n";

}}

void deleteNodeAtBeginning()

{

struct node \*ptr;

if(head == NULL)

{

cout<<"Underflow";

}

else{

ptr=head;

head=head->next;

free(ptr); }

}

void deleteNodeAtEnd()

{

struct node \*ptr,\*temp;

if(head == NULL)

{

cout<<"Underflow";

}

else

{

ptr=head;

while(ptr->next!=NULL){

temp=ptr;

ptr=ptr->next;

}

temp->next=NULL;

free(ptr);}

}

void deleteNodeAtLoc()

{

struct node \*ptr,\*temp;

int pos;

if(head == NULL)

{

cout<<"Underflow";

}

else

{

int i=0;

cout<<"\n enter the location to be deleted:";

cin>>pos;

ptr=head;

while(i<pos){

temp=ptr;

ptr=ptr->next;

i=i+1

}

temp->next=ptr->next;

free(ptr);}

}

void displayList()

{

struct node \*temp;

temp = head;

while(temp != NULL)

{

cout<<temp->data<<" ";

temp = temp->next;

}

}

**OUTPUT:**

Enter the total number of nodes: 5

Enter the data of node 1: 11

Enter the data of node: 2:22

Enter the data of node: 3:33

Enter the data of node: 4:44

Enter the data of node: 5:55

SINGLY LINKED LIST CREATED SUCCESSFULLY

Data in the list: 11 22 33 44 55

Select an Option

1.Deletion At Beginning

2.Deletion At End

3.Deletion At Specific Location

4.Exit

Enter Your choice:1

Data in the list: 22 33 44 55

Select an Option

1.Deletion At Beginning

2.Deletion At End

3.Deletion At Specific Location

4.Exit

Enter Your choice:2

Data in the list: 22 33 44

Select an Option

1.Deletion At Beginning

2.Deletion At End

3.Deletion At Specific Location

4.Exit

Enter Your choice:3

enter the location to be deleted:1

Data in the list: 22 44

Select an Option

1.Deletion At Beginning

2.Deletion At End

3.Deletion At Specific Location

4.Exit

Enter Your choice:4

exit

**DOUBLE LINKED LIST**

In a doubly linked list, a node consists of three parts: node data, pointer to the next node in sequence (next pointer) , pointer to the previous node (previous pointer).



**Structure of Node in Double Linked List:**

struct node

{

    struct node \*prev;

**int** data;

    struct node \*next;   }

A node can be added in three ways

* Insertion at beginning of list
* Insertion at specified position of list
* Insertion at end of list

**Insertion at beginning of list**

**Algorithm:**

**Step 1**: Create a NewNode and allocate memory for NewNode

**Step 2:** SET NEW\_NODE -> DATA = DATA

**Step 3:** SET NEW\_NODE -> PREV = NULL

**Step 4:** SET NEW\_NODE -> NEXT =HEAD

**Step 5:** SET HEAD -> PREV = NEW\_NODE

**Step 6:** SET HEAD = NEW\_NODE

**Step 7:** EXIT

Diagram

Description automatically generated

**Insertion at End of list**

**Algorithm:**

**Step 1**: Create a NewNode and allocate memory for NewNode

**Step 2:** SET NEW\_NODE -> DATA = DATA

**Step 3:** SET NEW\_NODE -> NEXT = NULL

**Step 4:** SET NEW\_NODE -> PREV= TAIL

**Step 5:** SET TAIL->NEXT=NEW\_NODE

**Step 6:** SET TAIL=NEW\_NODE

**Step 7:** EXIT

A picture containing diagram

Description automatically generated

**Insertion at Specified Position of list**

**Algorithm:**

**Step 1**: Create a NewNode and allocate memory for NewNode

**Step 2:** SET NEW\_NODE -> DATA = DATA

**Step 3:** SET TEMP = HEAD

**Step 4:** SET I = 0

**Step 5:** REPEAT 6 and 7 until I<POS-1

**Step 6:** SET TEMP = TEMP -> NEXT

**Step 7:** SET I=I+1

[END OF LOOP]

**Step 8:** SET NEW\_NODE -> NEXT = TEMP -> NEXT

**Step 9:** SET NEW\_NODE -> PREV = TEMP

**Step 10:** SET TEMP -> NEXT = NEW\_NODE

**Step 11:** SET TEMP -> NEXT -> PREV = NEW\_NODE

**Step 12:** EXIT

Diagram

Description automatically generated

**PROGRAM:**

#include <iostream>

#include <stdlib.h>

using namespace std;

struct node {

int data;

struct node \*next;

struct node \*prev;

}\*head,\*tail;

void insert(int data){

struct node \*new\_node;

new\_node = (struct node \*)malloc(sizeof(struct node));

new\_node->data=data;

new\_node->prev=NULL;

new\_node->next=NULL;

if(head == NULL){

head=new\_node;

tail=new\_node;

} else {

tail->next = new\_node;

new\_node->prev=tail;

tail=new\_node;

}}

void InsertAtBeg(int data){

struct node \*new\_node;

new\_node = (struct node \*)malloc(sizeof(struct node));

new\_node->data=data;

new\_node->prev=NULL;

new\_node->next=head;

head->prev=new\_node;

head=new\_node;

}

void InsertAtEnd(int data){

struct node \*new\_node;

new\_node = (struct node \*)malloc(sizeof(struct node));

new\_node->data=data;

new\_node->prev=tail;

new\_node->next=NULL;

tail->next=new\_node;

tail=new\_node;

}

void InsertAtLoc(int data,int pos){

struct node \*new\_node,\*temp;

new\_node = (struct node \*)malloc(sizeof(struct node));

temp=head;

for(int i=0;i<pos-1;i++){

temp=temp->next;

}

new\_node->data=data;

new\_node->next=temp->next;

temp->next->prev=new\_node;

temp->next=new\_node;

new\_node->prev=temp;

}

void display(){

struct node \*temp;

temp=head;

while(temp!=NULL){

cout<<temp->data<<" ";

temp=temp->next;

}}

int main(){

int choice;

insert(3);

insert(4);

insert(7);

insert(9);

cout<<"The doubly linked list is:"<<" ";

display();

while(1){

cout<<"\n\nSelect an Option";

cout<<"\n1.Insertion At Beginning\n"<< "2.Insertion At End\n"<<"3.Insertion At Specific Location\n"<<"4.Exit\n";

cout<<"\nEnter Your choice:";

cin>>choice;

switch(choice){

case 1:{

InsertAtBeg(8);

cout<<"\n The doubly linked list after insertion at beginning: "<<" ";

display();

break;}

case 2:{

InsertAtEnd(25);

cout<<"\nThe doubly linked list after insertion at End: "<<" ";

display();

break; }

case 3:{

InsertAtLoc(75,2);

cout<<"\nThe doubly linked list after insertion at specific Location: "<<" ";

display();

break; }

case 4:{

cout<<"exit";

exit(1); }

default:{

cout<<"\n Incorrect choice.";

break; } }

}

return 0;

}

**OUTPUT:**

The doubly linked list is: 3 4 7 9

Select an Option

1.Insertion At Beginning

2.Insertion At End

3.Insertion At Specific Location

4.Exit

Enter Your choice:1

The doubly linked list after insertion at beginning: 8 3 4 7 9

Select an Option

1.Insertion At Beginning

2.Insertion At End

3.Insertion At Specific Location

4.Exit

Enter Your choice:2

The doubly linked list after insertion at End: 8 3 4 7 9 25

Select an Option

1.Insertion At Beginning

2.Insertion At End

3.Insertion At Specific Location

4.Exit

Enter Your choice:3

The doubly linked list after insertion at specific Location: 8 3 75 4 7 9 25

Select an Option

1.Insertion At Beginning

2.Insertion At End

3.Insertion At Specific Location

4.Exit

Enter Your choice:4

Exit

**Deletion in Double Linked List**

A node can be deleted in three ways

* Deletion at beginning of list
* Deletion at specified position of list
* Deletion at end of list

**Deletion at beginning of list**

**Algorithm:**

**Step 1:** IF HEAD = NULL

Write Underflow

Go to step 7

[End of If]

**STEP 2:** SET TEMP= HEAD

**STEP 3:** SET HEAD = HEAD → NEXT

**STEP 4:** SET TEMP->NEXT=NULL

**STEP 5:** SET HEAD → PREV = NULL

**STEP 6** FREE TEMP

**STEP 7:** EXIT

**![Chart

Description automatically generated](data:image/jpeg;base64,/9j/4AAQSkZJRgABAQEAAAAAAAD/4RCyRXhpZgAATU0AKgAAAAgAAodpAAQAAAABAAAIMuocAAcAAAgMAAAAJgAAAAAc6gAAAAgAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAFkAMAAgAAABQAABCAkAQAAgAAABQAABCUkpEAAgAAAAMwMAAAkpIAAgAAAAMwMAAA6hwABwAACAwAAAh0AAAAABzqAAAACAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAMjAyMzowMToxMSAxNDowMDowNQAyMDIzOjAxOjExIDE0OjAwOjA1AAAA/+EJnGh0dHA6Ly9ucy5hZG9iZS5jb20veGFwLzEuMC8APD94cGFja2V0IGJlZ2luPSfvu78nIGlkPSdXNU0wTXBDZWhpSHpyZVN6TlRjemtjOWQnPz4NCjx4OnhtcG1ldGEgeG1sbnM6eD0iYWRvYmU6bnM6bWV0YS8iPjxyZGY6UkRGIHhtbG5zOnJkZj0iaHR0cDovL3d3dy53My5vcmcvMTk5OS8wMi8yMi1yZGYtc3ludGF4LW5zIyI+PHJkZjpEZXNjcmlwdGlvbiByZGY6YWJvdXQ9InV1aWQ6ZmFmNWJkZDUtYmEzZC0xMWRhLWFkMzEtZDMzZDc1MTgyZjFiIiB4bWxuczp4bXA9Imh0dHA6Ly9ucy5hZG9iZS5jb20veGFwLzEuMC8iPjx4bXA6Q3JlYXRlRGF0ZT4yMDIzLTAxLTExVDE0OjAwOjA1PC94bXA6Q3JlYXRlRGF0ZT48L3JkZjpEZXNjcmlwdGlvbj48L3JkZjpSREY+PC94OnhtcG1ldGE+DQogICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgCiAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAKICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgIAogICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgCiAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAKICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgIAogICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgCiAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAKICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgIAogICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgCiAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAKICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgIAogICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgCiAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAKICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgIAogICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgCiAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAKICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgIAogICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgCiAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAKICAgICAgICAgICAgICAgICAgICAgICAgICAgIDw/eHBhY2tldCBlbmQ9J3cnPz7/2wBDAAIBAQEBAQIBAQECAgICAgQDAgICAgUEBAMEBgUGBgYFBgYGBwkIBgcJBwYGCAsICQoKCgoKBggLDAsKDAkKCgr/2wBDAQICAgICAgUDAwUKBwYHCgoKCgoKCgoKCgoKCgoKCgoKCgoKCgoKCgoKCgoKCgoKCgoKCgoKCgoKCgoKCgoKCgr/wAARCAEfBAoDASIAAhEBAxEB/8QAHwAAAQUBAQEBAQEAAAAAAAAAAAECAwQFBgcICQoL/8QAtRAAAgEDAwIEAwUFBAQAAAF9AQIDAAQRBRIhMUEGE1FhByJxFDKBkaEII0KxwRVS0fAkM2JyggkKFhcYGRolJicoKSo0NTY3ODk6Q0RFRkdISUpTVFVWV1hZWmNkZWZnaGlqc3R1dnd4eXqDhIWGh4iJipKTlJWWl5iZmqKjpKWmp6ipqrKztLW2t7i5usLDxMXGx8jJytLT1NXW19jZ2uHi4+Tl5ufo6erx8vP09fb3+Pn6/8QAHwEAAwEBAQEBAQEBAQAAAAAAAAECAwQFBgcICQoL/8QAtREAAgECBAQDBAcFBAQAAQJ3AAECAxEEBSExBhJBUQdhcRMiMoEIFEKRobHBCSMzUvAVYnLRChYkNOEl8RcYGRomJygpKjU2Nzg5OkNERUZHSElKU1RVVldYWVpjZGVmZ2hpanN0dXZ3eHl6goOEhYaHiImKkpOUlZaXmJmaoqOkpaanqKmqsrO0tba3uLm6wsPExcbHyMnK0tPU1dbX2Nna4uPk5ebn6Onq8vP09fb3+Pn6/9oADAMBAAIRAxEAPwD98tF/5Bsf/Av/AEI1aqrov/INj/4F/wChGrVBUviYUUUUEhRRRQAUUUUAFFFFABRRRQAUUUUAFFFFABRRRQAUUUUAFFFFABRRRQAUUUUAFFFFABRRRQAUUUUAFFFFABRRRQAUUUUAFFFFABRRRQAUUUUAFFFFABRRRQAUUUUAFFFFABRRRQAUUUUAFFFFABRRRQAUUUUAFFFFABRRRQAUUUUAFFFFABRRRQAUUUUAFFFFABRRRQAUUUUAFFFFABRRRQAUUUUAFFFFABRRRQAUUUUAFFFFABRRRQAUUUUAFFFFABRRRQAUUUUAFFFFABRRRQAUUUUAFFFFABRRRQAUUUUAFFFFABRRRQAUUUUAFFFFABRRRQAUUUUAFFFFAFXRf+QbH/wL/wBCNWqq6L/yDY/+Bf8AoRq1QVL4mFFFfOvxE/bT+LPh/wCNfi74SfDv9nHT/EEHhK5s4LjVrzx0LAzPcWUN0MRfY5MACbb985254zgZVq1OhFOb302b/I7MBluMzKpKnh4puKu7uMbK6V7yaW7S76n0VRXzP/w2f+07/wBGf6H/AOHUH/yvo/4bP/ad/wCjP9D/APDqD/5X1h/aGF7v7pf5Hpf6r51/JH/wZT/+TPpiivmf/hs/9p3/AKM/0P8A8OoP/lfR/wANn/tO/wDRn+h/+HUH/wAr6P7Qwvd/dL/IP9V86/kj/wCDKf8A8mfTFFfM/wDw2f8AtO/9Gf6H/wCHUH/yvo/4bP8A2nf+jP8AQ/8Aw6g/+V9H9oYXu/ul/kH+q+dfyR/8GU//AJM+mKK+Z/8Ahs/9p3/oz/Q//DqD/wCV9H/DZ/7Tv/Rn+h/+HUH/AMr6P7Qwvd/dL/IP9V86/kj/AODKf/yZ9MUV8z/8Nn/tO/8ARn+h/wDh1B/8r6P+Gz/2nf8Aoz/Q/wDw6g/+V9H9oYXu/ul/kH+q+dfyR/8ABlP/AOTPpiivmf8A4bP/AGnf+jP9D/8ADqD/AOV9H/DZ/wC07/0Z/of/AIdQf/K+j+0ML3f3S/yD/VfOv5I/+DKf/wAmfTFFfM//AA2f+07/ANGf6H/4dQf/ACvo/wCGz/2nf+jP9D/8OoP/AJX0f2hhe7+6X+Qf6r51/JH/AMGU/wD5M+mKK+Z/+Gz/ANp3/oz/AEP/AMOoP/lfXTfs+/tgeJPiv478RfDv4kfCO18J6hoNnZ3K/ZfE41FLlZzJjn7PDtI8v3zntVU8Zh6tRQi3d+T/AFRjiOH82wmHlXqQXLHV2nB2u0tlJvdroe5UVHa3UN5EJoG3KakrqPFCiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKAKui/8g2P/AIF/6EatVV0X/kGx/wDAv/QjVqgqXxMK+Mtcuzb/ALYHxoUHrrWin/yiWdfZtfD/AI5uzbftjfGRc9dY0X/0yWdebmj5aMX/AHv0Z9hwVD2mOrx/6d/+5KZ1n9qN6mj+1G9TXPf2of7360f2of73614/tD776p5HQ/2o3qaP7Ub1Nc9/ah/vfrR/ah/vfrR7QPqnkdD/AGo3qaP7Ub1Nc9/ah/vfrR/ah/vfrR7QPqnkdD/ajepo/tRvU1z39qH+9+tH9qH+9+tHtA+qeR0P9qN6mj+1G9TXPf2of7360f2of7360e0D6p5HQ/2o3qaP7Ub1Nc9/ah/vfrR/ah/vfrR7QPqnkdD/AGo3qaP7Ub1Nc9/ah/vfrR/ah/vfrR7QPqnkdD/ajepryPQ/E+o6X+134mNjOy+doWlB8d8Gb/Gu7/tQ/wB79a8s0Kfz/wBrHxBJ/wBQTTf5zV35VLmzKmn3f/pLPJz7D+zyHEv+7H/0uB+gXwuuprvw3FNO+5igrpq5X4Sf8itD/wBcxXVV71T42fki2CiiioGFFFFABRRRQAUUUUAFFFFABRRRQAUUUUAFFFFABRRRQAUUUUAFFFFABRRRQAUUUUAFFFFABRRRQAUUV43/AMFCtf1zwv8AsT/ErxD4a1i60+/s/C88lre2U7RSwuCMMrqQVPuDQB7JRX85P7E/wI/4KRft8/8ABNg/FPxx/wAFO/iB4Vh8PzX0/hWx0m9m+1X+2JG/0q9EyzMpwoAO4LliOpr2r9iv/guH+0X8JP8AgkF8Qvid8ZvESeI/Fvw91GXQ9C1e8w0lw/miNZJSQTIV37stkkLWyo1HG9vMXMj9zKK/l5u/2xfGnw6+Dtr/AMFCtF/4LoXmv/GhWGrXnwWbUZW0tpCPMa2WAyeXnd8uzy9o6V9lftw/t+/F7xN8Nv2Yf+Cj/g74l+I9L0261mztPGXh3S9Ymhs7sTTCHMkSMEb5pM8jtUqlOUboLpH7eUV/PL/wWr/4Km/tJaZ/wUZ+Hfij4C/EHxbY+CPBWn2OoeKbHQdRlitZBcxwuPPSNwsgxIeGBFfSHx3/AGsfij+2f/wV2+DvwX+Enxd8RaP4X8K+H08ReKLPRdZmtYrtX5RZhE4EqnyGG1sgc+pp+xqdg5kfr94g1eLw/oN9r08LSJY2clw8a9WCKWIHvxXyZ+x9/wAFf/ht+2N8OPid8QvBXwY8R2H/AArPUriyutNuLiGWfUZImkB8kR56+WcA8819A614ti1f4c69bRMzKuh3S5bknELV+KX/AASB8Sa34a/Z1/ao1HQNVuLO6i8V6m8NzazNHJGwluMMrKQQR6iiFKUp8oX0ufrr+wj+3FY/tvfB5vi03wW8UeAduoSWv9jeLYPLuTs/5aYwPlPavdVYOu5TxX4OfsjftHfHjW/+CFfxK8f6x8aPFF54itbi8Fprl1r1xJeQbSMBJmfeuPY15f8ABX9nv/gpP+1r/wAExNN/ai8c/wDBTrx94c1Tw1ptzP4b8P6PeTRLewpIcG6ukmWSRiAPvK2MkVcsPLTl1FzH7gf8FEf26fB//BOz9mfVv2mfHPgbUvEWn6TJGkmm6TPHHNJvOODJ8tdt+zh+0B4d/aQ+Cvhv41aDpVxptr4k0uO9t7G8kVpYVcZ2sV4J+lfhZ8aP26/jB+2J/wAG3Ooa58ddaXU/EFjqkenzantw9ysXl4Z+OWJY5PJNedfAP/go58XP+ClepfBz/gnV+zD8YdW+F/hfQ9JgPjbxKLxrLUbwxA5htWibJyQv8a8ZPtS9hLaXyDmP6Vs55FFeX/Agaf4F8E6R8ONG1zUtUh0qzjtkv9XvnubifaoXfJJISzscZJJJr1BSSoJrKUZR3KCiiipAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAq6L/AMg2P/gX/oRq1VXRf+QbH/wL/wBCNWqCpfEwr4N+Kt2bf9sz4wqG66tov/pls6+8q/Pn453TW/7aXxeCnrqmjf8Apls68nOHy4eL/vfoz7vw9h7TNq0f+nT/APS6Zof2of79H9qH+/XO/wBpSf3qP7Sk/vV877Q/WfqZ0X9qH+/WfZfEDwvqOuXPhnT/ABPYz6lZor3mnw3iNNApJALoDuUEg9R2rN/tKT+9XwrN+2b+zX+yx/wUV+JFz8fvi5p/hn+0tCshZ/bY5W80iWfOPLRvXvWlNSq35dWl/kceMlRwPI6rSjJ2u3ZLRvd+h+gWt+MtE8M6ZJrXiPXLXT7OHBmury4WKNMnAyzEAckD8ayfGHxu+Fnw90OHxP49+Jmg6LptwQLfUNW1eG3hlz02u7BTn2NfCX7aX/BU/wDYI+Lv7O2vfD74fftMaLqmsal9njsbC3t7kNM32iM4G6IDoD1NZ3/BSiL4Iap+zT8H7D9o67hh8GtrNgNakmkkVRDlN2TF8w4z05raFGfNFTTV3212PPxGPw6hUlh5QnyRTvzK122tWr2XU+1P+G4v2Qv+jqPh7/4Wdl/8drvPDPj3w1400aHxH4P8S2Wq6fcLm3vtOu0mhkHqroSD+Br8l7b4c/8ABs5OY4V8WaC0j4G3+2NZ5Y1+ivwE074LfC74F6Vp/wADPJt/Btrp4n0llnkdPJKhg26U7sEY60VoxpxVlL5qxeW1K+MqSU5UmkvsT5n89FZHqMnxC8LQ+IY/CMviixXVZIvNj01rxBcMnPzCPO4jg84xxVnVPFGm6Jp82razqkNpa26F7i5uZhHHGvqzEgAfWvyx8Sar8RNa+Kt1/wAFQLHWL9dP0vxTHpFtpySExnR/MQPKVPGAsspyPSvuD9ojxbpfjv8AZK8Ra/plws1lqnhnzomVuGjkCsP0NTOHI4676PyfY1w9dYiFR8tuVXX96Otn87M6v/huL9kL/o6j4e/+FnZf/Ha73w7448P+L9Hg8ReFPENnqen3S7re+0+6WaGVc4yroSCM+hr85v2SP+CN/wDwTt+Jf7N3g7x542+BJvNV1TQ7e4vrn+3rxPMkZAScLKAOfQV0X7Onwr1D/gnv+2nH8BvBPjnULz4d+PLFrrQ9B1C6ec6TcIGzHGzkkR7Yt3Xqxqpxo6qEndd129Gc+HqZklTqYmlFQnZJxk203tdNLTpo9z9BP7UP9+j+1D/frnf7Sk/vUf2lJ/erl9oe59TOi/tQ/wB+uC8Ezmf9qTxBIT/zB9NH6y1t/wBpSf3q4fwZ438JeHf2mNdbxR4q03TjJpOn+X9uvo4d2DL03EZr1MklzZpT+f8A6Sz53i7D+z4bxL8o/wDpcD9JvhJ/yK0P/XMV1VeV/Cj46/BGLwxCsvxj8KqfLHDeIbYf+z11H/C+vgZ/0Wfwn/4UVr/8XX09T42fhUdjrKK5P/hfXwM/6LP4T/8ACitf/i6P+F9fAz/os/hP/wAKK1/+LqBnWUVyf/C+vgZ/0Wfwn/4UVr/8XR/wvr4Gf9Fn8J/+FFa//F0AdZRXJ/8AC+vgZ/0Wfwn/AOFFa/8AxdH/AAvr4Gf9Fn8J/wDhRWv/AMXQB1lFcn/wvr4Gf9Fn8J/+FFa//F0f8L6+Bn/RZ/Cf/hRWv/xdAHWUVyf/AAvr4Gf9Fn8J/wDhRWv/AMXR/wAL6+Bn/RZ/Cf8A4UVr/wDF0AdZRXJ/8L6+Bn/RZ/Cf/hRWv/xdH/C+vgZ/0Wfwn/4UVr/8XQB1lFcn/wAL6+Bn/RZ/Cf8A4UVr/wDF0f8AC+vgZ/0Wfwn/AOFFa/8AxdAHWUVyf/C+vgZ/0Wfwn/4UVr/8XR/wvr4Gf9Fn8J/+FFa//F0AdZRXJ/8AC+vgZ/0Wfwn/AOFFa/8AxdH/AAvr4Gf9Fn8J/wDhRWv/AMXQB1lFcn/wvr4Gf9Fn8J/+FFa//F0f8L6+Bn/RZ/Cf/hRWv/xdAHWUVyf/AAvr4Gf9Fn8J/wDhRWv/AMXR/wAL6+Bn/RZ/Cf8A4UVr/wDF0AdZRXJ/8L6+Bn/RZ/Cf/hRWv/xdH/C+vgZ/0Wfwn/4UVr/8XQB1lFcn/wAL6+Bn/RZ/Cf8A4UVr/wDF0f8AC+vgZ/0Wfwn/AOFFa/8AxdAHWUVyf/C+vgZ/0Wfwn/4UVr/8XR/wvr4Gf9Fn8J/+FFa//F0AdZRXJ/8AC+vgZ/0Wfwn/AOFFa/8AxdH/AAvr4Gf9Fn8J/wDhRWv/AMXQB1leNf8ABQvQtb8TfsTfErQPDmkXWoX114XnjtbOzgaWWZyRhVVQSx9gK7n/AIX18DP+iz+E/wDworX/AOLob48/AxlKn4z+E/8AworX/wCLoA/Kr/gjh8IfiT4C/wCCVNr4G+Ifw71zRNaW1vh/Y+raXLb3WTCgA8qRQ3JHHFfMH7G//BOT4v8Ax8/4JmfGz4C+MfhvrXhvxBrHjW6uvD9v4i02WyM5WcSKR5yrlWC43dOa/cTWPG/wMv8AWV1H/hc3hPKtlf8Aiorb/wCLqLXPGXwM1WWJ5PjN4T+T/qYrb/4uu6NeNkjPlP5wPCvws1Pw94O0/wDZRi/4ITWV98XLSRdOk8ea9o8z6HczL8hmdwFUqW53+aBiv07/AGj/ANg7UfiP/wAEsdT+AEfw/wBNs9Y0nw8t7p2i+FbFhBBfQHz0SBNzt99FAwTzX3n4i1f4DajErR/GLwkrL3HiK2/+Lq14L8U/AzRG3y/Gjwn/AOFFbf8AxdCrU4xYcrPwh/4JqfsKfHrx/wDsIfGq/wD2oPBPiNPGXiyzWwsYPFGkTRXix2q7I1RZFViuI1xgcjFd5/wblfsx/H/wXr3i/wDae/aY8PeJrLxBfJHomlx+KtOltrhLeEucqsqq2wibg4wcV+z3iXXvgFrWqreyfGPwmSp4/wCKitv/AIuodW1P4AXtxDK3xh8J/Ke3iK1/+Lqo1qfu+X9f16g4yOg8PaNczfC7WmhgaSabRbkJGi5ZmMTAADua/H//AIJmfA740eA/2ef2mtI8b/CXxJo91rHiTUn0m21PQ54JL1Wkn2tEroDIDkYK5zmv2W8O/GH4EaJp62i/Gbwn8q4/5GK2/wDi6xfFvj74F6/eLcN8ZvCeA2f+Ritv/i6whUSrczKt7p+Lv7JfwN+Negf8EPfiP8Odb+Eniaz8QXlxemz0O60OeO7myRjZCyb2z7DmvoL9hb4UfELQv+CPlr4D8S+AtYsteXwrcx/2HeabLHdbz0XymUPk+mK/Q+48U/AqWeORvjJ4T+X/AKmO2/8AjlOXxb8C01NLz/hcvhPcv/Ux23/xddH1iNieVn4N+Hv2aPj7p3/BvprHw1u/gf4uh8UTeKJHj8OyeHblb1k/dfMICm8jg847V0fxr/4JZ3/wh/Yn+E37Sv7K3wi1LSfiP8P4bbU9S02y0+QXt/vJEqPEF3s+JW4xnC9K/cLXPGHwM1K6S4f4z+E/l6f8VFbf/F1Da+IfgKl607/GPwkf9lvEVr/8XRHER5V6A4vY4/8A4J9fFu9+LvwS8N/EHxT4T1PRdS1TToZrzS9YsZLa4t5SoLK0cihlIJ7ivplCCoK15BYeNfgbp2p/a4PjN4TX/uYrb/4uuyg+O/wNjiVD8aPCfA/6GK2/+LrkrSjKV0VHQ66iuT/4X18DP+iz+E//AAorX/4uj/hfXwM/6LP4T/8ACitf/i6xKOsork/+F9fAz/os/hP/AMKK1/8Ai6P+F9fAz/os/hP/AMKK1/8Ai6AOsork/wDhfXwM/wCiz+E//Citf/i6P+F9fAz/AKLP4T/8KK1/+LoA6yiuT/4X18DP+iz+E/8AworX/wCLo/4X18DP+iz+E/8AworX/wCLoA6yiuT/AOF9fAz/AKLP4T/8KK1/+Lo/4X18DP8Aos/hP/worX/4ugDrKK5P/hfXwM/6LP4T/wDCitf/AIuj/hfXwM/6LP4T/wDCitf/AIugDrKK5P8A4X18DP8Aos/hP/worX/4uj/hfXwM/wCiz+E//Citf/i6AOsork/+F9fAz/os/hP/AMKK1/8Ai6P+F9fAz/os/hP/AMKK1/8Ai6AOsork/wDhfXwM/wCiz+E//Citf/i6P+F9fAz/AKLP4T/8KK1/+LoA6yiuT/4X18DP+iz+E/8AworX/wCLo/4X18DP+iz+E/8AworX/wCLoA6yiuT/AOF9fAz/AKLP4T/8KK1/+Lo/4X18DP8Aos/hP/worX/4ugDrKK5P/hfXwM/6LP4T/wDCitf/AIuj/hfXwM/6LP4T/wDCitf/AIugDrKK5P8A4X18DP8Aos/hP/worX/4uj/hfXwM/wCiz+E//Citf/i6AOsork/+F9fAz/os/hP/AMKK1/8Ai6P+F9fAz/os/hP/AMKK1/8Ai6AOsork/wDhfXwM/wCiz+E//Citf/i6P+F9fAz/AKLP4T/8KK1/+LoA6yiuT/4X18DP+iz+E/8AworX/wCLo/4X18DP+iz+E/8AworX/wCLoA6yiuT/AOF9fAz/AKLP4T/8KK1/+Lo/4X18DP8Aos/hP/worX/4ugDrKK5P/hfXwM/6LP4T/wDCitf/AIuj/hfXwM/6LP4T/wDCitf/AIugDrKK5P8A4X18DP8Aos/hP/worX/4uj/hfXwM/wCiz+E//Citf/i6AOsork/+F9fAz/os/hP/AMKK1/8Ai6P+F9fAz/os/hP/AMKK1/8Ai6AOsorlofjl8FLltlv8YfC0jAZKp4gtjgev3629G8UeGfEaNJ4e8RWN8q8s1neJKB/3yTQBeooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAq6L/yDY/8AgX/oRq1VXRf+QbH/AMC/9CNWqCpfEwr85/2j7s2/7bHxaUd9R0f/ANM1nX6MV+af7Vt2bb9t74rKJNub7R//AEzWdeLnz5cHF/3l+TP0bwvh7TPqsf8Ap0//AEuBT/tM+lH9pn0rnP7Ub/ntR/ajf89q+S9oz91+qHR/2mfSvjHSf2d/gL8ev+Cg3xIk+NXwh0DxQLTRbM2q61pqXHlZlmzt3Divqv8AtRv+e1YWkeBvAug+NNQ+IekaFDBrWrQpFqGoKzb50UkqDk44LHoB1rWliHT5rXu1+qODHZR9clSuk4xldpq91ZrbXufPf7an7DP7GPgT9nTXvFngf9mLwXpOqWSwSWl/Y6DFHLE3nxjKsBkHmsX9ueb4TT/BP4Mx/HAaT/wi661YDVf7c2/ZfL3JnzN3GMZ619TeM/Dvhj4h+HLjwj4002PUNNu1UXFrMxCuAwYfdIPUA9e1cr8Xf2cPgJ8efBNp8OPi78OrLXNEsWVrPT7qWVUiK9MFHU8fWtqeNty87bs3+XQ8/GcOylGr9WjCPNGKStZNqTetltbTqeF2sn/BB2ERyrB8BFdcEN/oWQfWuu/a6+PPga5/ZItvAv7KnibSbh/GEiaF4Vl0ELNbx7lIygQ4wu0Diqv/AA6d/wCCcP8A0azoP/gZd/8Ax6vR/CX7K/7OPgSDQbXwl8MLGyj8Muz6DHHNMVs2YgkqGc5OR3zWksVhuZNOTt3t/mc9HJc49nOnKnRpqStenzJ9L/ZXS9vM+R7P/gjZ+1yvwyPw6T/gph4ph0GexaKTw7Hpsv2XY6ndFt+042nJHSvRf2bfidrFv+xV8QP2cvH+s/aPEHw5tX028eZsPJbg5ik2nlQY9nrwa+uxqZAwJq4y6+B3wavNd1zxLceB7Nr7xLai3124Ejg3kYCjawDY6KvIAPFT/aEqkbVddU1ZJa/h0NI8J0cHUU8FpdOMuaUpJprS127WdnpbS55D+xx/wUC/Yy8G/sxeC/DHif8Aag8E2N/ZaDbxXVnda9CkkLhACrAtwRWH8Pfjj4W/bn/bys/HHwauP7Q8J/DOxeC58RQNmC6vJFb93Ew4dQkynIPXIre/4dO/8E4f+jWdB/8AAy7/APj1e3fDL4dfDj4NeF7fwX8LvCdjoml2q7YbWyj2gDPcnLN9SSadTFYWPNKmpXd97WV/QnC5PnlRUqWLdNU4WfuczcuXbdJJX10PRP7TPpR/aZ9K5z+1G/57Uf2o3/PauH2jPpfqh0f9pn0ry2b9l/8AZz/aP/aJ1a4+OfwT8N+LGstLsVszrulx3HlbjJkLuBxnA/Kuv/tRv+e1T/s+zfaPj7r0hbd/xLrD+ctezw/Lmzamn5/+ks+P49w/s+E8Q/8AD/6XE9r8Df8ABIb/AIJlX2hQz3X/AAT/APhbKzRgln8H25z/AOO1s/8ADnr/AIJg/wDSPb4U/wDhG23/AMTX1J8JreB/C8JaJf8AVjtXUfZLb/ngv5V9rOcVJqx/OK2PjP8A4c9f8Ewf+ke3wp/8I22/+Jo/4c9f8Ewf+ke3wp/8I22/+Jr7M+yW3/PBfyo+yW3/ADwX8qn2kewz4z/4c9f8Ewf+ke3wp/8ACNtv/iaP+HPX/BMH/pHt8Kf/AAjbb/4mvsz7Jbf88F/Kj7Jbf88F/Kj2kewHxn/w56/4Jg/9I9vhT/4Rtt/8TR/w56/4Jg/9I9vhT/4Rtt/8TX2Z9ktv+eC/lR9ktv8Angv5Ue0j2A+M/wDhz1/wTB/6R7fCn/wjbb/4mj/hz1/wTB/6R7fCn/wjbb/4mvsz7Jbf88F/Kj7Jbf8APBfyo9pHsB8Z/wDDnr/gmD/0j2+FP/hG23/xNH/Dnr/gmD/0j2+FP/hG23/xNfZn2S2/54L+VH2S2/54L+VHtI9gPjP/AIc9f8Ewf+ke3wp/8I22/wDiaP8Ahz1/wTB/6R7fCn/wjbb/AOJr7M+yW3/PBfyo+yW3/PBfyo9pHsB8Z/8ADnr/AIJg/wDSPb4U/wDhG23/AMTR/wAOev8AgmD/ANI9vhT/AOEbbf8AxNfZn2S2/wCeC/lR9ktv+eC/lR7SPYD4z/4c9f8ABMH/AKR7fCn/AMI22/8AiaP+HPX/AATB/wCke3wp/wDCNtv/AImvsz7Jbf8APBfyo+yW3/PBfyo9pHsB8Z/8Oev+CYP/AEj2+FP/AIRtt/8AE0f8Oev+CYP/AEj2+FP/AIRtt/8AE19mfZLb/ngv5UfZLb/ngv5Ue0j2A+M/+HPX/BMH/pHt8Kf/AAjbb/4mj/hz1/wTB/6R7fCn/wAI22/+Jr7M+yW3/PBfyo+yW3/PBfyo9pHsB8Z/8Oev+CYP/SPb4U/+Ebbf/E0f8Oev+CYP/SPb4U/+Ebbf/E19mfZLb/ngv5UfZLb/AJ4L+VHtI9gPjP8A4c9f8Ewf+ke3wp/8I22/+Jo/4c9f8Ewf+ke3wp/8I22/+Jr7M+yW3/PBfyo+yW3/ADwX8qPaR7AfGf8Aw56/4Jg/9I9vhT/4Rtt/8TR/w56/4Jg/9I9vhT/4Rtt/8TX2Z9ktv+eC/lR9ktv+eC/lR7SPYD4z/wCHPX/BMH/pHt8Kf/CNtv8A4mj/AIc9f8Ewf+ke3wp/8I22/wDia+zPslt/zwX8qPslt/zwX8qPaR7AfGf/AA56/wCCYP8A0j2+FP8A4Rtt/wDE0f8ADnr/AIJg/wDSPb4U/wDhG23/AMTX2Z9ktv8Angv5UfZLb/ngv5Ue0j2A+M/+HPX/AATB/wCke3wp/wDCNtv/AImj/hz1/wAEwf8ApHt8Kf8Awjbb/wCJr7M+yW3/ADwX8qPslt/zwX8qPaR7AfGf/Dnr/gmD/wBI9vhT/wCEbbf/ABNH/Dnr/gmD/wBI9vhT/wCEbbf/ABNfZn2S2/54L+VH2S2/54L+VHtI9gPjP/hz1/wTB/6R7fCn/wAI22/+Jo/4c9f8Ewf+ke3wp/8ACNtv/ia+zPslt/zwX8qPslt/zwX8qPaR7AfGf/Dnr/gmD/0j2+FP/hG23/xNH/Dnr/gmD/0j2+FP/hG23/xNfZn2S2/54L+VH2S2/wCeC/lR7SPYD4z/AOHPX/BMH/pHt8Kf/CNtv/iaP+HPX/BMH/pHt8Kf/CNtv/ia+zPslt/zwX8qPslt/wA8F/Kj2kewHxn/AMOev+CYP/SPb4U/+Ebbf/E0f8Oev+CYP/SPb4U/+Ebbf/E19mfZLb/ngv5UfZLb/ngv5Ue0j2A+M/8Ahz1/wTB/6R7fCn/wjbb/AOJo/wCHPX/BMH/pHt8Kf/CNtv8A4mvsz7Jbf88F/Kj7Jbf88F/Kj2kewHxn/wAOev8AgmD/ANI9vhT/AOEbbf8AxNH/AA56/wCCYP8A0j2+FP8A4Rtt/wDE19mfZLb/AJ4L+VH2S2/54L+VHtI9gPjP/hz1/wAEwf8ApHt8Kf8Awjbb/wCJo/4c9f8ABMH/AKR7fCn/AMI22/8Aia+zPslt/wA8F/Kj7Jbf88F/Kj2kewHxn/w56/4Jg/8ASPb4U/8AhG23/wATR/w56/4Jg/8ASPb4U/8AhG23/wATX2Z9ktv+eC/lR9ktv+eC/lR7SPYD4z/4c9f8Ewf+ke3wp/8ACNtv/iaP+HPX/BMH/pHt8Kf/AAjbb/4mvsz7Jbf88F/Kj7Jbf88F/Kj2kewHxn/w56/4Jg/9I9vhT/4Rtt/8TR/w56/4Jg/9I9vhT/4Rtt/8TX2Z9ktv+eC/lR9ktv8Angv5Ue0j2A+M/wDhz1/wTB/6R7fCn/wjbb/4mj/hz1/wTB/6R7fCn/wjbb/4mvsz7Jbf88F/Kj7Jbf8APBfyo9pHsB8Z/wDDnr/gmD/0j2+FP/hG23/xNH/Dnr/gmD/0j2+FP/hG23/xNfZn2S2/54L+VH2S2/54L+VHtI9gPjP/AIc9f8Ewf+ke3wp/8I22/wDiaP8Ahz1/wTB/6R7fCn/wjbb/AOJr7M+yW3/PBfyo+yW3/PBfyo9pHsB8Z/8ADnr/AIJg/wDSPb4U/wDhG23/AMTR/wAOev8AgmD/ANI9vhT/AOEbbf8AxNfZn2S2/wCeC/lR9ktv+eC/lR7SPYD4z/4c9f8ABMH/AKR7fCn/AMI22/8AiaP+HPX/AATB/wCke3wp/wDCNtv/AImvsz7Jbf8APBfyo+yW3/PBfyo9pHsB8Z/8Oev+CYP/AEj2+FP/AIRtt/8AE0f8Oev+CYP/AEj2+FP/AIRtt/8AE19mfZLb/ngv5UfZLb/ngv5Ue0j2A+M/+HPX/BMH/pHt8Kf/AAjbb/4mj/hz1/wTB/6R7fCn/wAI22/+Jr7M+yW3/PBfyo+yW3/PBfyo9pHsB8Z/8Oev+CYP/SPb4U/+Ebbf/E0f8Oev+CYP/SPb4U/+Ebbf/E19mfZLb/ngv5UfZLb/AJ4L+VHtI9gPjP8A4c9f8Ewf+ke3wp/8I22/+Jo/4c9f8Ewf+ke3wp/8I22/+Jr7M+yW3/PBfyo+yW3/ADwX8qPaR7AfGf8Aw56/4Jg/9I9vhT/4Rtt/8TR/w56/4Jg/9I9vhT/4Rtt/8TX2Z9ktv+eC/lR9ktv+eC/lR7SPYD4z/wCHPX/BMH/pHt8Kf/CNtv8A4mj/AIc9f8Ewf+ke3wp/8I22/wDia+zPslt/zwX8qPslt/zwX8qPaR7AfGf/AA56/wCCYP8A0j2+FP8A4Rtt/wDE0f8ADnr/AIJg/wDSPb4U/wDhG23/AMTX2Z9ktv8Angv5UfZLb/ngv5Ue0j2A+M/+HPX/AATB/wCke3wp/wDCNtv/AImj/hz1/wAEwf8ApHt8Kf8Awjbb/wCJr7M+yW3/ADwX8qPslt/zwX8qPaR7AfGf/Dnr/gmD/wBI9vhT/wCEbbf/ABNH/Dnr/gmD/wBI9vhT/wCEbbf/ABNfZn2S2/54L+VH2S2/54L+VHtI9gPjP/hz1/wTB/6R7fCn/wAI22/+Jo/4c9f8Ewf+ke3wp/8ACNtv/ia+zPslt/zwX8qPslt/zwX8qPaR7AfGf/Dnr/gmD/0j2+FP/hG23/xNH/Dnr/gmD/0j2+FP/hG23/xNfZn2S2/54L+VH2S2/wCeC/lR7SPYD45tP+CRn/BNfTVmOj/sG/DOxknhaJ57PwnbxybTjIBC+wqjqH/BOfS/hHp7eI/2NviHr3w78RWe6SxtbW+ZtJumxxFc2i7FmT0yw2k55r7U+yW3/PBfypkum2Uww1utLnj1QHg37Ev7XWs/H3w1q/hT4naEmjePPBeorpfjTR1fPlTlS0U4GBtWeILMq44WQDJxmvfkdZF3Ia+M/wCwLv4b/wDBWfVtMsAkdp4y+FsmtTKkigvLbXVrahivU4U4yRx0r658K3zXNltduVpzjG14iNWiiishhRRRQAUUUUAFFFFABRRRQAUUUUAFFFFABRRRQAUUUUAVdF/5Bsf/AAL/ANCNWqq6L/yDY/8AgX/oRq1QVL4mFfmD+2Tei1/bk+Ka/wB680f/ANM9nX6fV+V37ct59l/bq+KC+t1pH/pns68DiN8uAT/vL8mfqHhFD2nE1SP/AE6l/wClwOV/tcetH9rj1rnv7W96P7W96+I9of0h9TR0P9rj1o/tcetc9/a3vR/a3vR7QPqaOh/tcetH9rj1rnv7W96P7W96PaB9TR0P9rj1o/tcetc9/a3vR/a3vR7QPqaOh/tcetH9rj1rnv7W96P7W96PaB9TR0P9rj1o/tcetc9/a3vR/a3vR7QPqaOh/tcetH9rj1rnv7W96P7W96PaB9TR0P8Aa49a3f2ZJxcfHHXpB/z4WP8AOSuB/tb3rtP2S5/tHxl16T/pzsh+sle/wxLmzqn/ANvf+ks+D8S8P7Pg3EP/AAf+lxP0++En/IrQ/wDXMV1Vcr8JP+RWh/65iuqr7qp8bP5Yj8IUUUVAwooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooA+Nfi3/AMpqvCJx/wA20ax/6fbOvqjwOxaNjivlj4t/8pqPCP8A2bPq/wD6fbOvqfwJ/qWrT/l2B0VFFFZgFFFFABRRRQAUUUUAFFFFABRRRQAUUUUAFFFFABRRRQBV0X/kGx/8C/8AQjVqqui/8g2P/gX/AKEatUFS+JhXwH+27+wp8b/Gf7RHi744+EfEPhwaXr7WMkNreSTidPJsYLdgdqFeWiYjnoRX35Wfr/h61163MFyoPGPrXPiMJh8bBU66vG9+x7GRcQZlw5jHisDJRm4uN2k9G09n5pH5F3H7NHx7gmaEvoLbTjIkm/8AiKb/AMM3fHr/AKgP/fyb/wCJr9SpPgN4Ukcu1nHk/wCyKb/woPwl/wA+Uf8A3yKx/sHIf5H/AOBM+r/4itxt/wA/o/8AgEf8j8t/+Gbvj1/1Af8Av5N/8TR/wzd8ev8AqA/9/Jv/AImv1I/4UH4S/wCfKP8A75FH/Cg/CX/PlH/3yKP7ByD+R/8AgTD/AIitxt/z+j/4BH/I/Lf/AIZu+PX/AFAf+/k3/wATR/wzd8ev+oD/AN/Jv/ia/Uj/AIUH4S/58o/++RR/woPwl/z5R/8AfIo/sHIP5H/4Ew/4itxt/wA/o/8AgEf8j8t/+Gbvj1/1Af8Av5N/8TXF+DPCPxp8Y/F7xt8JIdJ0mCfwWumm4u5WmEdz9sgaZdh287QuD05r9fP+FB+Ev+fKP/vkV5H4MtPg942/aK+KPwY8P+EVtda8CroZ17UPLA+2fbbR5oOcc7EUr1OM0Lh/IW/gf3sT8VuNtP30f/AI/wCR8JD9m/48sMg6D/38m/8AiaX/AIZu+PX/AFAf+/k3/wATX6faV8BfCr2i77GP/vkGrX/Cg/CX/PlH/wB8ij+wMh/kf/gTH/xFbjb/AJ/R/wDAI/5H5b/8M3fHr/qA/wDfyb/4mj/hm749f9QH/v5N/wDE1+pH/Cg/CX/PlH/3yKP+FB+Ev+fKP/vkUf2DkH8j/wDAmH/EVuNv+f0f/AI/5H5b/wDDN3x6/wCoD/38m/8AiaP+Gbvj1/1Af+/k3/xNfqR/woPwl/z5R/8AfIo/4UH4S/58o/8AvkUf2DkH8j/8CYf8RW42/wCf0f8AwCP+R+W//DN3x6/6gP8A38m/+Jr0D9mL4JfEPwF47v8AW/Fgs5G1COCONbEuQuwt13Af3q/Qj/hQfhL/AJ8o/wDvkVJafA3wxZzrcQ2kYZefuiujCZXk+BxCrUYtSW2re6seZnHH/FGe5fLBYypF05WulFLZprVeaNL4UI8fhiFXXB8sV1FVtL0yDS7YW0A4FWa6JPmk2fGhRRRUgFFFFABRRRQAUUUUAFFFFABRRRQAUUUUAFFFFABRRRQAUUUUAFFFFABRRRQAUUUUAFFFFABRRRQAUUUUAFFFFABRRRQAUUUUAFFFFABRRRQAUUUUAFFFFABRRRQAUUUUAFFFFABRRRQAUUUUAFFFFABRRRQAUUUUAFFFFABRRRQAUUUUAFFFFABRRRQAUUUUAFFFFABRRRQAUUUUAFFFFABRRRQAUUUUAFFFFAHxt8W/+U1HhH/s2fV//T7Z19T+BP8AUtXyx8W/+U1HhH/s2fV//T7Z19T+BP8AUtWn/LsDoqKKKzAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigCrov/INj/4F/wChGrVVdF/5Bsf/AAL/ANCNWqCpfEwooooJCiiigAooooAKKKKACvHPCnxA+E/ib9o34peAPBOhrbeLPDa6GfGd8EIN2Li0eS0ye+2IMPbNex1454V1T4JXv7RnxR0v4ewKvjazXQ/+E/cKMuHtHNjk98Q76qHxAeqaBHMloPNOeKv1n6As4tB53XFaFEviAKKKKkAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKAPjb4t/8pqPCP/Zs+r/+n2zr6n8Cf6lq+WPi3/ymo8I/9mz6v/6fbOvqfwJ/qWrT/l2B0VFFFZgFFFFABRRRQAUUUUAFFFFABRRRQAUUUUAFFFFABRRRQBV0X/kGx/8AAv8A0I1aqrov/INj/wCBf+hGrVBUviYUUUUEhRRRQAUUUUAFFFFABXjnhXR/gpYftGfFHWPh9cq3jW+XQ/8AhPowwzGEtHFjkdsw7q9jrxvwp8PPhR4W/aN+KXxD8F68tz4r8TLoQ8ZWIck2Yt7R47TjtviLHjriqh8QHqugNObQed6VfqhoEksloPNXHFX6JfEAUUUVIBRRRQAUUUUAFFFFABRRRQAUUUUAFFFFABRRRQAUUUUAFFFFABRRRQAUUUUAFFFFABRRSOcIT7UAcr8UPjv8EfghZwah8Z/jD4X8JW90xW2m8Ta9b2KSsOymZ1DH6V8MftCf8Fbfiz4E/wCCsfwr/Y9+F174P1j4c+OtFa9u9YhiNxPJ9zDQzxy+XtO7+6a+I/Gfwa8B/wDBVv8A4LrfFb4U/tmzahrXgn4caOiaP4Vi1i4gt3JMmGxG64YEgkjkgAHivJfFv7KXgf8AYi/4LtfDfwx8HNb1mXw7caW1xpGjapqk13/ZaJGgMUbzOzBCecZralRlUt2JcrH9Bnjr9pz4AfCaC3uPi/8AGjwp4Tjusi3k8SeIraxWUjqFMzru6jp61rS/Ezwv4g+GF98SPhv4q0vW7FdKuLrT9Q028S5tpyiMQQ8bFWGV7Gv5c/jH+2D+zD8cP+Ci/wAVtc/4KQfAz4nfFLQdE1B9L8K+F/B9xKLfT1jeT946pPHtY5XkZzt56V9Yf8G/P7WHiTT/ABp8bvgj4I8MeNNL+Fsmj3F74J0vxlHL52moYkUwguzDGd5wGPXPWqdG8+WLDm01P0M/4I2f8FPfjV+3fpvxMuPjxYeH7WTwj40n0rSRodk8AeBJZlBffI+Wwi8jHfivrCb9r39lW08ZL8ONR/aR8B2viNphD/wj9x4us0vfMPRPJMm/cfTGa/Af9iv9orxt+zL/AME8P2rvjF8PJmh1rTvGGoNYzL/yyk8y6Ib8MV8geEfjJ/wT81v9l2TXPE37L/x01n4+3SveW/xStZp3hS8zuQqRc8oGA52Zxn0FEqPLFa6hfU/pl/4Kkf8ABR/4ff8ABN39mTVfjZrb6bqWsRxgaN4fn1iG3mvHJxlFc7pAO4UVpf8ABPf9uXwb+2B+zJ4W+Kd38QfC2oeIL7RobjxHZ+HtSikSxnZcsjIrs0WPRjmvxb/bDvtM/bV/4IJaD8cv2lvCF9deP/BNqtrpepajPPFMgSQx7mTIDkqOrA9c1z/xxi8Nf8E6P+CKmj+Mf2XdEv8Aw3rHxOs7GDxNq1hqMzTOJvlkZCzny2Ic4245qo4aWvM9lcOY/oK8LftZfst+OPGH/CvPBn7SHgTVvEAkZDoem+LbOe8DDqvkpIXyMHIxxiuw8T+LvCvgnQLnxV4y8S2Gk6XZxmS81HUrxIIIFHVndyFUe5NfyReJvjZ+wv4W+EfhXxR+yB+yp8cvDPxv0q6tbq68f3jXDQalN5gMpfFw/wArKW6Lz36mvvH/AIKs/tUfF/8Aa++HX7Lf7Mfi/wAR6joOkfFC8sf+EwW1uJLeaZisQZX2kHb+8bKnj1FZqjKUW1qPmP0E/wCCuX/BXNP2WP2Q4/j7+xL8Tvh5421CPxTZaZfNDqUWq28Ky3EMbBvs0w2vtkOMn04r60/Z1+M1z8V/g14U8c6+LddR1rw/Z3t4lqu2NZJYUdgoJJAyxxyeK/AX/gt9/wAEo/2T/wBjT9l/wt8RP2VLTWvDKW/iaxXXNJXxBd3FrqzvLCvmPHLKyhgSGyAOgr9gf2Atcvrr4IeA4JXYbfCmnD/yWjrSOHlytyFzdj6W8TfGv4OeC/FFl4H8Y/Fbw3pOtakudP0jUtbggurkf9M4ncM/4A1l+D/2ov2afiH4vm+H3gH9oTwTrevW5YT6LpPiq0uLuMjqGhjkLjHfIr8a/wDg4a+D8Hx3/wCCsX7P3wyvPEmoaVbagqC7utLu3gn8rem9VkQhlyOMgg14f/wUd/Yu+BX/AATa/bC+Afxa/Yzt9c8K6lrviyKz12Rdeubg3xKylndpZGYk7eR0rOnRlU1Q72P6J/HPxE8AfDDw1P4z+JPjfSfD+j2v/HxqmtajHa28X+9JIyqvQ9TWP8Lv2hvgH8b1uG+DHxt8J+Lhaf8AH0fDPiK2vvJ/3vJdtv41+LH7d3izV/8AgpJ/wWD8B/sDfHDxVq1v8MLHwy2qa1oNjfSW8erOPJ3JIY2UkHc3fjPFcN+29+zt8MP+CLP7WPwZ+NH7AX9r+FdJ8TeKodG8R+GI9auLiC+SRwhkKzO24/vPw25HNV9XqczTFzI/ez4lfGP4SfBnRl8R/F74oeHvCunySBEvvEWswWULN/dDzMq54PGe1fB/7eP/AAV6+InwQ/bM+BPwg/Zy17wT4k8DfE7UvJ1TWoT9u3x7ohmCeGbZ0c84btXw7+1Lo+mf8FTv+C3+j/s0ftQalqFz8O/DXgiPUf8AhF7bVJ4ILp2S2JDiNl+bMh+bqOea8n/bN/Ya+E37D3/BVr4D2n7PWpaxZ+FNQ1aFNO8KX2sT3cGmMkqFmiMzuyhgyjAwPkojh5ylbzDmR/Qp4x/aQ+CPwu0e31z4ufFnw14VtbhgsV34i1y3sYnbGdoaZ1BPtmt7wD8TPhz8VfD0fi34YePdG8R6VMxWHUtC1OK7t3I6gSRMyn86/mH/AG1f2yvgd8Yf+CoXi/w3/wAFAfhN8RPiJ4B8G2sdloHgzwVM4RXMcbmWVUljwcu3zZJ5A6V79/wQJ/a4tvBv7fniz4R/s4fDnx94X+DGuab9r0Xw740jlJ0yZCchSzuAPn/vZOOaJUf3nLFhfTU/oWorH8I+IYte09Z1fcdvX1rYrBrldmUFFFFIAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooA+Nvi3/AMpqPCP/AGbPq/8A6fbOvqfwJ/qWr5Y+Lf8Aymo8I/8AZs+r/wDp9s6+p/An+patP+XYHRUUUVmAUUUUAFFFFABRRRQAUUUUAFFFFABRRRQAUUUUAFFFFAFXRf8AkGx/8C/9CNWqq6L/AMg2P/gX/oRq1QVL4mFFFFBIUUUUAFFFFABRRRQAV4x4R+D3gjwJ+0n8VvjD4e8TLea544XQR4g00SA/YPsdnJDBkZ43oxbtnFez14t4O+CVv8Pf2mPix8bIvGP2+Tx+ugiTR/ODf2Z9hs5IB8v8Pmb93viqh8QHregzSS2gLrt4zir1UNAn860DbMd6v0S+IAoooqQCiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACgjIwaKKAPx7/AOCln/BJr9vP4Zftx3n/AAUD/wCCXPjLw3BrfiS18jxR4d8S4itnPzfvS27EmQ3C4GCuc814r8Lf+CYX/BSvVf8Agor4J/bc/an+IXhnxItvZgeILfT7kW6aUwRV8m3h3PvU4yWBGSOlfvFq+j22sW/kXC1ycnwb0iSVpDCuD2rpo1IxWpMo3Pxu+O//AATS/wCCjn7PX7Tfir9q7/glx4u8Jsnj6Qy+JvDPiyMRwQS5ZvNUlsOTvPYY2+9e5/8ABP39lT9uDwf4I8V+If20vj9ceKfF3iyOX7P4ft7xjpui5j2CKHLMMEjdkAY3niv0xsvhzpltY/YxGoXpjFUbT4SaXbX32oIv3s/StViI8zZPKz8pP2KP+CRXxb+Hn7Pfxq+AP7VWk6XFp/xO8QXc9i2m6gt1i3leYhmGBtbEg4rxDSv2Bv8Agvj+zx4ZT9k/4J/GbwVH8L7e48jSvGDSeVq1raM3LIN+5WUE4UHkj3r93r3wJp93t3gHaeKg1b4caZqVqIJI1bb04pfWI6XDlZ+bv7SP/BOr4tftEf8ABObVP2T7D4p3XiHxlNosSR+IvEFyWN3eLtJZ2YkhSc9ScV4X8Kv+CY/7d/x1/Ye1T9h7/goKfB+i/wBj29ungHWPC8i3L2skWfLeXBG4ghDnIzX7KeH/AIa6dozFkVabqvw107ULz7WYl3U/rEXIOVn4f+Gv2P8A/g4HvbzQf2f/AB98e/Cfhf4faLdRRL4u8MXG3V5rOI5UHa4bLABSM8A98V9Bf8FJP+CTnxE/bK+DPhnW/gZ8QmsfiZ8P5o7rw3rmpyFnupEAyryZBQsyr8/OPQ1+nWp/CrSr6JVKLkfpVzw/8P8ATdGh8sIPpR9Yjy2DlPwB/ab/AOCbn/Bdb9tj4SaV4O/ab+I/gKKTwrrlvNp/hrRroQxalGrx5knn3EFlClgpU5Kjnmv12/Y68Jax4I8BeF/BmuQRre6PoNnZ3nkvuQSRwojYPcZU896961f4T6TqN0Z/KXmrHh/4b6dodz59vEqn2qfbU+Vj5WfAf/BSz/gnT+0n+0F/wUc+D/7Vnw707R5PCPgqPGuS3eqiK4XlT8keCX6HvXAf8FUf+CcX7R/7aPxG+Dfi74H6Xo81j4F8WrqHiE6nqgt2WEJKCYwQd5+ccV+rGqaTDqdp9kl4Wsyx8D2VpE0afKGrOnVUIWG1dn5U/wDBRD/gl7+0H8Ufi5oX7an7DnjSw0P4t+EbU28Fnqi4tNUj+X93K+4bVzGAcg5zXnPwN/4Jy/8ABQz9rH9orw58bv8Agrl4x8Lix+H832jwv4Z8GODDNc84lfDEKQSO3IXFfs1ZeAtOtizBF+brxWZf/CTS7u7+0GNeGz9a09tGUrsXKfkp/wAFD/8AglX+2Nr/AO1Do/7ff/BODxZo2meNtPsxZalo+uHyra8hAUKzvuAIAjUbMc9c8V4zcf8ABMT/AIKv/GX9sD4W/tgftU/ETwnrF3oF5Ede8P6TdC2tdJijdWzAu5hKzfMCQF+4tfvVaeCtMttP+xeWuCPSsO5+D2kTXDS+SvNCrQcrsXL2PyB/au/4Joft0eAv2oL/APbd/wCCYnirw5b+I/Edqtt4q8PeJowlrOFVQJdxYBj8iDbj3zXqn/BOL9mr/goT4d8U638Xf2+/j1/aWtawypZ+C9DumOk6aq7sui7ipLZHQDGO9fp1pvw30uytmt1jVQfaqKfCLSkvvtSxr97NUsRHmuHKyT4T6fJZaPGh/hWuwqvpmmwaZbiCEVYrjlLmlc0CiiipAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKq6xrWj+HdNm1nxBq1rY2duhe4ury4WKONQMkszEAD3JoA+QPi3/ymo8I/wDZs+r/APp9s6+p/An+pavkn4peK/C15/wWJ8I+K7TxJp8ult+zFq8q6lHeI1uU/t2z+bzAduPfOK+qfhL4j8PeK9Ej17wtr1nqVjcLut7zT7pJoZV9VdCVI+hrT/l2B19FFFZgFFFFABRRRQAUUUUAFFFFABRRRQAUUUUAFFFFABRRRQBV0X/kGx/8C/8AQjVqqui/8g2P/gX/AKEatUFS+JhRRRQSFFFFABRRRQAUUUUAFeKeDPgx4l8AftOfFr4zav4ra803x4vh8aTpTSFhp/2KzkhlwucL5jMG464r2uvEvBPwv+Jvg/8Aaf8Ai58T/F/iP7V4Z8XL4fHhPTfMz9iNrZSRXXGeN8jK3QZxVQ+ID1/QZlmtAypir1UdBliltAYkxV6iXxAFFFFSAUUUUAFFFFABRRRQAUUUUAFFFFABRRRQAUUUUAFFFee/tX/FDxJ8Fv2cPGXxW8HxWr6poOhy3dit5GXiMi4xuUEEj8RQB6FRXzVpl5/wUL1LTbfUV+M3wwUXECSBT4LvPl3AHH/H171Pj/goZ/0Wr4X/APhFXn/yVQB9HUV844/4KGf9Fq+F/wD4RV5/8lUY/wCChn/Ravhf/wCEVef/ACVQB9HUV844/wCChn/Ravhf/wCEVef/ACVRj/goZ/0Wr4X/APhFXn/yVQB9HUV844/4KGf9Fq+F/wD4RV5/8lUY/wCChn/Ravhf/wCEVef/ACVQB9HUV844/wCChn/Ravhf/wCEVef/ACVRj/goZ/0Wr4X/APhFXn/yVQB9HUV844/4KGf9Fq+F/wD4RV5/8lUY/wCChn/Ravhf/wCEVef/ACVQB9HUV844/wCChn/Ravhf/wCEVef/ACVRj/goZ/0Wr4X/APhFXn/yVQB9HUV8WeBf2mP25vGkGqSL43+Hlu2la9eaXJu8L3R8xreUxmQf6TwGIyB2rc/4XL+3T/0Ub4df+Erdf/JNcH9oUez+7/gn1H+qeYWV5wXzf/yJ9b0V8kf8Ll/bp/6KN8Ov/CVuv/kmj/hcv7dP/RRvh1/4St1/8k0f2hR7P7l/mH+qeP8A+fkPvf8A8ifW9FfJH/C5f26f+ijfDr/wlbr/AOSaP+Fy/t0/9FG+HX/hK3X/AMk0f2hR7P7l/mH+qeP/AOfkPvf/AMifW9FfJH/C5f26f+ijfDr/AMJW6/8Akmj/AIXL+3T/ANFG+HX/AISt1/8AJNH9oUez+5f5h/qnj/8An5D73/8AIn1vRXyR/wALl/bp/wCijfDr/wAJW6/+SaP+Fy/t0/8ARRvh1/4St1/8k0f2hR7P7l/mH+qeP/5+Q+9//In1vRXyR/wuX9un/oo3w6/8JW6/+SaP+Fy/t0/9FG+HX/hK3X/yTR/aFHs/uX+Yf6p4/wD5+Q+9/wDyJ9b0V8kf8Ll/bp/6KN8Ov/CVuv8A5Jo/4XL+3T/0Ub4df+Erdf8AyTR/aFHs/uX+Yf6p4/8A5+Q+9/8AyJ9b0V8i+Hf2g/2zr/x6/gDUPGngNpv7I/tBbqPw3cqu3zRHswbjOcnOc11f/Cwf2xf+igeBP/CbuP8A4/X0GFyyvisPGtFpKSurt3/JnyWOrRy/GTw09XF2bWq2T0vbv2Po+ivnD/hYP7Yv/RQPAn/hN3H/AMfo/wCFg/ti/wDRQPAn/hN3H/x+t/7FxP8ANH73/kcn9pUf5Zfcv8z6Por5w/4WD+2L/wBFA8Cf+E3cf/H6P+Fg/ti/9FA8Cf8AhN3H/wAfo/sXE/zR+9/5B/aVH+WX3L/M+j6K+cP+Fg/ti/8ARQPAn/hN3H/x+j/hYP7Yv/RQPAn/AITdx/8AH6P7FxP80fvf+Qf2lR/ll9y/zPo+ivnD/hYP7Yv/AEUDwJ/4Tdx/8fo/4WD+2L/0UDwJ/wCE3cf/AB+j+xcT/NH73/kH9pUf5Zfcv8z6Por5w/4WD+2L/wBFA8Cf+E3cf/H6P+Fg/ti/9FA8Cf8AhN3H/wAfo/sXE/zR+9/5B/aVH+WX3L/M+j6K+cP+Fg/ti/8ARQPAn/hN3H/x+j/hYP7Yv/RQPAn/AITdx/8AH6P7FxP80fvf+Qf2lR/ll9y/zPo+ivlvxT+0D+1J8N20rXPFninwbfabca9ZWV5BZ6HPFJ5c0yoxVmmIBAPpX0Z4d8ZaZ4iXdZyK2fRq5cVgK2Ds52d+3l9x0YfFU8RflTVu5sUUUVwnQFFFFABRRRQAUUUUAFFFFABRRRQAUVBqlzJZ6ZcXcQG6KB3Xd0yFJr41+HX7Tn7b3xJ8Faf45sPGfw/s4dShMsVrN4ZuXaNdxABIuACePSueviqeHaUk9e3l/wAOetluTYrNITlSlFKLSfM2t72tZPsz7Ror5I/4XL+3T/0Ub4df+Erdf/JNH/C5f26f+ijfDr/wlbr/AOSaw/tCj2f3L/M9L/VPH/8APyH3v/5E+t6K+SP+Fy/t0/8ARRvh1/4St1/8k0f8Ll/bp/6KN8Ov/CVuv/kmj+0KPZ/cv8w/1Tx//PyH3v8A+RPreivkj/hcv7dP/RRvh1/4St1/8k0f8Ll/bp/6KN8Ov/CVuv8A5Jo/tCj2f3L/ADD/AFTx/wDz8h97/wDkT63or5I/4XL+3T/0Ub4df+Erdf8AyTR/wuX9un/oo3w6/wDCVuv/AJJo/tCj2f3L/MP9U8f/AM/Ife//AJE+t6K+SP8Ahcv7dP8A0Ub4df8AhK3X/wAk0f8AC5f26f8Aoo3w6/8ACVuv/kmj+0KPZ/cv8w/1Tx//AD8h97/+RPreivkj/hcv7dP/AEUb4df+Erdf/JNH/C5f26f+ijfDr/wlbr/5Jo/tCj2f3L/MP9U8f/z8h97/APkT63or5I/4XL+3T/0Ub4df+Erdf/JNO8D/AB+/bJ8T+Ida8L3PjPwIk+i/Z/MuF8O3G2XzkLjA8/IwBg88135f/wAKVZ0qWjSvrorXS6X7nl5xlNfJcKq9aUZJtRtFtu7TfVLTTufWtFfOH/Cwf2xf+igeBP8Awm7j/wCP0f8ACwf2xf8AooHgT/wm7j/4/Xsf2Lif5o/e/wDI+b/tKj/LL7l/mfR9FfOH/Cwf2xf+igeBP/CbuP8A4/R/wsH9sX/ooHgT/wAJu4/+P0f2Lif5o/e/8g/tKj/LL7l/mfR9cv8AG34VeGvjl8IPE3we8XwhtN8TaDd6Zdt5YZo0nhaIuv8AtKGyDxgivGP+Fg/ti/8ARQPAn/hN3H/x+j/hYH7Yv/RQPAn/AITdx/8AH6P7FxP80fvf+Qf2lR/ll9y/zP5dvjB8FP2+fhV/wUlm/wCCfv8Awt3xhHf2+oSeHdFt28RXXGgSyi5SHO7iNolSQoPl3Adetf08/sE+EfB37PHwL8K/s5/DYO2keE9Hh0+0mkGJJUjUKHfk5YgckkmvGviJ+xR4i+J37Sdv+1h4kuvAn/CaW2jtp6alD4dmBKFlIc5mJ3gKFBz04xXo3gfwn+0n4AO7RfiD4NY+s3h+dv8A2sK6IZPUjTac439X/kS8yp83wv7l/mfXVs7SQq7elSV83jx/+2IowPiB4E/8Ju4/+P0v/Cwf2xf+igeBP/CbuP8A4/XP/YuJ/mj97/yK/tKj/LL7l/mfR9FfOH/Cwf2xf+igeBP/AAm7j/4/R/wsH9sX/ooHgT/wm7j/AOP0f2Lif5o/e/8AIP7So/yy+5f5n0fRXzh/wsH9sX/ooHgT/wAJu4/+P1x/xE/aC/bX8M+OPBfgLSfG/gDz/F+sTWKXU3hq5Zbfy7aSfcVFwCc+Xjr3rKtlVfD0nUlKNl2b/wAi6eOp1aiik9fL/gn1/RXzjj/goZ/0Wr4X/wDhFXn/AMlUY/4KGf8ARavhf/4RV5/8lV5h2n0dRXzjj/goZ/0Wr4X/APhFXn/yVRj/AIKGf9Fq+F//AIRV5/8AJVAH0dRXzjj/AIKGf9Fq+F//AIRV5/8AJVGP+Chn/Ravhf8A+EVef/JVAH0dRXzjj/goZ/0Wr4X/APhFXn/yVRj/AIKGf9Fq+F//AIRV5/8AJVAH0dRXzjj/AIKGf9Fq+F//AIRV5/8AJVGP+Chn/Ravhf8A+EVef/JVAH0dRXzjj/goZ/0Wr4X/APhFXn/yVRj/AIKGf9Fq+F//AIRV5/8AJVAH0Jov/INj/wCBf+hGrVVdF/5Bsf8AwL/0I1aoKl8TCiiigkKKKKACiiigAooooAK8R8D+FvjJo37Ufxe8UePNSabwbrC+Hv8AhBbYtxAYrKRb3A7bpSpr26vD/C138bIf2qfifZePFmk8EzW+iP4HYxt5cLC0YXgB6EmUqeMdKqHxAexaC0LWoMQ7VeqjoPkfZB5FXqJfEAUUUVIBRRRQAUUUUAFFFFABRRRQAUUUUAFFFFABRRRQAV41/wAFDP8AkyT4mf8AYq3H9K9lrxr/AIKGf8mSfEz/ALFW4/pQBL4X1Jh4Z04eb/y4w/8AoAq9/abf89a43w1qr/8ACOaf+8H/AB5Rd/8AYFXf7Vf/AJ6D86AOl/tNv+etH9pt/wA9a5r+1X/56D86P7Vf/noPzoA/Lb/gsR/wU+/4KPf8E4v25fCvxTi8IQX/AMD1tRDNbWMIdLzcYzIJXZQySDBxggctzX6Mfsc/ts/Bf9t74K6X8a/gp4sgvbO+gU3VoJP31nLjmORexGfcc9ah/aH+B3wq/am+Euq/BT41eGrfWNA1eEpc2syg7WwQHXIOGGTg44zX4S+NfB37Xv8AwbY/td/8LD+HU95r3wW8Tagqzx/N5EsJbBjfGVWVQcgnnO3igD+jb+02/wCetH9pt/z1rwf9j/8AbU+Df7bPwY0741fBTxRDfWN5Ev2u3EgMlnNj5opADwQQRzjOM11Xxm+Pfw7/AGf/AIb6p8Wfix4qt9J0PSLdpry8uJAoAAJ2jJGWOOB3NAHSfGn49/Dn9n74bap8V/iv4ttdJ0XSLVp7q6upNuQBwo9STgAe9fkj+yv/AMFmv29/+CkH/BUzTbD9krwgkPwZ0OaS21hNStwIZbUyIGuJJAGYOQuVVT3bpXzP+0L+0V+1x/wcY/tYx/s+fASC80f4O+HdRBvLjLCHyQeZ5jwpY5woPqvNftJ+wr+xZ8Ef2AfgfY/Bb4J6JDbxRxo+q6iUAm1C4A5lcjryWIBzjPWgD6Lj1OYxqZZF3bfm25xml/tNv+etc1/ar/8APQfnR/ar/wDPQfnQB0v9pt/z1o/tNv8AnrXNf2q//PQfnR/ar/8APQfnQB8/fBHUjFB4tA7/ABA1k/8Ak01dv/azV5R8ItTMQ8VLn/metXP/AJMtXXf2x718i6lm0fvUcJzxjLul+SOp/tZqP7WauW/tj3o/tj3o9qH1JnU/2s1H9rNXLf2x70f2x70e1D6kzqf7Waj+1mrlv7Y96P7Y96Pah9SZ1P8AazUf2s1ct/bHvR/bHvR7UPqTOp/tZqP7WauW/tj3o/tj3o9qH1JnU/2s1H9rNXLf2x70f2x70e1D6kzQ8K6mf+Ggnlzj/ijSP/JsV6X/AGuf+ehrxPwpqYPxxeUt/wAymR/5MivRf7WX+/8A+O1+wZHT9pk9CX939WfzrxY/Y8SYmHaS/wDSUdP/AGuf+eho/tc/89DXMf2sv9//AMdo/tZf7/8A47XqfVz572x0/wDa5/56Gj+1z/z0Ncx/ay/3/wDx2j+1l/v/APjtH1cPbHT/ANrn/noaP7XP/PQ1zH9rL/f/APHaP7WX+/8A+O0fVw9sdP8A2uf+eho/tc/89DXMf2sv9/8A8do/tZf7/wD47R9XD2x0/wDa5/56Gj+1z/z0Ncx/ay/3/wDx2j+1l/v/APjtH1cPbHT/ANrn/noaP7XP/PQ1zH9rL/f/APHaP7WX+/8A+O0fVw9scz+1jqrv8LrfZIdy+ItPZfqJ1r2D9lLUry+tFa5nZunU14F+01frcfDq3jDf8x6x7f8ATZa91/ZG/wCPJfwrx87jyYeC9f0PVymXNKb9P1PoSiiivkT3AooooAKKKKACiiigAooooAKKKKAKmv8A/ICvf+vST/0E18Ifs06k0XwJ8NR/3dP/APZ2r7v1/wD5AV7/ANekn/oJr87v2fNU8r4M+H4/7tmR/wCPtXi5tLlnD0f6H6JwPR9thcSv71P8pnrf9rNR/azVy39se9H9se9eX7U+2+pM6n+1mr4A/wCC3v7Vn/BQD9lfRvCvxa/Zh0e3k8H6VfC48UTQwiWYsPupIrKcRkb+V54Ffa39se9UfFGn6B418PXnhPxVpUF9p2oQNDeWdwoZJYz1Ug9RV068YTTaucuMyupiMPKnCTi3s10Z5P8A8E4f+Cmfwl/4KDfB+28XeFr+Gy8SWsYXXvD0kn722kwMsBnlT257Gvo/+1mr8J/25P2Ifj3/AMEj/j+v7b/7Ddzcr4Pkug2qaXbgkWiFvmikC9YiGwCQAMmv0n/4J2f8FJ/hR+358JYfFPhO+jtfENjCq+INDeQeZbydC4HBKE8g44yBXRWhFR9rT1i/w8meZltepOs8Hi1y1o/dJd1+p9Y/2s1VNc8ZaT4a0i41/X9Rhs7OzhaW5uZ3CpGijJJP0rmdZ8Y6V4e0q41zXdRhtLO0haW6uriQKkSAZLMT0Ar8bv8Agox/wUp+Nv8AwUj+Mi/sIfsHJeSaDNdfZtc1azLf6cA3zMSv3Ylx3JBKmsqEZV5WWy3fY68yrUctopyV5S0jFbtnpv7TP/Bav9pv9pf9tXw9+zd/wTU0r+0NP0jWV/tPUmt1aPUWRtrbmIO2IEnngnjiv1o8Nazr8nh+zfxPFAmoNbqbxbViYxJjnaTg4zXyD/wTG/4Jv/Cr/gnp8K4tP02zt7/xhqMKtr+vNGN7MRkxIeoQE4684FfUX9se9ViK9FtRprRde5OWZZjowdXFS96WvL0j5L9Tqf7Waj+1mrlv7Y96P7Y965/anqfUmdT/AGs1Znwk1Mr8UfHEm7G7+zf/AEQ9ZP8AbHvVX4U6mB8RPGEhb739n/8Aol6+t4NftMyqR/uP/wBKifAeI1D2GR05f9PI/wDpMj2X+1z/AM9DR/a5/wCehrmP7WX+/wD+O0f2sv8Af/8AHa/R/q5+J+2On/tc/wDPQ0f2uf8Anoa5j+1l/v8A/jtH9rL/AH//AB2j6uHtjp/7XP8Az0NH9rn/AJ6GuY/tZf7/AP47R/ay/wB//wAdo+rh7Y6f+1z/AM9DR/a5/wCehrmP7WX+/wD+O0f2sv8Af/8AHaPq4e2On/tc/wDPQ0f2uf8Anoa5j+1l/v8A/jtH9rL/AH//AB2j6uHtjp/7XP8Az0NH9rn/AJ6GuY/tZf7/AP47R/ay/wB//wAdo+rh7Y6f+1z/AM9DXmnxb1Iy/tBfBk7/ALviy9P/AJTbiuk/tZf7/wD47XnfxP1EP8e/hA6v93xTeH/ynz1wZpR5cBN+X6o6sDU5sVFf1sfVH9pt/wA9aP7Tb/nrXNf2q/8Az0H50f2q/wDz0H518QfTnS/2m3/PWodR1TUlsJjpLQtdeWfs63DMEL443Ec4z6Vgf2q//PQfnR/ar/8APQfnQB+PPw2/4Lo/tmfsQ/8ABRvxT8B/+CmXhw2/hHxJru3Rb23tlWHTIiFWOSJwFLxkKM5yQWbjiv2Z8HfETwx8QfDNn4y8FeI7XU9L1C3WeyvrSTdHLGwyGB+h+tfJ/wDwU/8A+Ca/wU/4KVfBebwb4+0+G38Sadbs3hjxAijzbSXkhSepQkkEZH3s1+WP/BPL/gov+0h/wRR/aJb9gL9vdLxvALXhTR9Wm3FbJC2FmiLcGIggkAgfd5oA/oQ/tNv+etH9pt/z1riPCHxD8PePvDFj4z8H65b6hpepW6z2N5bSho5Yz0II4NeH/wDBRP8A4KT/AAV/4J2/BS6+JfxQ1uOTU54WXQNDjmHnX02DgAddowcnB6UAdB/wUd/4KY/BL/gnF8Dbz4p/E3WobjVHjZdB8PRyfv7+fHCgZGFyVycjg8V8lf8ABBH9uX/gpD+3F41+IHxo/aG0K3t/hnql15vhhru3EMkD/Ioig2qN67QSSx67vavhj9jT9jj9p/8A4L4/tRv+2p+2vcXln8MbO6zpunNuEVxGG3C2gDcbMYBYAjDDiv3j+HPhDwh8JvA+mfDj4e6Lb6Xouj2q2+n2NqoVIox2AHqck+5NAHoH9pt/z1o/tNv+etc1/ar/APPQfnR/ar/89B+dAHtui/8AINj/AOBf+hGrVVdF/wCQbH/wL/0I1aoKl8TCiiigkKKKKACiiigAooooAK8x/aO0X4xRW+meO/hDeteSaTcBtU8KsUVdWtj95Y3bGyYcbSzBMbs5OK9OpssYlTYaFoB57eftAfCD4ea9Y+CvHfjex0bU9Qjj+zWuoTiMOznCoGb5SxbjAOc10fjD4ufCj4eNap4/+J3h7Q2vcfY11jWoLXz8/wBzzHG78M1i/EP4BfCT4i3Vvqvj74Y6Dr1xYzLLYzatpMNw0EiNuV0LqdpDAEEdDWJ8R/2bvgd8bWsZfjR8DfCviptJ50tvEPh+3vDaYOR5fmo2zn0xWnLzapgdt4s+LXwq8A6bb6z46+Jnh/RbO7/49brVtagto5v9xpHAbqOhpfEHxX+F3hLw9B4t8VfEnQNM0m6/49tT1DWIIbeb/dkdgrfga4b4gfs7/BL40aLZ+FPjF8D/AAr4m0vTf+Qfp+v+H7e6htun+rSRGC9B0A6U7xf+z58GfiZ4Qtfhl8R/gn4X1zw3p+PsGhatoNvcWdvjpsidCq/gBR7N9wO4v/ir8L9L8JR+PtT+JGg2+hSAGPWrjWIEtHyMjExbYcj3pI/iv8LpfCH/AAsGL4k6A2g4z/bi6xAbPHr527Z+tcRq37P/AMG/EXw/h+C3iD4K+GLzwfb7RD4YutBt5NPjCjC4gKbBgcDA4FEf7PvwZtvh3/wpG3+CfhePwZtC/wDCLR6DB/Z+B0HkbPL/APHaPZ+YHb6R8Vvhdr/haXxzoPxJ0C+0WAEzaxZ6xBJaxgHB3SqxQc8cmk8NfFn4V+M9CuPFHg/4l+H9W0y1DG61HTdZgnghx13SI5Vcd8niuJ8P/s+/BnwV4DuPg/4N+CfhfS/Cd4pW68N6foMEVjMCckNCqBGyeeR1pvgv9nn4KfCnwpefDv4X/BDwt4f8P6kGGoaNougW9va3O4YYSRRoFbI65BzR7PzA7fwl8W/hT4/trq98CfE3w9rUNjn7dNpOtQXK2+Bk7zG5CYHrik8I/F34T/EC5ubLwH8T/DutzWZxeQ6RrUFy0B9HEbkr+OK4j4e/s6/BH4K2N/pPwc+BvhXwva6xuOrW/h/w/b2qXhYYbzRGih8jg5zkUnw8/Zy+B3wTvb3Vfg18C/Cnha61Zt2qXHh/w/b2j3Tc8yGJFLnk9c9aPZ+YHb+GPjD8JPGut3Hhnwb8UvDmralaf8fWn6XrlvcTw/76I5ZencUmkfGT4ReIPE83gnQfip4bvtat8+fpFnrlvJdR4ODuiVy4wfUVw/gf9mv4EfCXxPefED4V/Abwl4d17U+NS1jRPDtvbXVz1/1kkaBm6nqe9J4d/Zn+AvgnxxcfFvwZ8A/COl+Krzd9s8Raf4ctob6bJyd0yoHbJ55NHs/MDuY/jF8I5fGDfDyL4p+G28QKxDaGuuW5vFI6gw79/wClF98ZPhDpnixPAWpfFXw3b67I2I9Fn1y3S7Y4zgQl95456Vw0P7NPwHt/iLJ8cbf4C+EU8aSMzSeKl8O2w1FiTkkz7PMJJ/2qTVP2ZvgH4h+IMfxo134BeEbzxhbtug8TXXhu2fUI2xjInKFwccdelHs/MDu9f+MPwk8K+JIPBvij4peHNN1i6x9l0rUNct4bmbPTbE7hmz7CjxX8YfhJ4E1K10bxx8UvDmjXl7tFna6trlvbyT7jgbFkcFsngYBzXDeLP2a/gR8SPGtr8VPiF8BvCWueJ9O2nT9f1bw7bz3luV+7smdC64wMYPGKPHv7NfwI+Mmt2Pir4ufAbwl4l1TSSp0rUNe8O291NaFDuTy3kRimDyMEYNHs/MDuvF/xd+E/w+FqfHvxP8O6H9uwbL+2NagtvtGemzzHG78M07xZ8WvhX4C0u31vxz8TPD+i2V3/AMet5q2swW8U3+40jgN1HQ1wvxD/AGcvgd8bfsK/Gb4F+FPFI0hQNJ/4SDw9b3f2PHQReajbMe2Kd8QP2ePgn8Z9Es/CPxg+CHhXxNpOm/8AIP03XvD9vdQW3T/VpIhC9B0A6UezfcD5C/4LT/8ABXb45f8ABK3XPhz8dvB3gDw740+EnikHTta8ud2vEuwZJfMt3SQR7TGqDLZGc1lWf/BcD9h//gpL+w18QtH+E3jn+y/Fk3g+Z7jwfrHyXUBO35Q2Akh/3Ca5j/guF/wTD/aj/wCCjc/w2/Zi+D8Hhvwn8IfDO7Utc1JZPJntp/3kQgt7dI9jqIyjYLqM54714x4N/wCCMX7MH/BPj4FeMtb8C+HpvEHiSHw7Ilz4q1iPMmRj544yWEJ7/KauFGUrvsS5WP0P8O6nKPD9iPMP/HnF/wCgCrv9qS/89WrkPD+q40GxHmN/x5x/+girf9rf9NGrAo6T+1Jf+erUf2pL/wA9Grm/7W/6aNR/a3/TRqAOk/tSX/nq1cX8fvgn8MP2mvhfqnwi+MPhi31bR9UtmikhuI8mJiMCRD1DKcEe45zWh/a3/TRq5v4t/HL4f/A3wBqXxO+J/ieHS9H0u3aa6uriQAYAztGerHoB6mgD8Q/iB4e/av8A+DcD9rWPx34Ev7vxF8IvEl8wWzZ/3NxGx3eQem2VQCAeMhT1pnxc/ai/az/4ONP2mrP4EfBfTtQ8I/DLSWSbUrWaZT9miZuZrhlJDOQpwBxlTxUH7Rvxp/ab/wCDhT9qW2+B/wAGNOm0v4V+GtSL/bpFJjiVcp9qkwMFyGO0EnhutUNR+GH7V3/Bub+1RH8SPB13ceLPhl4geO31DUmtvKW/hVj8kiDcscihzjBOC3UUAftz+xF+xt8Gf2Evgtp/wh+D+gQ2/kwL/amp+X++v5v4pHb+QGBgDivZP7Um/wCerV4j+yl+2F8Iv2w/hDp3xf8Ag/4njvrO8hX7Vbqw8y0mx80bgE4IP6Yr0r+1v+mjUAdJ/akv/PVqP7Ul/wCerVzf9rf9NGo/tb/po1AHSf2pN/z1aj+1Jv8Anq1c3/a3/TRqP7W/6aNQB4J8NdS8qXxOrN18aaof/Jhq6f8Atcf3v1rzjwNqflXfiRQ/XxfqR/8AI7Vvf2uf+egr4KpUtUa82f1FhMJzYSnLvGP5I6n+1x/e/Wj+1x/e/WuW/tc/89BR/a5/56Co9sdH1LyOp/tcf3v1o/tcf3v1rlv7XP8Az0FH9rn/AJ6Cj2wfUvI6n+1x/e/Wj+1x/e/WuW/tc/8APQUf2uf+ego9sH1LyOp/tcf3v1o/tcf3v1rlv7XP/PQUf2uf+ego9sH1LyOp/tcf3v1o/tcf3v1rlv7XP/PQUf2uf+ego9sH1LyOp/tcf3v1o/tcf3v1rlv7XP8Az0FH9rn/AJ6Cj2wfUvI3/CGqY+MMkqt/zLRH/kwK9C/tl/7/AOleOeDtTJ+KEku//mAkcf8AXcV3f9rf9NGr+gOFaftOHcNL+7/7cz+QvED9zxljIdpL/wBJidR/bL/3/wBKP7Zf+/8ApXL/ANrf9NGo/tb/AKaNX0HsPI+O9qdR/bL/AN/9KP7Zf+/+lcv/AGt/00aj+1v+mjUew8g9qdR/bL/3/wBKP7Zf+/8ApXL/ANrf9NGo/tb/AKaNR7DyD2p1H9sv/f8A0o/tl/7/AOlcv/a3/TRqP7W/6aNR7DyD2p1H9sv/AH/0o/tl/wC/+lcv/a3/AE0aj+1v+mjUew8g9qdR/bL/AN/9KP7Zf+/+lcv/AGt/00aj+1v+mjUew8g9qY/7SninS9K+HUOoa9q9vZ2sOtWTTXF3MscaDzhyWYgD8TX0B+xx8Rfh9qfhKTxbpvjrRrjSrbm41ODVInt4h6tIG2r+Jr5d/aH0fw3468AR+F/F2iWuqafdaxZrcWOoW6zRSr5o4ZWBBH1r379kH4KfCjw38P7r4Z+G/hR4es/Dd8m280O10eGO0nBOSGiVdrc+or5fiSEoxgvJ/ofRZHLmjN+aPqDwx8WvhX430a48R+DPiZ4f1fT7TP2q+0vWoLiGHHXe6OVX8TSeEvi58KfH8N1ceBPid4e1qOxz9tk0nWoLlbfHXeY3OzHviuI8Dfs7/BL4R+GrvwJ8Kfgf4V8O6JqWRqOk6HoFvbW9znr5kcaBWzk9QetHw9/Zz+B/wVt76x+DnwM8KeF4dY3DVovD/h+3tVvN33vNEaLvz3znNfF+z8z6I7bwn8X/AITePr2507wL8UPDutXFkSLy30nW4Ll4Mdd6xuSuPfFJ4c+MPwk8Y6/ceFfCPxS8Oapqlrn7Vpum65bz3EP+9Gjll/EVxHw//Zu+BXwX1K+174P/AAI8J+GL7VmY6pd6B4et7WS7LHLeY0aKXyeuc0ngz9mr4DfCzxZefEj4Z/AXwjoHiLUs/wBoa3o/h23t7u5znO+VEDNnJ6nvR7PzA7fS/jJ8Idb8UyeB9F+Knhu81qE4m0e11y3kuk+sSuXH4igfGP4RN4v/AOFer8U/Df8Ab+cf2H/blv8AbM+nk79/6Vw+h/szfAPwl47m+MHhX4B+EdO8WXRzdeJLHw7bR30pPdplQOfxNC/s0fARPiL/AMLwX4CeER403bv+Eq/4R22/tHOc58/Z5mc/7VHs/MDuNR+Mnwh0fxVH4F1f4qeG7XW5jiLR7jXLeO6f6RFw5/Kl8RfGH4SeEPEFv4T8WfFLw5peqXWPsum6jrlvBcTZ6bY3cM34CuG1n9mf4B+KfHkPxi8TfAPwjqHi225t/El54dtpL6IjuszIXH4Gl8Y/s1/Ab4oeLrP4l/Er4DeEde8R6bj+z9c1jw7b3F3bYxjZK6FlxgdD2o9mB3Piz4v/AAm8B39tpfjn4oeHdFurwgWdvq2twW8k+emxZHBbPtmjxd8XfhR4AjtZvHnxO8O6It9j7C2r61BbC4z02eY4359s1w/j/wDZu+Bfxm1Wx8RfF74EeE/E2oaUynS7zXvD1vdS2pU5HltIjFMdsEUvxC/Zy+B/xrhsLX4x/Azwp4oj0fb/AGTH4g8P292LPb93yhIjbMdsYo9n5gdz4o+LXwq8D6Pb+IfGvxM8P6Pp93g2t9qmtQW8M2emx5HCt+Br5P8A+Cy/7ffxr/YL/ZV0n9r79nS38H+ItH0nWIz4ls9ZvGZL+xlKxL9maKRd7h33cEjjmvePHf7O/wAEvi/4dtPBHxY+B/hXxHoum4GnaXrmgW91b2wHTy45EKrjA6AV8m/8Fq/2If2gP2wP2QdH/Y5/ZV+HPg230rVNYji1a41YC1i0KxiZZQ9qEhcByy7do2Ag4zzR7NgZP/BPv/g5P/YW/b60hPA+s6xJ4B8b3lq8Ufh3Xm+W4cKQzxzKDGqk9Az7sEehrn/gbqyL8J9ECOCv2UlSG6/O1eB/sgf8G8X7NP7EusWnjPxXHcePvGFjA0n9oanbhLW3crkqsG5kfaeA5AOB0Ga9Y+C+oC2+Fui267VCWeAuBx8x4r5/iCLo+yb63/Q/V/DGn7ani12dP/289Q/tcf3v1o/tcf3v1rlv7XP/AD0FH9rn/noK+b9sfqf1LyOp/tcf3v1o/tcf3v1rlv7XP/PQUf2uf+ego9sH1LyNrxNYeHfGWg3XhjxVpNvf6fewtDdWlyoZJEYYII+h+tfjn+3J+xr8aP8AglF8c4/21P2O9Wuh4PkvvO1fTIm+SzDH5o5B3jJ6HnBK81+tWreLdO0LTLjWdY1GK2tbWFpbi4mYBY0UZLE+gAr8lP8Agod+3v8AFX/got8Uv+GG/wBjqzmvPDtxdCDWtQhXi92n5txXOIlI9eSo4r0stqVpVXb4ftX2sfIcX4bL6eCi53Va/wC65fj5ulvLv/mYf7Sf/BVv9o//AIKwXPh39kf9mbwLeeGm1pVXxAsc43XDcBzuDHbCuT3BweTX6Pf8E3P+CePwr/YI+GMVlp9tDqPi7UYg+u69MoMjOcZRDjhRgds5zzX5c/Fz/gn/APtRf8Ei/Efhv9rT4J+K28RR6bAo16aG32i2ZtpkhYDO6I4I3HqB0r9Pv2DP+ChHwx/bf+F0HiTwzqMdvr9nCq69orkb7eXuR6qcjnjrXRjppYdPD/w+tu/meVw3h5Tzacc2usWl7qla3Lb7FtL73t/mfUX9rj+9+tH9rj+9+tct/a5/56Cj+1z/AM9BXi+2P0T6l5HU/wBrj+9+tH9rj+9+tct/a5/56Cj+1z/z0FHtg+peR1P9rj+9+tN+FmqFPG3iqRW+8bH/ANFNXMf2uf8AnoKl+GWpkeK/Ecm/7xtOn/XNq+68PZe0zucf+nb/APSon5V4wUPYcLU5f9PY/wDpMz1z+2X/AL/6Uf2y/wDf/SuX/tb/AKaNR/a3/TRq/ZvYeR/M3tTqP7Zf+/8ApR/bL/3/ANK5f+1v+mjUf2t/00aj2HkHtTqP7Zf+/wDpR/bL/wB/9K5f+1v+mjUf2t/00aj2HkHtTqP7Zf8Av/pR/bL/AN/9K5f+1v8Apo1H9rf9NGo9h5B7U6j+2X/v/pR/bL/3/wBK5f8Atb/po1H9rf8ATRqPYeQe1Oo/tl/7/wClH9sv/f8A0rl/7W/6aNR/a3/TRqPYeQe1Oo/tl/7/AOlcH8RNTeX45fChg/3fE12f/JCetT+1v+mjVxnjfUt/xr+F7eYfl8RXR/8AJGavLzqjy5XVfl+qO7LanNjoLz/Q+m/7Um/56tR/akv/AD1aub/tb/po1H9rf9NGr8zPtjpP7Ul/56tR/akv/PVq5v8Atb/po1H9rf8ATRqAOk/tSX/nq1fM/wDwUv8A+CcPwc/4KN/Bi68E+MrGCx8SW8bN4f8AEyxfvrSTBwCQDlCcZBB6DFe6f2t/00avCf2+/wDgoj8Hf2Bfg1efEj4jawkupPEy6HoaSDzr6bBwAM8Lnqe2RxQB+S37L/8AwVJ/at/4IXfEDxJ+xz+1X4K1LxXotjDJJ4dtxcLviYgiKWNmYZhLKMjJOA2MGtX9jL9k79pH/gvL+0zN+2X+1trV1a/DTTdSb+z9O3YjnVH2i2hHPA24ZvVTzzXE/B/9hP8Aa4/4L1fF3xH+1p8ePFM3hPQGs5IfDt8bPzFcqGaK2iU7R5e5hlgeNzHBxz2v/BPP9un47f8ABGH48yfsIftl6TND4Dn1Jzpeo7SVtN77hPGWAyjbtxGRy3egD91fh94Q8H/CrwfY+Afh54fttJ0fTYFhsbCzj2xxIBgD3+p5ra/tSX/no1cX4V+IHh3xx4es/FnhLXIdQ02/hWazvLWQNHKhGQwPpWh/a3/TRqAOk/tSX/nq1H9qS/8APRq5v+1v+mjUf2t/00agD620X/kGx/8AAv8A0I1aqrov/INj/wCBf+hGrVBUviYUUUUEhRRRQAUUUUAFFFFABRRRQAdeoo2j0oooAasMSnKoKBDEDuEYrP8AGXi/w94B8K6h408WalHZ6bpdq9zfXUp+WKNRlmP0Ffkj4z/4OIv29/i14v8AEHir/gnt/wAEy5viN8LfDV5Jb6h4w1TUjbyP5XMrRosoDALkjijfYD9ddX1LQ/D2mz67r2o2tjZ2sZkuby8mWOOJR1ZmYgKPcms3wJ8RPhr8VNEXxT8MvHWi+ItNaQouo6FqkV3AWHUeZEzLkema+HPCP/BUL4Rf8FJf+CZHxM+IPgvSrrRdY0/w3dWXiPw9f4E1jdqCrpwx+XcrY5zgc4r5j/4Il/tneCf2LP8AgjndfGf4jyyf2VoepX1w0MOPMncIhWNckDcxwo5HJrSNKcpWsK5+yGs6nofh/TJ9d8Qala2NnaxmS5vLyZY4olHVmZiAo9yazfAXxH+GfxZ0GPxZ8MPHmieJNLlYrHqWg6pFeW7kdQJImZSfxr8Vfil/wX1/bx+M37KHjr4h/FP/AIJiatpfwU17w/cQ6P460a9WW5Cs4VJJYnmAC467QeenHNeyf8GwHxBbRv8AgmR4Z0VH/wCYrcf+gx0RpylsFz9RvH/xJ+GPws0lPEHxR8faH4csZJVijvde1WGzhZycBQ8rKpJJHGcmqXin41/BHwVZ6bf+NPi14X0m31gqNHm1TXre3S+LDKiEu4EmR025zX5zf8HSd2br/gnt4fuf+p3sD/5Hirx3/gpJ8efhl4P+Af7L9t8UP2edL8dNJNpUOnSahrl3ZnT5PIXEq/Z2XeR6NkURpynewN2P2iiNpcwrLAySRsuUZTkEeoNO8mLbt2DFfl3/AMFF/wDgup46/wCCcnxb+Fvw10H9n6HxdpfjS3hjlgtLp/tsbvhUjhUsFYk4HzEVwui/8HHv7X/wp/aq8L/Cr9uP/gndJ8NvBPja+8nw5rjaj5t66MT5byIJWjGV6gHOelEqcoysFz9Wr/4v/B3SvHNv8K9U+J/hu38S3UfmWvh241qBL6Zf7ywF/MYe4XFdIIoiuFQYr8SP23fiz4P8J/8ABx98OfiPrHiK1stLtfBLSTX1zMEjRRg5JNfT/wCwp/wWQ+Nf7e/7XfiT4f8AwD+A2myfBzwpM1vqXxG1e6mjkurgAZjtkXKvyw5IAxnmj2cuW47n6MCGIDATrQsMa/dTrVfTdWttSXMBq1WYDVhiXlUoWGJTlUFOooAp6npUN/btDsHzV4H+3r8P9K0n9jD4makluokXwtcMrAd+K+iK8Z/4KIf8mQfE7/sU7j+lVzStYD510LV3GiWY3n/j1j/9BFW/7Xf/AJ6GuR0TU1GjWg3r/wAesf8A6CKs/wBqL/fWpA6X+13/AOeho/td/wDnoa5r+1F/vrR/ai/31oAd8WPjf4E+CPgHUfiZ8TPE0Gl6PpcDS3V1cSBRgDOBnqxxwBya/Ff46fHr9qL/AIL2/tOr8DPgit5o3wr0W8DXE0issXlBstNKxwC5A4Xg5A4Oa9w/4Ksfsv8A7fn7cX7Vvh/4C6JMNO+FE0Imk1KxuCII8bBI04xuZhuOBgj71fcH7Hv7KPwg/Yv+Edj8K/hXo0MQhjBv9RaP99eS4GXduvbpnFAHQfsafslfB79iP4PWPwm+EeiRwLHGralflB5t5Pj5pGOB3JwOwrq/jr8IPhp+0f8ADTUvhP8AFvw9Dqmj6pA0c0MqglCQQHU9mGeDVv8AtRf760f2ov8AfWgD8U/GHhT9qr/g38/amXxz4Fu7vW/hL4kvv30KqzQvATzHJg4SRdvBOMgDjmv2J/ZY/a++FH7X3wk0/wCL3wi8Qx3VleRKbi18wGa0kI5jkXqCDkcgZwcVV+OHwj+G37Q3w31L4WfFHQLXUtL1K3aOSOaPc0ZPR0PVWBwcjFfmh+yt+wZ+3F/wTx/4KD2fhb4AXz6t8K9dmabULi9uD9ljtQ65RxyRKobAKjn5uaAP2B/td/8AnoaP7Xf/AJ6GuZj1QlFMrLu2/NtJxml/tRf760AdL/a7/wDPQ0f2u/8Az0Nc1/ai/wB9aP7UX++tAHhfhnVDFqHiEZPPirUD/wCRjWt/bB9TXD6XqXlatrw39fEl8f8AyKavf2t/00r8zxFS2ImvN/mf2ZlOD58roS7wj+SOq/tg+po/tg+prlf7W/6aUf2t/wBNKx9qd/1E6r+2D6mj+2D6muV/tb/ppR/a3/TSj2ofUTqv7YPqaP7YPqa5X+1v+mlH9rf9NKPah9ROq/tg+po/tg+prlf7W/6aUf2t/wBNKPah9ROq/tg+po/tg+prlf7W/wCmlH9rf9NKPah9ROq/tg+po/tg+prlf7W/6aUf2t/00o9qH1E7HwVqjf8ACfyyh8f8SfH/AJFFdv8A2w//AD2ryXwRqWfFs0gk/wCYdjP/AG0Fdl/ah/57Cv6h4Hp+04Uwsv7r/wDSmfwh4pv2PiBj4dpL/wBIidR/bD/89qP7Yf8A57Vy/wDah/57Cj+1D/z2FfV/Vz8/9qdR/bD/APPaj+2H/wCe1cv/AGof+ewo/tQ/89hR9XD2p1H9sP8A89qP7Yf/AJ7Vy/8Aah/57Cj+1D/z2FH1cPanUf2w/wDz2o/th/8AntXL/wBqH/nsKP7UP/PYUfVw9qdR/bD/APPaj+2H/wCe1cv/AGof+ewo/tQ/89hR9XD2p1H9sP8A89qP7Yf/AJ7Vy/8Aah/57Cj+1D/z2FH1cPalj4j6i1zpNlEZM/8AE5tP/Ror7G/ZHANioI9K+HfFV958VhH5uf8AibWv/owV9xfsjf8AHkv4V8PxhH2bgvJ/mfWcNy5qc35o+gmhiY5ZBQ0MTcslOor89PqBphibgpQYYiNpQcU6igBphiI2lOKPJi27dgxTqKAG+TFt27BigQxAbQg5p1FADRDEvAShYYl5VKdRQA1YYlOVQVHc2UM8LRbB81TUUAcdrPw30k2N7fT26lvssh3Y/wBk1+Vnw31P7P4I0+EE/LGw/wDH2r9edf8A+QFe/wDXpJ/6Ca/GTwbqXleGrWPf90N/6Ga+Y4mqO1K77/oftng7R9tLGL/r3/7cd9/bB9TR/bB9TXK/2t/00o/tb/ppXyntT9v+onVf2wfU1X1Txbp2iadNq+sahHa2tvGXnuJ5AiRqOpJPAFc7/a3/AE0r46/4Kt+GP2xvi7oeg/Cj9n62/wCKd1q58nXbi0mKzKx6bz2TG7OD6VvhVHEVlBySv1Z5mcSnleXTxMKUqkltGKu23ZL5X3fRHiv7eH7evxb/AG+vion7Gv7HEd2+jSXQh1XVLYN/pXzYZmbosQAzz155r7P/AOCfX7CHw4/Yh+HUdpZWsd94p1CFW1vWXUFmY8mNT2UHHrnFc/8AsHfsQ/Dn9jD4fx2dhDFfeJbyMNq+tOmXZsfcUnovXpjOa+gP7W/6aV14zMKfJ7DD6QX3yfdng5Bwri/b/wBqZr72Ilsvs04/yx8+7NzxHaaF4u0O68NeJtLhvrC8haK6tbhdySIRggj6V+UP7YP7IXxm/wCCaPxpj/a3/ZFvLtvDEl55mo6bbqW+yqT80bqvWM56443deK/UL+1v+mlU/EFrovirRbnw74isYbyyvIWiuLe4jDK6kYIINYYPMJYWfeL3XdHqZ9wpRzrDpX5KsNYTW8X/AJd0cH+wz+338N/20/hjD4n8OXkdrrVrGqazo8koEkUmOWVTyUJB5GQOOa9y/tg+pr8p/iz+wB+0J+yX+1PpHxl/YdeabS9W1ILdab5pEdsXbcVkB4MfB9SABX6SaJrGqvpFs+u+Wt4YV+1LAxKB8c4z2q8dHD05RqUJJxl06ryZy8N1M0xUKmFzGi41aTScre7NPaUX+a6Haf2wfU0f2wfU1yv9rf8ATSj+1v8AppXD7U+n+onVf2wfU1f+GuqMuu65IHxua2/9AauG/tb/AKaVrfDrUsX+qP5mNzQ/+gmv0jwtftOJJR/6dy/9Kifinj1h/q/BMJf9Pof+kzPUv7Yf/ntR/bD/APPauX/tQ/8APYUf2of+ewr+hPq5/HftTqP7Yf8A57Uf2w//AD2rl/7UP/PYUf2of+ewo+rh7U6j+2H/AOe1H9sP/wA9q5f+1D/z2FH9qH/nsKPq4e1Oo/th/wDntR/bD/8APauX/tQ/89hR/ah/57Cj6uHtTqP7Yf8A57Uf2w//AD2rl/7UP/PYUf2of+ewo+rh7U6j+2H/AOe1H9sP/wA9q5f+1D/z2FH9qH/nsKPq4e1Oo/th/wDntXJ+LdUaT4x/DdvMzt165P8A5Jy1J/ah/wCewrm/EGob/i58P2Mudut3B/8AJSWvF4io8uS1n5L80ejk9TmzKmvP9GfSP9rv/wA9DR/a7/8APQ1zX9qL/fWj+1F/vrX4+foZ0v8Aa7/89DR/a7/89DXNf2ov99aiv9Vu1spTpvktceWfJWZiELds45xQB5x+3j/wUG+Ef7CHwguviH8RNYjk1GaFl0TRY5B515N2wvJ2g9TjHBGa/LD9kz9lH9o3/gtj+0fJ+1z+11d3lr8O7G9Is9NkBQTx7vlghDZ+XauC2COldXon/BNf9r79vr9unXviN+3fd3Gn+EfDOrY0+zjmzDeRDDJHAB0XDAknafvfj+rXgXwx4P8Ahp4UsfA/gbRbXTdL063WG0s7WMKqKowOnU4HXqaAOk8BeGvCnwx8I2PgXwLo0Om6VptusNnZ20e1Y1FeF/8ABRb/AIJ//CL/AIKBfCO48JeMbCO28QWcTN4f15EHmW0uDgE45U56fTmvZv7UX++tH9qL/fWgD8df2Ef29f2gv+CQ/wAfZP2If21xeTeC/tQh0vU2VmjtMttEsT8hozgZAzgDrX7QeFviBoXjXw7Z+LPCutQ32n6hbrNaXdtIHSRCOCCOP8DXzN/wUJ/YH+Ef7fHwnn8H+LbW3s9eto2bRNejjxLbyY4DEclMgZBzgdBXz1/wRc+FP7fH7Nev+MPgX8frfzPAuh3HlaDeahcFnd8qQYDydhUnhiOc8dKAP0t/td/+eho/td/+ehrmv7UX++tH9qL/AH1oA/QDRf8AkGx/8C/9CNWqq6L/AMg2P/gX/oRq1QVL4mFFFFBIUUUUAFFFFABRRRQAUUUUAFFFFAHiv/BRb4c+Mfiz+xL8Svh94BSR9X1LwneQ2MUIJaSQwsAoA6kk1+TH/BHz/gpH+yN+zv8AsAt+zz8cfjToXg3xV4NuNQttc0bXr6O1ubiXzJX+RHYFyQwA468V+58iLIhjYcNwa+K/2qv+CMP/AAT3/aE+Jj/GP4m/sp+HtX8QySK8moGS4hLkHIJSGVEJz7c1tRdpEyPzB/4JgeFPFesfs+/tXftKabpF1Y+DPHWrX0/hpZoyi3MfmXGZl7HduBOPWvM/CPgXxb8Rv+Dc/XdO8I6S95LYa9Ld3EcYOVhjaJ3f6BVJ/Cv3I0P9mzwfdfDxfgsvgSztfCos/sbaNbxmOIwY27PlIPQdc596h+Fn7DHwM/Z+8CL8Hfgz8LbHRfC8jOZtHjMk0Llxhs+azE5HvXaqkY6Mmx+YOmf8FZf2F4f+COEfg25+Nuirq9n4CTTW8Ii8jN/JcKFjIEG7cQWBbOOnNejf8G01i6/8E8fC9wH+9qk/y/8AAY6+mrr/AIIlf8E9vB/ja/8AiH4W/ZG8Nxapq8ci6lcbrhxKJDlvkaUquT/dAx2xXpX7NP7HHw1/Zk8NWfw1+Bfw6t/DPhu1lMkOl2csrojHGTmVmbsO/alGWl7iPmb/AIOhP+UdXh3/ALHWw/8AR8VfJf8AwV6Zl+Bn7K+P+glpP/okV+1P7Qv7JfwE/ay+Fdv8Nf2ifhpaeKNItbhLq3sbyaWNVmQhlfMTqcggd+1eTfE//gnj+zZ8aYNB0D4o/Bux1fTfCMkb+Gba4uJ1WwaMYQrskBOB/eJrGhKMW/UuXQ/Mf/gqYseo/wDBSD9l+O7hWQLLCyqwzyMEfrV3/g4bjvrjxN8AXuJ25+IVuEXPT93NX6W+OP2AP2eviv4/0P4ofEr4R2Wqa94TI/4RvUpp51exx02hHCn/AIEDV/4qfsF/s7/tKjSbv9oL4RWfiWfw3fC80F7yedPskwBAdfLdcnDHrkc10e0j/XyIPw8/4K//ALI2pftuf8Fc/Av7Ov8Awkq6WupeG4Xub5o9/lwofmwMjJwfWvoH/gi78TviN+w1+0X4m/4JbfGTWVFrYyfbPAd067ftVsQu4DJ5+bzTxX6Wa5/wT++AXiL46W37Ser/AAks5/HGl2pttN8QNPMJYYT/AABQ+wj6qad4p/4J3/s9fEX4oaL8e/G3wps7zxl4cUro2uedNHNbg7sjCOqsPmb7wPWj2lLmcu/+QuyPo74c20cekrMJdzMorpK5H4b2eoadALW6B4rrq8+p8RstgooorMArzf8AbA+Gvin4xfsxeN/hf4Jhhk1bXNAmtdPS4m8uNpGxgM38I969IpGUMu0igD8wvEvg39qz4dxR6brvwW0gtFGqZh8VK2cDH/PKsX+3P2if+iKWH/hSL/8AGq/STxd8IdH8Uz+bdwo3OfmFYv8Awzf4Z/58ofyNdUfqvL71yfePz1/tz9on/oilh/4Ui/8Axqj+3P2if+iKWH/hSL/8ar9Cv+Gb/DP/AD5Q/kaP+Gb/AAz/AM+UP5Gq/wBj8w98/PX+3P2if+iKWH/hSL/8ao/tz9on/oilh/4Ui/8Axqv0K/4Zv8M/8+UP5Gj/AIZv8M/8+UP5Gj/Y/MPfPz1/tz9on/oilh/4Ui//ABqj+3P2if8Aoilh/wCFIv8A8ar9Cv8Ahm/wz/z5Q/kaP+Gb/DP/AD5Q/kaP9j8w98/PX+3P2if+iKWH/hSL/wDGqP7c/aJ/6IpYf+FIv/xqv0K/4Zv8M/8APlD+Ro/4Zv8ADP8Az5Q/kaP9j8w98/PX+3P2if8Aoilh/wCFIv8A8ao/tz9on/oilh/4Ui//ABqv0K/4Zv8ADP8Az5Q/kaP+Gb/DP/PlD+Ro/wBj8w98/PX+3P2if+iKWH/hSL/8ao/tz9on/oilh/4Ui/8Axqv0K/4Zv8M/8+UP5Gj/AIZv8M/8+UP5Gj/Y/MPfPyzt/gr8fo7q9vG8D2IN5qE11s/tYfJ5jbtv3ecetTf8Kd+Pv/Qj2P8A4Nh/8TX6jf8ADN/hn/nyh/I0f8M3+Gf+fKH8jXhz4eyOpNyfNd67/wDAP0rD+K3FuFw8KMPZ2ikl7nRK38x+XP8Awp34+/8AQj2P/g2H/wATR/wp34+/9CPY/wDg2H/xNfqN/wAM3+Gf+fKH8jR/wzf4Z/58ofyNT/q3kP8Ae+//AIBt/wARe4w/6d/+Af8A2x+XP/Cnfj7/ANCPY/8Ag2H/AMTR/wAKd+Pv/Qj2P/g2H/xNfqN/wzf4Z/58ofyNH/DN/hn/AJ8ofyNH+reQ/wB77/8AgB/xF7jD/p3/AOAf/bH5c/8ACnfj7/0I9j/4Nh/8TR/wp34+/wDQj2P/AINh/wDE1+o3/DN/hn/nyh/I0f8ADN/hn/nyh/I0f6t5D/e+/wD4Af8AEXuMP+nf/gH/ANsflz/wp34+/wDQj2P/AINh/wDE0f8ACnfj7/0I9j/4Nh/8TX6jf8M3+Gf+fKH8jR/wzf4Z/wCfKH8jR/q3kP8Ae+//AIAf8Re4w/6d/wDgH/2x+XP/AAp34+/9CPY/+DYf/E0f8Kd+Pv8A0I9j/wCDYf8AxNfqN/wzf4Z/58ofyNH/AAzf4Z/58ofyNH+reQ/3vv8A+AH/ABF7jD/p3/4B/wDbH5c/8Kd+Pv8A0I9j/wCDYf8AxNH/AAp34+/9CPY/+DYf/E1+o3/DN/hn/nyh/I0f8M3+Gf8Anyh/I0f6t5D/AHvv/wCAH/EXuMP+nf8A4B/9sfmX4V+Evxn029k1DUPCVqrNF5axrqAbjOc521vf8IP8V/8AoUbf/wADh/8AE1+i3/DN/hn/AJ8ofyNH/DN/hn/nyh/I195lPEVTJsvp4PD25IKyurve+r+Z+TZ/hY8SZxVzLGfxKjTly6LRJaLXoj86f+EH+K//AEKNv/4HD/4mj/hB/iv/ANCjb/8AgcP/AImv0W/4Zv8ADP8Az5Q/kaP+Gb/DP/PlD+Rr0P8AXXHf3fu/4J4/+rWA/vfefnT/AMIP8V/+hRt//A4f/E0f8IP8V/8AoUbf/wADh/8AE1+i3/DN/hn/AJ8ofyNH/DN/hn/nyh/I0f6647+793/BD/VrAf3vvPzp/wCEH+K//Qo2/wD4HD/4mj/hB/iv/wBCjb/+Bw/+Jr9Fv+Gb/DP/AD5Q/kaP+Gb/AAz/AM+UP5Gj/XXHf3fu/wCCH+rWA/vfefnT/wAIP8V/+hRt/wDwOH/xNH/CD/Ff/oUbf/wOH/xNfot/wzf4Z/58ofyNH/DN/hn/AJ8ofyNH+uuO/u/d/wAEP9WsB/e+8/On/hB/iv8A9Cjb/wDgcP8A4mj/AIQf4r/9Cjb/APgcP/ia/Rb/AIZv8M/8+UP5Gj/hm/wz/wA+UP5Gj/XXHf3fu/4If6tYD+995+dP/CD/ABX/AOhRt/8AwOH/AMTR/wAIP8V/+hRt/wDwOH/xNfot/wAM3+Gf+fKH8jR/wzf4Z/58ofyNH+uuO/u/d/wQ/wBWsB/e+8/Of/hXPxPv72zW78NwQxQ30Mski3gYhVcE8Yr7m/ZLt5oLNRKhXpXZf8M3+Gf+fKH8jXVeDfh7p/hNNtqirjsK8XNs7qZrZ1LXWmiselgcuo5fFqnfXudJRRRXz56IUUUUAFFFFABRRRQAUUUUAFFFFAFfVbeS70u5tYR80lu6Ln1KkV+THij9in9qv4W232LWvBujyJG7BZIdeVsjcTnGyv1vrmPGfw403xaCLuNW3diK48VluDzBxWIvZX2dtz6bhzi7N+FfaPA8t6lr8yvte1tV3PyS/wCFOfH3/oSLH/wbD/4mj/hTvx9/6Eex/wDBsP8A4mv1G/4Zv8M/8+UP5Gj/AIZv8M/8+UP5Gsf9W8h/vff/AMA+o/4i9xh/07/8A/8Atj8uf+FO/H3/AKEex/8ABsP/AImj/hTvx9/6Eex/8Gw/+Jr9Rv8Ahm/wz/z5Q/kaP+Gb/DP/AD5Q/kaP9W8h/vff/wAAP+IvcYf9O/8AwD/7Y/Ln/hTvx9/6Eex/8Gw/+Jo/4U78ff8AoR7H/wAGw/8Aia/Ub/hm/wAM/wDPlD+Ro/4Zv8M/8+UP5Gj/AFbyH+99/wDwA/4i9xh/07/8A/8Atj8uf+FO/H3/AKEex/8ABsP/AImj/hTvx9/6Eex/8Gw/+Jr9Rv8Ahm/wz/z5Q/kaP+Gb/DP/AD5Q/kaP9W8h/vff/wAAP+IvcYf9O/8AwD/7Y/Ln/hTvx9/6Eex/8Gw/+Jo/4U78ff8AoR7H/wAGw/8Aia/Ub/hm/wAM/wDPlD+Ro/4Zv8M/8+UP5Gj/AFbyH+99/wDwA/4i9xh/07/8A/8Atj8uf+FO/H3/AKEex/8ABsP/AImj/hTvx9/6Eex/8Gw/+Jr9Rv8Ahm/wz/z5Q/kaP+Gb/DP/AD5Q/kaP9W8h/vff/wAAP+IvcYf9O/8AwD/7Y/Ln/hTvx9/6Eex/8Gw/+JrZ8J/Cv4xaQk8174TtfMuGXKLqAO3AI67a/S3/AIZv8M/8+UP5Gj/hm/wz/wA+UP5GvayGlgeHMc8XhL8/K4+87qzs9tOx8vxZxhnXGmVrAZjy+zUlP3Y8ruk0tbvTVn50/wDCD/Ff/oUbf/wOH/xNH/CD/Ff/AKFG3/8AA4f/ABNfot/wzf4Z/wCfKH8jR/wzf4Z/58ofyNfZf6647+793/BPzb/VrAf3vvPzp/4Qf4r/APQo2/8A4HD/AOJo/wCEH+K//Qo2/wD4HD/4mv0W/wCGb/DP/PlD+Ro/4Zv8M/8APlD+Ro/11x3937v+CH+rWA/vfefnT/wg/wAV/wDoUbf/AMDh/wDE0f8ACD/Ff/oUbf8A8Dh/8TX6Lf8ADN/hn/nyh/I0f8M3+Gf+fKH8jR/rrjv7v3f8EP8AVrAf3vvPzp/4Qf4r/wDQo2//AIHD/wCJo/4Qf4r/APQo2/8A4HD/AOJr9Fv+Gb/DP/PlD+Ro/wCGb/DP/PlD+Ro/11x3937v+CH+rWA/vfefnT/wg/xX/wChRt//AAOH/wATR/wg/wAV/wDoUbf/AMDh/wDE1+i3/DN/hn/nyh/I0f8ADN/hn/nyh/I0f6647+793/BD/VrAf3vvPzp/4Qf4r/8AQo2//gcP/iaP+EH+K/8A0KNv/wCBw/8Aia/Rb/hm/wAM/wDPlD+Ro/4Zv8M/8+UP5Gj/AF1x3937v+CH+rWA/vfefnT/AMIP8V/+hRt//A4f/E1i+Ivhp8bW8TeH/FGk+B7S4bRr6Sd7d9UCeYGhaPGdpx97PSv0w/4Zv8M/8+UP5Gj/AIZv8M/8+UP5GuXGcVYjH4WWHq25Zb2Wvfub4bIsJha8asL3Xmfnr/bn7RP/AERSw/8ACkX/AONUf25+0T/0RSw/8KRf/jVfoV/wzf4Z/wCfKH8jR/wzf4Z/58ofyNfN/wCx+Z7Xvn56/wBuftE/9EUsP/CkX/41R/bn7RP/AERSw/8ACkX/AONV+hX/AAzf4Z/58ofyNH/DN/hn/nyh/I0f7H5h75+ev9uftE/9EUsP/CkX/wCNUf25+0T/ANEUsP8AwpF/+NV+hX/DN/hn/nyh/I0f8M3+Gf8Anyh/I0f7H5h75+ev9uftE/8ARFLD/wAKRf8A41R/bn7RP/RFLD/wpF/+NV+hX/DN/hn/AJ8ofyNH/DN/hn/nyh/I0f7H5h75+ev9uftE/wDRFLD/AMKRf/jVH9uftE/9EUsP/CkX/wCNV+hX/DN/hn/nyh/I0f8ADN/hn/nyh/I0f7H5h75+ev8Abn7RP/RFLD/wpF/+NUf25+0T/wBEUsP/AApF/wDjVfoV/wAM3+Gf+fKH8jR/wzf4Z/58ofyNH+x+Ye+eiaL/AMg2P/gX/oRq1VXRf+QbH/wL/wBCNWq4zSXxMKKKKCQooooAKKKKACiiigAooooAKKKKACmSwRTDbKgan0UAQxafaQtujgUH/dpxtoGOTEvHtUlFAEUtnbTDEkKn8KamnWcbbkgUf8BqeigACgLtA4qP7NBu3eUvPtUlFAEYtbcdIV59qEtoEGFjH5VJRQBEbO2JY+UPm9qVbaBV2iJfyqSigBscEUX+rTFOoooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAq6L/AMg2P/gX/oRq1VXRf+QbH/wL/wBCNWqCpfEwooooJCiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACviX/gpf/wVS8efsJftOfBv4EeF/hbo+uWXxN1L7NqGoajeSxyWK5k+aMJwx+QdfWvtqvwy/wCDt3xh8VfAH7QH7PnjD4I6I2peKrC6eXRbFbI3BmmBuML5Y5f6ChAfr3+0z+0PP8Df2cvFnxs8PaTb6ld+HdElvobG5kZY5mUZ2sV5A+leW/8ABKj/AIKD69/wUE/ZI0n9ozx74J03w3f6heTQyabpt08sKKmOQ0nPevwY+NH/AAVk/wCDinxN8GfEngT4pfstyWXhnUtMkg1q5k+FlzD5UBHzHzC+E+temfCf9uj4l/sZf8G9Wm+L/hhrc2m+INU1Q6fHfW+VksxO0UckiYOQyhyR7gV0RpczfZInmP6I9P8AiZ8OdV1P+xdM8faLcXpbb9jh1SFpd3psDbs/hWpqeraVoljJqes6nb2dtEMyXF1MscaD1LMQBX8jWoftD/8ABMj4M/DbQv2i/wBlP9oL4wJ+01aXEN9qmualDKlne3TNmZM/wg7m+bcc496/UH/gpP8AtufCj9pD/gml8KfFf7Vn7WetfDvwr4vS0m8Z6H4X0d7zUPEMLBDLAuyVDGhBYFirAb+lR7KTi2ug7n7J6D8Rvh74puzYeGPHejalOq7mhsNUimcD1wjE1b13xN4c8L2f9oeJfEFjp1vux599dJCmfTLkCv5U/g7r3hT9ir/gpj8Mtc/YS+DHxa+GHhXXr+OBNc8eanL/AMVTZMjEuIWhj2JnOF3P9zOeePtb/godrvir/gqj/wAFgPCf/BPH4vfETXtN+FGm+GG1bWdC0bU2gXUJB9nG48EBx5jbWIOAx9afsanLzedhc0T79/4KH/8ABXPxJ+x1+0t8G/gt8OfAXh3xVpHxP1pbK81qTVHzaKWUbovKJVzz3Nfa+keIrPVoY3hcbnUHj6V/Nr+3V+w/pf7Bf7f/AOzz4E+D3xZ8Vap8O49et08O+F/Emqtd/wBlTCYGRo2OAFZdowFHSv32+Aeu3eqJH5zH/Vr1qvq8o03JhzanslFA6UVzlBRRRQAUUUUAFFFFABRRRQAUUUUAFFFFAFXRf+QbH/wL/wBCNWqq6L/yDY/+Bf8AoRq1QVL4mFFFFBIUUUUAFFFFABRRRQAUUUUAFFFFABRRRQAUUUUAFFFFABRRRQAUUUUAFFFFABRRRQAUUUUAFFFFABRRRQAUUUUAFFFFABRRRQAUUUUAFFFFABRRRQAUUUUAFFFFABRRTZJBGNzf3gKAHUUUUAFIzKil3OAoyT6UtBAIwRQAKwZdynINFHTgCigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAr8yf+C4H7EP7Tf7TX7Xn7P8A8Uvgp8NJda0HwXqhm8SX0d9BGLNP33JWR1ZvvL90HrX6bVV1bTI9UtWt3P3qqL5ZJgfEP7Uf7Onjn4yfs5eOPB/gHSXvta1DRJoNPsRIqebKRwuWIA/E18T/AAc/4I4/GT4i/wDBJxP2NPj5oA8J+MFumu7GO7mjnW2uI3WSJmMRcEFkGcE8V+0Hh/wTa6NK0kWPmPpUepeAbO/1EXj7d38JxXZ9a94z5T8B9C/Z8/4LYS6dof7Kc37KHwz8HaZpM8dm3xeh02zmuJbaI480xuz53KP+eWeee9ezf8FP/wDgmv8AtZfEHRfhL8af2XNOs/G3ir4V3qyTeG9Sht4bfV1JiJ3IQkYX93yBt4PFfsH4h+FWnasFZkUMO9S6B8MdP0y2aIheeKPbx5A5WfgL8d/2Tf8Agrz+1R+0N8G/2ofjl8BvDulw+F9QS2vvAfhO9jQ6TCpk/wBJdpZirZDY2xseCOK9j/b3/wCCfn7bnh39rDwv/wAFH/2AdA0/XvFOk2H2DXPB+oXAja9j/dnG9mRQn7vDYYHJGO9fsLf/AAX0q4vHm2D5jV/RfhdpunwNAka4btR7aHJZMOVn4HfFT9kT/gsH+1t+1r8IP2p/2jPhFoem6doWqQ/2h4Q8P6hHt0REkDNO7SysZGYHGEZvu9K/br4AIFddsW3AA6V0158FdMmuWkCqAx5X1rc8HeA7Pw1/x7qAO9KVWHs2kxqJ0w6UUDjiiuIsKKKKACiiigAooooAKKKKACiiigAooooA/9k=)**

**Deletion at end of list**

**Algorithm:**

**Step 1:** IF HEAD = NULL

Write Underflow

Go to step 7

[End of If]

**Step 2:** SET TEMP = TAIL

**Step 3:** SET TAIL=TAIL->PREV

**Step 4:** SET TEMP ->PREV = NULL

**Step 5:** SET TAIL->NEXT=NULL

**Step 6:** FREE TEMP

**Step 7:** EXIT

Diagram

Description automatically generated

**Deletion at specified position of list**

**Algorithm:**

**Step 1:** IF HEAD = NULL

Write Underflow

Go to step 10

[End of If]

**Step 2:** SET TEMP = HEAD

**Step 3:** SET I=0

**Step 4:** Repeat Step 5 and 6 until I<POS-1

**Step 5:** SET TEMP = TEMP -> NEXT

**Step 6:** SET I=I+1

    [END OF LOOP]

**Step 7:** SET TEMP -> NEXT =TEMP->NEXT -> NEXT

**Step 8:** SET TEMP -> NEXT -> PREV = TEMP

**Step 9:** FREE TEMP

**Step 10:** EXIT

![Diagram

Description automatically generated with medium confidence](data:image/jpeg;base64,/9j/4AAQSkZJRgABAQEAAAAAAAD/4RCyRXhpZgAATU0AKgAAAAgAAodpAAQAAAABAAAIMuocAAcAAAgMAAAAJgAAAAAc6gAAAAgAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAFkAMAAgAAABQAABCAkAQAAgAAABQAABCUkpEAAgAAAAMwMAAAkpIAAgAAAAMwMAAA6hwABwAACAwAAAh0AAAAABzqAAAACAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAMjAyMzowMToxMSAxNDozNTozNQAyMDIzOjAxOjExIDE0OjM1OjM1AAAA/+EJnGh0dHA6Ly9ucy5hZG9iZS5jb20veGFwLzEuMC8APD94cGFja2V0IGJlZ2luPSfvu78nIGlkPSdXNU0wTXBDZWhpSHpyZVN6TlRjemtjOWQnPz4NCjx4OnhtcG1ldGEgeG1sbnM6eD0iYWRvYmU6bnM6bWV0YS8iPjxyZGY6UkRGIHhtbG5zOnJkZj0iaHR0cDovL3d3dy53My5vcmcvMTk5OS8wMi8yMi1yZGYtc3ludGF4LW5zIyI+PHJkZjpEZXNjcmlwdGlvbiByZGY6YWJvdXQ9InV1aWQ6ZmFmNWJkZDUtYmEzZC0xMWRhLWFkMzEtZDMzZDc1MTgyZjFiIiB4bWxuczp4bXA9Imh0dHA6Ly9ucy5hZG9iZS5jb20veGFwLzEuMC8iPjx4bXA6Q3JlYXRlRGF0ZT4yMDIzLTAxLTExVDE0OjM1OjM1PC94bXA6Q3JlYXRlRGF0ZT48L3JkZjpEZXNjcmlwdGlvbj48L3JkZjpSREY+PC94OnhtcG1ldGE+DQogICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgCiAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAKICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgIAogICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgCiAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAKICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgIAogICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgCiAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAKICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgIAogICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgCiAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAKICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgIAogICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgCiAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAKICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgIAogICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgCiAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAKICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgIAogICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgCiAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAKICAgICAgICAgICAgICAgICAgICAgICAgICAgIDw/eHBhY2tldCBlbmQ9J3cnPz7/2wBDAAIBAQEBAQIBAQECAgICAgQDAgICAgUEBAMEBgUGBgYFBgYGBwkIBgcJBwYGCAsICQoKCgoKBggLDAsKDAkKCgr/2wBDAQICAgICAgUDAwUKBwYHCgoKCgoKCgoKCgoKCgoKCgoKCgoKCgoKCgoKCgoKCgoKCgoKCgoKCgoKCgoKCgoKCgr/wAARCAD9BBsDASIAAhEBAxEB/8QAHwAAAQUBAQEBAQEAAAAAAAAAAAECAwQFBgcICQoL/8QAtRAAAgEDAwIEAwUFBAQAAAF9AQIDAAQRBRIhMUEGE1FhByJxFDKBkaEII0KxwRVS0fAkM2JyggkKFhcYGRolJicoKSo0NTY3ODk6Q0RFRkdISUpTVFVWV1hZWmNkZWZnaGlqc3R1dnd4eXqDhIWGh4iJipKTlJWWl5iZmqKjpKWmp6ipqrKztLW2t7i5usLDxMXGx8jJytLT1NXW19jZ2uHi4+Tl5ufo6erx8vP09fb3+Pn6/8QAHwEAAwEBAQEBAQEBAQAAAAAAAAECAwQFBgcICQoL/8QAtREAAgECBAQDBAcFBAQAAQJ3AAECAxEEBSExBhJBUQdhcRMiMoEIFEKRobHBCSMzUvAVYnLRChYkNOEl8RcYGRomJygpKjU2Nzg5OkNERUZHSElKU1RVVldYWVpjZGVmZ2hpanN0dXZ3eHl6goOEhYaHiImKkpOUlZaXmJmaoqOkpaanqKmqsrO0tba3uLm6wsPExcbHyMnK0tPU1dbX2Nna4uPk5ebn6Onq8vP09fb3+Pn6/9oADAMBAAIRAxEAPwD9/KKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKAPBfjp8b/jf4f8AjpdfDP4ba74X03T7HwnpupyS614cuL6aaa5utQiIBjvIAiqtmmBhiS7c9BXPf8Lp/aw/6KR4A/8ADf3v/wAtqoftHXws/wBrjWMtjd8OdA/9L9arB/txf79eDWxFSNaS5nu+p+o5bk+Eq5fSm6UXeKd3FN3sr6tHW/8AC6f2sP8AopHgD/w397/8tqP+F0/tYf8ARSPAH/hv73/5bVyX9uL/AH6P7cX+/WX1qp/M/vZ2/wBh4P8A58x/8Bj/AJHW/wDC6f2sP+ikeAP/AA397/8ALaj/AIXT+1h/0UjwB/4b+9/+W1cl/bY/v0f22P79H1qp/M/vYf2Jg/8An1H/AMBj/kdb/wALp/aw/wCikeAP/Df3v/y2o/4XT+1h/wBFI8Af+G/vf/ltXJf22P71H9t/7VH1qp/M/vYf2Jg/+fUf/AY/5HW/8Lp/aw/6KR4A/wDDf3v/AMtqP+F0/tYf9FI8Af8Ahv73/wCW1cl/bf8AtUf23/tUfWqn8z+9h/YmD/59R/8AAY/5HW/8Lp/aw/6KR4A/8N/e/wDy2qTwh+0/8Z9D+LWi+EPip4l8K3+l6zY3rq+j+Gbixmimh8nb80l7OrAiRsjaDwOa47+2x/erh/H2oG6+MXgkK5+WHUf/AG3rfC16lTEwi5PVpb+Zx5jk+EpZfWmqUVaEnpFLVJ21S7n3to+pxarZrdQnKsuc1arn/hqf+KWtf+uK/wAq6CvekrSsfloUUUVIBRRRQAUUUUAFFFFABRRRQAUUUUAFFFFABRRRQAUUUUAFFFFABRRRQAUUUUAFFFFABRRRQAUUUUAFFFFABRRRQAUUUUAFFFFABRRRQAUUUUAFFFFABRRRQAUUUUAFFFFABRRRQAUUUUAFfPf/AAUS/wCCk3wI/wCCZfw58N/FP9oHRPEV3pHiXxVBoUM3h61gma0mlRmEsolmj/dAIc7NzdMKa+hK/OX/AIOQ/gB8Xf2hPgF8IfD/AMJPhJr3i+TS/jVpd/rFroWkS3jWtmiyb55VjU7IxkZY4AzQB6h+yl/wXD/ZL/as/aB179mmz8GfELwH4m0bQ21qzh+JXhU6SusaYoJ+2WoZ2cxFAXHmLGSvOODjjPBP/Bxz+wl46+LFh4J0/wAE/FCz8Iat4mbw9o/xi1LwW0XhK+1INs8iO98zdktwCYwvIOcEGvmj9qf9lr47/FL/AILEeG/iNoPw/wDEA8L3HwT17RrzxlDpc32C1upreZIoZLgDYrEsMKTnnivg34Ef8E2f2mNL03wv+x18Y/2f/wBqq/u9D8XRzXXk/EqGy+HccEVwJEvoBJbTJuGN/lhdxP8AEpPHS6GuhPMftV4t/wCC8n7JPgf9nX4pftG+JvAPjy2s/hJ44/4RXxD4ebT7I6ldXpfbGbZBdeW8cg+ZS8iEjqo6V1nhb/gsJ+zR448U/Avwh4P8L+LNSvPj9pcupeF/s1naldLtYlJklvs3AMaqw2HyhL83tzX5h/FP9h79o+f/AIKrzfDWL4N69e/BHx14l0rxv4w8TPp8smmz3lnaPF9mkm2+WshO04JBJ6Ctf/glZ+yv+074E/4KC6xqXxz+E2vaR4X+EGi6poHww13UtNmhtdQtr6+Mx+zM42uFQgZUkcU/YIOY+3P+C2X7bP7RX7J2s/s7x/s8fEltBg8bfF230bxYiaTaXX2+wby90P8ApEUhjzk/NHtbnrXGeDP+Cudt8Jfjt+1h4t+Lnxf8eeM/Cfwaa1nTwPH8P9HsV0eJn2mOxu47vzdQ3ZGTdCHAHGap/wDBcf4C/G74w2n7Mt58K/hZ4h8TR+HfjFBqHiCbRdJluhp1qFTM8xjU+XGMH5mwK+Q/ij+yz+0vqHiD9v6bT/gN4umj8eWVqng14dAnYa6yyAkWmE/fkd9mamnTjK9wuz70+E3/AAccfsP/ABU+Lfw++Fq/D74oaHY/E0rD4R8c+IfB4t9BvbwqCbRLrzT5kisQhaNXj3EAOcgnyP4b/wDBbHxJ8BP2rf2wJf2x/izdat4B+GHjaz0r4beGNM0Wy+2tJM0wWxtBFHHJdzSFVAErtjBJZQGNeC/tF/so/tE+IP2Pv2JvBvhH4FeKrnUPAnjTw/N4m02x0Gd5tCgj8nzZLhFXMCrtO4sABjnpXi/xu/4Jg/tNfEv9rP8AaW/a68E/D7xlpnjbwX8QrLxR8IZLzT5V07xE8DtJMkSMm25Z1UAFCTkKoxuzVewQcx/QZ8Efi4/xi+GWh/ES98Ba14Vn1rT47tvDviRYFv7HeuRHOsEssauAeQrtjpXZV85/8E+fi54p+N3wG8O/Ef4h/D7XvCWvappscmreG/EenSWt3YXGMSRPHIoYYYHBxgjBGQc19GDpWNSPLKw0FFFFZjCiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigD49/a5vvsX7XOof7Xw50P/wBL9Zrif7cre/bkvhZftc3GT974c6L/AOl+sV5l/bi/3q+NxtTlxc15s/oLhvB+1yHDytvBHX/25WP4S+Lvgbx5LqcHgvxbp+qPoupPp+rLY3SyGzukVWaCTafkcBlJU8gEVkf22P79fF37GPiP9pLTfH3x2j+D/hPwPfaa/wAaNQaabxL4hvLWcS/ZrXICQWkylcY5LA5J4rOnL2kZO+1juxVP6vWpw5b8zeybeivsj7Q1v9qH4F+GtJ8Ra74g+Lfh+zs/CNylv4nurjVI1j0qZ1DKk5JxGxVlIDYJBFcvp/8AwUK/Y01a9h07TP2m/Bc9xcSLHBDHrsRaRicAAZ5JNfnf8VtU8bXX7O/7XM3jWw0mDXJfiNpLXVvpd5LNaLILe12hZJI0crjGSUBzng45+4PA/jH9slzpcfiLwJ8MYdN2RC6lsvF2oyTrFgZKo1gqs2OxYD3racY043b/ABS6J9tdzzsPUrYqq4xjbS/wt/bnHWzVvh6932PaPiT8c/hl8HdBj8U/FXx9pPh3TZLqO2jvtYvkt4mmkbakYZyBuYnAHc0/x98bPhz8LPB83xB+JHjjS9D0OAKZtW1S8WG3QN93LsQOe3rXwt+2R4y8I/tIftSR/Azxr4I8VeIPBfgTw/Nc6wvhnwvd6mv9tXcZjtlk+zI2xok3SgN3xXGfEX406v8AEn/glpP4M8ZpqEPiPwP4s03QdZt9WtHt7oiC9jEEskTgMheHYcMAetONPmjF33av89v680TWxip1K0YpWipcr7uCXMn89v8ADI++fAH7ZH7NXxU1ddA+HPx08L61fSf6u10/WIpJG+gBya9A/tz3r5F/4KRad4L1L9i7xJ4y1a0t4da8P6VFe+F9YVAt1ZX6snktDIPmDFvlwDyDg5r2v4T+Kdb1T4W+G9S8TlhqVxoNpJf7uvnGFS+ffdmsJTj7NTj3t91j0qFCo8VLD1Em1FSutNG2rNXdnp31PT/7c965zVb/AO2fGbweM/dt9Q/9t6p/22P79UtKvxefGvwqoP3ba+/9oV0ZbV5swpL+8jk4iwfs8hxMu0Jfkfop8NP+RWtf+uK/yroK5/4af8ita/8AXFf5V0FfYVPiZ/PYUUUVIBRRRQAUUUUAFFFFABRRRQAUUUUAFFFFABRRRQAUUUUAFFFFABRRRQAUUUUAFFFFABRRRQAUUUUAFYPxB+KHw5+E+hP4n+JvjnS9B0+P715q18kEY/FiK3q+O/2Zvh94W/ahvJv22fjHocPibVPFGoTyeAdP1qJbmz8O6KkrJafZYXBRJpIwJZJsby0hUNtAAqMeYD0Wf/gqp/wTftpWguP22PhwrKcMreKIOD/31TP+Hrf/AATZ/wCj3fhv/wCFRB/8VXrVr4X1J4FQXPlqq4VIxgL7YqT/AIRLUP8An/kquWPcDyH/AIet/wDBNn/o934b/wDhUQf/ABVH/D1v/gmz/wBHu/Df/wAKiD/4qvXv+ES1D/n/AJKP+ES1D/n/AJKOWPcDyH/h63/wTZ/6Pd+G/wD4VEH/AMVR/wAPW/8Agmz/ANHu/Df/AMKiD/4qvXv+ES1D/n/ko/4RLUP+f+Sjlj3A8h/4et/8E2f+j3fhv/4VEH/xVH/D1v8A4Js/9Hu/Df8A8KiD/wCKr17/AIRLUP8An/ko/wCES1D/AJ/5KOWPcDyH/h63/wAE2f8Ao934b/8AhUQf/FUf8PW/+CbP/R7vw3/8KiD/AOKr17/hEtQ/5/5KP+ES1D/n/ko5Y9wPIf8Ah63/AME2f+j3fhv/AOFRB/8AFUf8PW/+CbP/AEe78N//AAqIP/iq9e/4RLUP+f8Ako/4RLUP+f8Ako5Y9wPIf+Hrf/BNn/o934b/APhUQf8AxVH/AA9b/wCCbP8A0e78N/8AwqIP/iq9e/4RLUP+f+Sj/hEtQ/5/5KOWPcDyH/h63/wTZ/6Pd+G//hUQf/FUf8PW/wDgmz/0e78N/wDwqIP/AIqvXv8AhEtQ/wCf+Sj/AIRLUP8An/ko5Y9wPIf+Hrf/AATZ/wCj3fhv/wCFRB/8VR/w9b/4Js/9Hu/Df/wqIP8A4qvXv+ES1D/n/ko/4RLUP+f+Sjlj3A8h/wCHrf8AwTZ/6Pd+G/8A4VEH/wAVR/w9b/4Js/8AR7vw3/8ACog/+Kr17/hEtQ/5/wCSj/hEtQ/5/wCSjlj3A8h/4et/8E2f+j3fhv8A+FRB/wDFUf8AD1v/AIJs/wDR7vw3/wDCog/+Kr17/hEtQ/5/5KP+ES1D/n/ko5Y9wPIf+Hrf/BNn/o934b/+FRB/8VUGo/8ABUz/AIJq6hbtbyftu/DfDcH/AIqi3/8Aiq9l/wCES1D/AJ/5KP8AhEtQ/wCf+Sjlj3A8B/4eMf8ABMD7K1r/AMNq/DXaxzj/AISi3/8Aiqq3/wDwUF/4Jcagqib9tP4a7l6H/hKLf/4qvoj/AIRLUP8An/ko/wCES1D/AJ/5Krm8wsfP03/BRP8A4Jez6eunyftp/DUovT/iqLf/AOKpkn/BQz/gl5J5W79tT4a/ueV/4qi34/8AHq+hP+ES1D/n/ko/4RLUP+f+Sjm8wPE5/wDgp/8A8E0ZtP8A7PP7bfw22bcY/wCEot//AIqsc/8ABQ7/AIJekKP+G1fhr8rZ/wCRot//AIqvoT/hEtQ/5/5KP+ES1D/n/kpL1A8Bg/4KMf8ABMC38zy/21fhr+8GG/4qi3/+KqE/8FDP+CXZCg/tqfDX5Wz/AMjTb/8AxVfQn/CJah/z/wAlH/CJah/z/wAlPm8wseD6V/wUl/4JkaTdm6tf22PhqrHrt8Uwf/FVtD/gq1/wTZAx/wANu/Df/wAKiD/4qvXv+ES1D/n/AJKP+ES1D/n/AJKl2luwPIf+Hrf/AATZ/wCj3fhv/wCFRB/8VR/w9b/4Js/9Hu/Df/wqIP8A4qvXv+ES1D/n/ko/4RLUP+f+Sjlj3A8h/wCHrf8AwTZ/6Pd+G/8A4VEH/wAVR/w9b/4Js/8AR7vw3/8ACog/+Kr17/hEtQ/5/wCSj/hEtQ/5/wCSjlj3A8h/4et/8E2f+j3fhv8A+FRB/wDFUf8AD1v/AIJs/wDR7vw3/wDCog/+Kr17/hEtQ/5/5KP+ES1D/n/ko5Y9wPIf+Hrf/BNn/o934b/+FRB/8VR/w9b/AOCbP/R7vw3/APCog/8Aiq9e/wCES1D/AJ/5KP8AhEtQ/wCf+Sjlj3A8h/4et/8ABNn/AKPd+G//AIVEH/xVH/D1v/gmz/0e78N//Cog/wDiq9e/4RLUP+f+Sj/hEtQ/5/5KOWPcDyH/AIet/wDBNn/o934b/wDhUQf/ABVH/D1v/gmz/wBHu/Df/wAKiD/4qvXv+ES1D/n/AJKP+ES1D/n/AJKOWPcDyH/h63/wTZ/6Pd+G/wD4VEH/AMVR/wAPW/8Agmz/ANHu/Df/AMKiD/4qvXv+ES1D/n/ko/4RLUP+f+Sjlj3A8h/4et/8E2f+j3fhv/4VEH/xVH/D1v8A4Js/9Hu/Df8A8KiD/wCKr17/AIRLUP8An/ko/wCES1D/AJ/5KOWPcDyH/h63/wAE2f8Ao934b/8AhUQf/FUf8PW/+CbP/R7vw3/8KiD/AOKr17/hEtQ/5/5KP+ES1D/n/ko5Y9wPIf8Ah63/AME2f+j3fhv/AOFRB/8AFUf8PW/+CbP/AEe78N//AAqIP/iq9e/4RLUP+f8Ako/4RLUP+f8Ako5Y9wPIf+Hrf/BNn/o934b/APhUQf8AxVH/AA9b/wCCbP8A0e78N/8AwqIP/iq9e/4RLUP+f+Sj/hEtQ/5/5KOWPcDyH/h63/wTZ/6Pd+G//hUQf/FVe8Pf8FNf+CfHizUo9H8N/tj/AA9vLqVtscFv4mgZmP8A31XqH/CJah/z/wAlUfEfws0zxfpM2g+LNMtdVsZ12zWOpWiTQyL6MjgqR9RRyx7gdTpesaVrdjHqej6jDdW8yhoZ7eQOjqe4I4IqzXyL4A8IyfsU/tcab8HvA0stv4A+Jem3l3pXh/zi1vo2q2q+ZKtqp/1UMkO5jGvyKyjaACa+s9Puhd2yyg9qmUeXUCeiiipAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooA+Df+CiF99j/a4+9jd8OdI/8AS/Vq8Z/tv/ppXp//AAU5vTZ/tcQ/Njd8OdK/9L9VrwT+2j/fr8/zSpy5hUXmf1VwVg/a8K4SVvs/qzrP7b/6aVyHwn+Fngr4NXfie98HSXm/xb4km1zVvtc4f/SpI4422YUbUxGuBzznmn/20f79H9tH+/XCsRJJpPc+llldOU4zcdVt5X0ON8X/ALIHwa8aeGfiB4V1S71qO3+JWrQ6j4ikt75VkSeKONEMJKEIMRrwQ3NWrL4Aa/YtD5f7V3xSaOFl2wyanpxUgfwn/Qc4/Guo/to/36P7bP8AfrT65Uta/wDWiOb/AFfwfNzKFn5XXVvo11bfzZH8I/hX4N+DL6/deGbvULq78Ta3JqusX2p3AlmmnZVXGQq4RVUBVxwK4/4o/sf/AAd+LF54tvdc1HXLNvGi2J1qPS79I1860bdDOgaNtsnYscggdK7T+2z/AH6P7bP9+pWKqKXMnqVLIsHUoqlKmnFX09U0/vTd/U48fsqeCtZ1LT734n/Evxl42t9JukubDTPFGqQvaxzJyjmKCCISFTyN+4A84r1wa0FG1WwB0Fcn/bZ/v0f22f79EsTKduZmlHJ8Ph7+zja+76v1b1Z1n9t/9NKs/D+/+2fHDw2u7O20vT+sFcV/bZ/v1u/Be+N58ddBBbO2yvP5w135PU5s0or+8jwOMsH7PhXGSttTZ+o/w0/5Fa1/64r/ACroK5/4af8AIrWv/XFf5V0Fff1PiZ/KQUUUVIBRRRQAUUUUAFFFFABRRRQAUUUUAFFFFABRRRQAUUUUAFFFFABRRRQAUUUUAFFFFABRRRQAUUUUAI/KMD6V8t/8EsB5v7CHwZ3fw/D7Tv8A0UK+pH+4fpXy5/wSs/5MR+Df/ZPdP/8ARQq4gfUaAKuBS0DpRUAFFFFABRRRQAUUUUAFFFFABRRRQAUUUUAFFFFABRRRQAUUUUAFFFFABRRRQAUUUUAFFFFABRRRQAUUUUAFFFFABRRRQAUUUUAFFFFABRRRQAUUUUAFFFFABRRRQAUUUUAFFFFABRRRQAUUUUAFFFFAHzb+10gP7Yn7Opx/zE/FH/plkr37woxbTkz/AHa8D/a6/wCTw/2df+wl4p/9Msle9eEv+Qav+7VfYA1qKKKkAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigD4e/4KU/s1/G34ifHG1+KvgDw1a32kx+DbPT5nk1JIpFniu72VhtbkjbOnP19K+Xj8DPj0p2n4fx8f8AUSjr9dtW0az1mHyLyMMvTDCsY/CvwieTpcP/AH7ryq2RZZia0qtXmu+zt+h+g5X4mcS5Pl9PBYdU+SCsrxbe99XzI/KL/hRvx6/6J8n/AIMo6P8AhRvx6/6J8n/gyjr9Xf8AhVXhD/oFQ/8Afuj/AIVV4Q/6BUP/AH7qP9W8l/v/APgS/wAjv/4i9xd2pf8AgD/+SPyi/wCFG/Hr/on6f+DKOqfiP4U/HHwz4ev/ABJe/D0NDp9nLczKupR5KxoWOPfAr9af+FVeEP8AoFQ/9+65T48fC7wnD8D/ABlKmlQ7l8K6gR+7H/PtJR/q3k39/wD8CX+Qf8Re4v8A+nX/AIA//kj8s/h34G+MvxJ8BaH8QdF+Huy117R7XUbWOTUo9yxzxLKoPvhhmtr/AIUd8ev+ifp/4Mo6+7P2Efh74ZvP2T/hRJNpkbF/hjoDNle506A17j/wqrwj/wBAqH/v2KHw3ky/n/8AAv8AgAvF7jD/AKdf+AP/AOSPyi/4Ud8ev+ifp/4Mo6P+FHfHr/on6f8Agyjr9Xf+FVeEP+gVD/37o/4VV4Q/6BUP/fuj/VvJf7//AIEv8g/4i9xf/wBOv/AH/wDJH5Rf8KO+PX/RP0/8GUddZ8Bfgz8W9G+L2m+IfE3hNbW1t7WaMut2shLOY8cD/dNfpj/wqrwh/wBAqH/v3Tovhf4UhkEkemQhl6Hy62wuR5VhMRGtT5rxd1d6fkcOaeJnE2b5fUwWI9nyVFZ2i07Ps+Yk+HCPF4Yt0dcERL/Kt+orOzhsoRDCuAKlr1JPmlc/PQoooqQCiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAR/uH6V8uf8ErP+TEfg3/2T3T//AEUK+o3+4fpXy5/wSs/5MR+Df/ZPdP8A/RQq4gfUg6UUDpRUAFFFFABRRRQAUUUUAFFFFABRRRQAUUUUAFFFFABRRRQAUUUUAFFFFABRRRQAUUUUAFFFFABRRRQAUUUUAFFFFABRRRQAUUUUAFFFFABRRRQAUUUUAFFFFABRRRQAUUUUAFFFFABRRRQAUUUUAFFFFAHzf+11/wAnh/s6/wDYS8U/+mWSvevCX/INX/drwX9rr/k8P9nX/sJeKf8A0yyV714S/wCQav8Au1X2ANaiiipAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigArkvj7/wAkK8af9inqP/pNJXW1yXx9/wCSFeNP+xT1H/0mkoA8z/YG/wCTTPhL/wBkv8P/APptgr3mvBv2Bv8Ak0z4S/8AZL/D/wD6bYK95q5gFFFFQAUUUUAFFFFABRRRQAUUUUAFFFFABRRRQAUUUUAFFFFABRRRQAUUUUAFFFFABRRRQAUUUUAFFFFABRRRQAUUUUAFFFFACP8AcP0r5c/4JWf8mI/Bv/snun/+ihX1G/3D9K+XP+CVn/JiPwb/AOye6f8A+ihVxA+pB0ooHSioAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACsbwx8Rfh/42vNQ0/wZ450fV7jSbr7Nq0Ol6lFcPZTc/uphGxMb8H5WweDX54f8FAf+CgP7eGr/wDBUDw//wAE2/2D/GXhHwfJY/D8+LvGPizxXoY1BXiaSRI7ZIiRjJjALKQf3oPavgD9j7/gpL8Xf+CeHwB/bQ/aQ8ReGdJuPHk3xugtZLNWf+z11KeWaIzf3vJDOZNvXbx1rSNOUtQuf0SzzwWsD3VzMsccalpJJGAVVAySSegArL8G+PvAvxF0ceIvh9400nXdPaVohfaNqMV1CXXhl3xsy5HcZyK/GD9hn/gvF8Yvir8f/wDhlP44ftPfD34w6f49+HOqX1nrngXw1NpbeG7+K0aQ2MyyKonBGVDjqRnjv4N+yd/wVX+IX/BN7/gjz4D0r4Uapoul6948+MGs6bb+I/Etu81jocXmB5LuSJATJtA4XB57UKnIVz+i6jPOK/CPwx/wW2/a4/aO/Z4/aG/Zy8KftdeA/FXizwZ4HXXvD/xd8J+E5LW11CxbKXVl9lkCmKcA4WQfd681zvwz/wCCuv7Y3/BPv/gk/wDA2z8SfHTwxqWvfFC6jtfC/i7xJobtB4T0oBmkkuwpZ72QHJ3Hr0xR7OVrhdH7/Vj+KviF4B8CTWFv438caPo0mqXQttLj1XUordryY/8ALKISMPMfkfKuTzX4y/s+f8HI3xN8PfBb9oK2+JHxO8F/GDXPhV4Ug1rwh468I6NLptjrXnuIhDJbOAUaJzk4GCBjmvFf20fjP/wUi+KE/wCx38Tf23PH/grxFpviT4qWOt6LD4Y0P7DcaJI/lMLWUg7Z12bSHwDnINCpylsO5/Q2rq3Kmlr5L/4KDftX/Ff9nj/gm78VP2lfhFqtvZ+KfCvh37Zot1eWaXEccvnxJlo3yrDax4NfB3h3/gsB/wAFUPhT4o/Zl+O/7QviHwHqXw3+O2oWOk33g/SdBEN9pklykYjuXuP4nYyByigIuCoB4NHs5c1kK5+0tFfkL4p/4Kb/APBUr9sz44/HYfsFePfAPgTwP8CdYn0iK28T+HRqF34qvbdGeRd7EfZ1bbwwHAccZBNc14h/4OBv2tP2j/2Z/wBmXw3+ylofhvwp8W/jx4p1PRtc1bWLL7ZYaH/ZkoiuZo4X4kWQlpFB5Cptzk5o9nIZ+w9t8Rfh/eeMrr4c2fjnR5vENjarc32gxalE17bwtjbI8AbzEQ7lwxAB3D1FbAIIyDX4G/Bb9qb9oH9lb/gsL+1R+0X+1lc6L4i8WeB/2a7e71HUPC9q1ra65HaxWkkUiREnyGkWNFZckK27HGKP2Sv+DlL9pDUPjH8HfEfxk/aV+F/jDQ/it40t9C8R/CTwz4ZntNU8ER3cpjt7lrx1C3IRtm9cnG7HJ+YDpyja4rn75UVm+GvEdr4islvLVwysMrWlWbVtBhRRRQAUUUUAFFFFABRRRQAUUUUAFFFFABRRRQAUUUUAFFFFABRRRQB83/tdf8nh/s6/9hLxT/6ZZK968Jf8g1f92vBf2uv+Tw/2df8AsJeKf/TLJXvXhL/kGr/u1X2ANaiiipAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigArkvj7/AMkK8af9inqP/pNJXW1yXx9/5IV40/7FPUf/AEmkoA8z/YG/5NM+Ev8A2S/w/wD+m2Cvea8G/YG/5NM+Ev8A2S/w/wD+m2CveauYBRRRUAFFFFABRRRQAUUUUAFFFFABRRRQAUUUUAFFFFABRRRQAUUUUAFFFFABRRRQAUUUUAFFFFABRRRQAUUUUAFFFFABRRRQAj/cP0r5c/4JWf8AJiPwb/7J7p//AKKFfUb/AHD9K+XP+CVn/JiPwb/7J7p//ooVcQPqQdKKB0oqACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigD89/8AgpT/AMEoPi98b/2xvDf7fv7JP7TLfDXx5pfhlvDuvvNoKahBqem73YL5bMAJAZHwzbh904yor5z+Gf8AwQFul+APxm/Z2+LvxtvvEi/FfxdB4itfES6Z5N5p19EzSCWTLMsrGU7jgKuMjGK/ZCe3iuF2yoDUK6NYI25YFB/3a2hV5Y2sS0flb+y5/wAEqP2pvhJ4+uNe/aB/arj8XLa+HZtH0DTND8HwaVaIrxGMT3Kx5aeVR/Fkc5rk4P8Ag3x1x/2LvDH7NF98cVj8X+CPHF34q8J+NLHQv3NtczNny5LaR28xNvykE89sV+vzaRYs+8wLn/dp39m2mc+Uv/fNV9YDlPy3+BX/AASf+OVr4D+Ivgv9pj9pCXxZrHj7QZNI8/RfDUGl6fpkRUjfDbRj75OCxJ5rz2L/AIIIfGrUf2U/CH7OPxF/anhv/Enwp1xb/wCFXiyz8Ixxx6XCFI+z3ELM32lTnksc+mK/YhdIsVk80QLu/wB2nNpVm7+YYlz9KPb+Qcp+U/w4/wCCOvxO8Y/Bn4lfB/8Aa2/aEuPGV/8AEbShp81xo3h2DS7DS0Uhke3toxhW3gMxJOcV5k//AAQn/bFvrn4S6b8Zf26m8UWHwb8TW974Z01/CIhhNlEV/duVk3POVUL5jEgADC1+0a6RYq/mCBd3+7RJo9hK294FP/AaPb+Qcp8y/tRfsl69+2J+wj8QP2V7HxRD4du/Gmimxh1a6s2nS1PmxybzGGUt9zGAw6180fGL/gjx42+Kvwm/Z5+FkPxcsrOT4C+I9J1O8vZNFd11lbMRgxxr5g8kv5fBJbGehr9NYbK3gTZHGMU3+zbTJPlLz7VKq+82HKfkj8W/+CLX7TPh/wCLfxJ+Jv7F37Ytx8NdD+MVxJP8QPDt14aS/DTyKyyzWjllNu7h2+bkgnjgAUnxB/4IH2tz+zt8Hfhx+yr8YL/wP4u+BeqT6n4U8aX+ni7a5uLlxJeNcRAqHEsg3bQQBkrjBxX63tpVmybGhX/vmkh0qxgG2OBefaq9t5Byn5G/Bf8A4IufE7Sv2ivip8a/2m/2gZviKfjF8MW8JeLreTQxZy7nWON5IjG+yOIRpsSMLlcKckg5pfs2/wDBE39pj9nHx14J8N+Jv2xH1j4a+Ab5bjw74d03wbbWeoXcaEmK3vr1QWuIlOPlwPugDAr9ff7F0/dv+zrn120smkWMmN8C8f7NP6xrsHKcz8IdEuNG0GOKfd0/irsKbFDHAuyNcCnVhKXNK5QUUUVIBRRRQAUUUUAFFFFABRRRQAUUUUAFFFFABRRRQAUUUUAFFFFAHzf+11/yeH+zr/2EvFP/AKZZK968Jf8AINX/AHa+F/2v/wDgpD+wvZ/tefCybV/2m/DOmv8ADfXvFVr42tdXums7jSpf7FcBXgnVJGDEgKyKwckBSxIr7O+CXxC8I/FL4daT8QvA2qNeaPrVjHd6bdPaywmaFxuR9kqq6gjBG5QcGq+wB2FFFFSAUUUUAFFFFABRRRQAUUUUAFFFFABRRRQAUUUUAFFFFABRRRQAUUUUAFFFFABRRRQAUUUUAFFFFABXJfH3/khXjT/sU9R/9JpK62uS+Pv/ACQrxp/2Keo/+k0lAHmf7A3/ACaZ8Jf+yX+H/wD02wV7zXg37A3/ACaZ8Jf+yX+H/wD02wV7zVzAKKKKgAooooAKKKKACiiigAooooAKKKKAPN/Ef7WPwT8L+KNS8HX+pa/cX+j3QttSTSvBOrX0cExjSTyzLb2rxltkiNgMcBhmqn/DZXwO/wCpy/8ADY69/wDIVcl8M9U+x/Ef4qxb8f8AFxmP/lK06u0/t7/prQBB/wANlfA7/qcv/DY69/8AIVH/AA2V8Dv+py/8Njr3/wAhVP8A29/01r4A/wCCgH/BYj47/wDBOL9tzwno3x8+BNv/AMM6+J7NLP8A4WBpTST3VpqLN8zy9FQIMfusZZckMSMUwPvj/hsr4Hf9Tl/4bHXv/kKj/hsr4Hf9Tl/4bHXv/kKqPgj4q+EfiT4R03x94B8UWmr6Lq9ml1pupWEwkhuYXGVdWB5H8uh5rV/t7/prQBB/w2V8Dv8Aqcv/AA2Ovf8AyFR/w2V8Dv8Aqcv/AA2Ovf8AyFXxd/wV0/4LMeGP+Cffhqx+Enwf0RPG3xu8YFbfwd4JtVMxhaQ7UuLhEO7bk/KnBc+ijJ+nP2ZPiJ8cPGXwD8KeKP2kPBen+G/HF9o0M3iTRNLuzNDaXJUFkVj056rk4PGT1pgdr/w2V8Dv+py/8Njr3/yFR/w2V8Dv+py/8Njr3/yFU/8Ab3/TWj+3v+mtLQDOtf25v2br5Xez8R+IJBHM8MmzwHrJ2SIxV0P+icMrAqVPIIIPIqX/AIbZ/Z5/6DXiP/wgNZ/+RK+ffAOvNDJ4lUS9fHevHr66ncV0H/CRv/z2/WvEeZV79Puf+Z+jQ4Ry2UE/f1S+1H/5A9i/4bZ/Z5/6DXiP/wAIDWf/AJEo/wCG2f2ef+g14j/8IDWf/kSvHf8AhI3/AOe360f8JG//AD2/Wl/aWI8vuf8AmV/qjlv9/wD8Cj/8gexf8Ns/s8/9BrxH/wCEBrP/AMiUf8Ns/s8/9BrxH/4QGs//ACJXjv8Awkb/APPb9aP+Ejf/AJ7frR/aWI8vuf8AmH+qOW/3/wDwKP8A8gexf8Ns/s8/9BrxH/4QGs//ACJR/wANs/s8/wDQa8R/+EBrP/yJXjv/AAkbf89v1o/4SRv+e/60f2lX8vuf+Yf6o5b/AH//AAKP/wAgexf8Ns/s8/8AQa8R/wDhAaz/APIlH/DbP7PP/Qa8R/8AhAaz/wDIleO/8JI3/Pf9aP8AhJD/AM9//HqP7Sr+X3P/ADD/AFRy3+//AOBR/wDkD2y2/bE/Z7vIFubbxleOjfdYeG7/AP8AjHX+VSf8Nd/AL/obb7/wm7//AOMV8oeEb4jQY/mH+tmPX/pq1aX28/3h+dfyBivpL8aYfFTpLC4e0ZNfDU6O3/P09xeHOUtX9pU++P8A8gfTv/DXfwC/6G2+/wDCbv8A/wCMUf8ADXfwC/6G2+/8Ju//APjFfMX28/3h+dH28/3h+dYf8TOca/8AQLh//Aav/wAtK/4hzlP/AD8qffH/AOQPptv2u/gGVIHi2+/8Jy//APjFfPP/AATh+OHw++HH7GXwq8JeOptX0vVNL8EWNtqNjeeGr9JbeZYwGRl8nIIPasr7ef7w/Oj7ef7w/Orj9J7jRRf+yYe/+Gp8/wDl76dfz0n/AIhxlV/4lT74/wDyJ9OD9rv4BY58W33/AITl/wD/ABil/wCGu/gF/wBDbff+E3f/APxivmL7ef7w/Oj7ef7w/Oo/4mc41/6BcP8A+A1f/lpX/EOcp/5+VPvj/wDIH07/AMNd/AL/AKG2+/8ACbv/AP4xQf2vPgCOT4uvv/Cbv/8A4xXzF9vP94fnSSX58tvmH3fWj/iZzjX/AKBcP/4DU/8Alof8Q5yn/n5U++P/AMgfbHhzxToPi3R7PX/D2oLcWd9ax3FpMqlfMjdQytggEZBBwQDXzn+3b/wVm/ZY/wCCf3i/w98LfiXp3jLxZ448VW8lzofgH4b+GX1fWLi2TO6fyFZdsYweSwJ2tgEK2L/7D3ijVtX8E6bp97cs0dtptvHEpPRVjUAflXxB/wAFNPA37RP7Hf8AwWP8P/8ABTnw/wDsz+M/it8PdW+FB8Japb/D/SxqOraDeLNNIsiWu4MyOJFG5SAAZMkfKG/trL6k8VgaFadk5wjJ22vKKbt5an49iIRo15wjsm19zsehftm/8FwfCHxH/wCCTnxM/bD/AOCeXxFvNF8Y+CdWsdN1HTfFHh2JdS0K6e8ijkhurO4WRAxQuAfmGQ2DlTjuPGv/AAXR/Zy/Zo8PfCn4ffGzTPHHjL4jeOPA+n6xeeH/AIc+ETql5Gj2qPNeSwRMnlxbvMb5ASArYXC1+W/jj9kv9rj4h/sXftdftJan+zd4r0C8+OfjDRrzwf8ADOPS5LjWDaWtyM3E1pArOjsGDFcbgRISMYY+oa9oX7Sv7FP/AAUL+Gv7ZQ/ZG+IXxE8L658AtK8JXkPgvQzd3+h6hFByksTlfIUlgpZyoHz5ORg+j7FGHMfaX/BFP/gqB8Wv26/EHx01b4m/ECPXNB8P/Ew2Hw/xocNi9tpbCRkjkVI0dmACg+ZlwQQe9TfHb/gob+0Z8PP+Czcv7JNr8QY7T4br8C7zxG+mx6DazTR6gkUpWcSMokbBUHyy4RsYPWvn/wD4ICfDn41eDvG/x28S/Gn4Ga94AvPE3xYbVLHSdY02SBWhlEr5icqEnA3AM8ZZS2cGum/bC+Cvxl1T/guLffFfS/hL4muPC0n7Od5pyeJoNBuH09rwxTAWwuAnlmUkjEe7ccjik6cYySC52n7NP/Bd34SfCL9h/wAD/tA/te/GHxb46Xxp49v9AtPFUPw9sdHkheNvlE1nb3bokaDgOjO7n+DNewfB3/gt5+y3+1bd/Ez4RfBuz8beFfH3gvwZe6zY6b8QPCTaZJewpA7xXdvHKzGSLIVhvVSVIO3Br8jtC/ZS/aPu/wDgnZ+zl8PdW/Zv8cNqGj/tD3Gpa1os3hC88+yszOCLmeIxbooiBnzGAXHevqX4/wDwO+Neo/8ABYzUvixoXwp8TTeHbj9nfWdObxFZ6DcNYyXbQMsdr9oVPLMpzhY9249hVexjIOY9e/4Jx/8ABdGXXv2SvhTeftbaprXjz4ufE7xVqGm6PoPgnw/Zi8ngiuGQXLwI0EUMEaj5pCR04DGv0yvfGGh6T4buvFes3X2WysbGS7vZHXd5MSIXckLnOADwM9OM1/Nv+wZ+yN+1N/wT6tPhD+3x4V+CvxI1XUtQ16/8O/E7wJeeGbubUNP0ye4PlXdtaeV50aLjcxC4wM85Ffvz4j1KHV/2WPF2oxNL++8Fag+2aMq4zaSHBVsFT7EZFZVoezp8y7GuHj7bERg+rS+9lmP9t79nWaNZoNf8QujqGR08A6yQwPQg/ZOlO/4bZ/Z5/wCg14j/APCA1n/5Erw3wp4iZfC2mr53TT4e/wDsCtD/AISNv+e36187/aVfy+5/5n6NLg/LVK3v/wDgUf8A5A9i/wCG2f2ef+g14j/8IDWf/kSj/htn9nn/AKDXiP8A8IDWf/kSvHf+Ejf/AJ7frR/wkb/89v1p/wBpV/L7n/mL/VHLf+nn/gUf/kD2L/htn9nn/oNeI/8AwgNZ/wDkSj/htn9nn/oNeI//AAgNZ/8AkSvHf+Ejf/nt+tH/AAkb/wDPb9aP7Sr+X3P/ADD/AFRy3/p5/wCBR/8AkD2L/htn9nn/AKDXiP8A8IDWf/kSj/htn9nn/oNeI/8AwgNZ/wDkSvHf+Ejf/nt+tH/CRv8A89v1o/tKv5fc/wDMP9Uct/6ef+BR/wDkD2L/AIbZ/Z5/6DXiP/wgNZ/+RKbN+3B+zjbRNPd+JNehjXl5p/AusRoo9SzWgAHuTXj/APwkb/8APb9a534ta+03w21mLzvvWLDrW+FxlWviqdOVrSkls+rS7nLjeF8vw+DqVo894xbV5RtdK+vuL8z6U/4ax+Cn/QS1/wD8InVf/kaj/hrH4Kf9BLX/APwidV/+Rq89/wCEik/vtR/wkMn/AD0avvv7Fwn9771/8ifkn9pYjy+5/wCZ6F/w1j8FP+glr/8A4ROq/wDyNR/w1j8FP+glr/8A4ROq/wDyNXnv/CQv/wA9Go/4SF/+ejUf2LhP733r/wCRD+0sR5fc/wDM9C/4ax+Cn/QS1/8A8InVf/kaj/hrH4Kf9BLX/wDwidV/+Rq89/4SF/8Ano1H/CQyf89Go/sXCf3vvX/yIf2liPL7n/mehf8ADWPwU/6CWv8A/hE6r/8AI1H/AA1j8FP+glr/AP4ROq//ACNXnv8AwkT/AN9qD4ik/wCejUf2LhP733r/AORD+0sR5fc/8z0L/hrH4Kf9BLX/APwidV/+RqZP+138CbSPz73XtYt49yr5tx4P1ONAScAFmtgBkkAc8kgd64D/AISKT/no1cp8bNce4+HdxCXPOoWH/pZCa8DiulTyLhfH5lQTc6FGrUipO8XKEJSSaSTabWtmnbZo78pxFTH5pQw1SyjUnCLstbSkk7Xb1100Z7X/AMNd/AL/AKG2+/8ACbv/AP4xR/w138Av+htvv/Cbv/8A4xXzF9vP94fnR9vP94fnX8J/8TOca/8AQLh//Aav/wAtP3T/AIhzlP8Az8qffH/5A+nf+Gu/gF/0Nt9/4Td//wDGKP8Ahrv4Bf8AQ233/hN3/wD8Yr5i+3n+8Pzo+3n+8Pzo/wCJnONf+gXD/wDgNX/5aH/EOcp/5+VPvj/8gfTv/DXfwC/6G2+/8Ju//wDjFH/DXfwC/wChtvv/AAm7/wD+MV8xfbz/AHh+dH28/wB4fnR/xM5xr/0C4f8A8Bq//LQ/4hzlP/Pyp98f/kD6d/4a7+AX/Q233/hN3/8A8Yo/4a7+AX/Q233/AITd/wD/ABivmL7ef7w/Oj7ef7w/Oj/iZzjX/oFw/wD4DV/+Wh/xDnKf+flT74//ACB9O/8ADXfwC/6G2+/8Ju//APjFWdE/ak+CHiHWbXw/pfiy4a6vZhDaxzaJeRK7nou54go6dyK+Wft5/vD86zPEWu3emTaXqNlLtlh1eFo2U9DzXr8P/SM4wzfPsJgamGw6jVq04NqNS6U5KLavVavZ6XT9Dlx3h/leFwdWtGc7xjJ7x6Jv+U+94Z4rhd8T7hXwd8Z/+DjP/gnx8Gfib4g8AS6P8TPE+l+D9TGn+MPHvg/wJLfeH9DuNwBS5vA424J52q2e2a+uPhNdXvin4e7Lu8kWS6s2TzlPzJuXG4e4zX4Q6f4c/bN/YY/Zt+OX/BMjxF/wT4+I3jLxF4+8VatL4R8Z+F9FS60HUoL6UlJ7q83BbcoG6OMjAztr+z1T/eOJ+RX0P0K/aN/4KcfE2y/4Kifsx/CT9n/4r6fefCL4reC77WdYjttNtphqSqJTDKk7oZYxhU4Rl6cjrXnP7ef/AAcmfAXw/wDAH4w6D+yJeeOF8ZeDbdrHRfiSvgH7Z4Yj1dZY82xun3x+YU8zAljCt2J4z80/D39if9on4G/tYfsY+A9e8F69qkHgH4U32leKvEmlabPcafpVyyS/unuVUxx4LYUMwz2zXiNj4Y/bM+A//BOf47f8Ewr7/gnv8Ste8Sal4hv9Q0/xZpXh4z6Lf2cl2k4nWb71xMAu1IoldjwTt2kVr7HQXMfvF/wT9/aG8WfH/wDZM+GvxG+ImrR33iPXvBOnX+u3UdukKzXUtujyMEjAVMsTwoAHavzG8Af8Fpf2uNV/4JteJvj78XP2orzw34l0/wDaWbwlpviTQ/hfpeqv/ZYHyWZtJJLaLDHObgsZFx0bPH3b/wAEnPDuueFP2Sfht4c8S6NfabqFj4F0yG90/UrV4bi2lW3QNHJG4DIwIwVIBBr8cfEH7Jn7VMH/AASc8SfDeb9mj4gJ4in/AGtf7Wh0BvBt8L2Sw8wn7WsHleYYMf8ALUDZ70nTj7SwXdj9cPjP/wAF5/2Q/wBnn47+KP2YfGvg74la5428K6DY6o+keEPB41KbV47iFZsWkcUxdjHGxeQuscaBT85yoPiv/BRz/guNP4k/4Jb+A/26/wDgmh8UJNNbxJ8ZNJ8Nag2taBbS3VrC/wBpF1ZzW86ypHJlIzuUngqVbDc8D8Cfgz8ULD/gvT8Wvjbq/wALNet/D958GdI07S/Fl1ok6WM84+xebBFcsnlvIPK+ZFYsNhyODXw94v8A2Nf2qL7/AIJI+Lvgwn7PHxHtdWvv215tVstFtfCF8l8ulmBEGoQw+Vv8gZOJgpjyOvFP2MQ5j93/AIGf8FLfgZ+0L+1T4z/ZQ+E+j+Itcvvh9bIfGHi6zsYToWnXjkgac1yZgz3YwSY0jYLghmBBA9i+Jnxp+HPwg0mx1rx/rNxaw6lffY7FbTS7m8knm8p5dix28bv9yORs4wApya/J3/gk98MvjB/wTq/bM+IX7AGoeBvFmqfCjxBfHxl8OfiJJo89xAkkqhbrTr69RPLE4MeU8xgzBM9ZVFfoP+2NdLawfC+5ifgeNpsN/wBwbUa5cdzYei5x3/4J6WT4WnmGYQoVL2d9t9E31T7djrf+G2f2ef8AoNeI/wDwgNZ/+RKP+G2f2ef+g14j/wDCA1n/AORK8d/4SN/+e360f8JG3/Pb9a8T+0q/l9z/AMz7j/VHLf7/AP4FH/5A9i/4bZ/Z5/6DXiP/AMIDWf8A5Eo/4bZ/Z5/6DXiP/wAIDWf/AJErx3/hJG/57/rR/wAJI3/Pf9aP7Sr+X3P/ADD/AFRy3+//AOBR/wDkD87P+Cuv/BM/4Iftqf8ABTL4Z/tb/DaHWm8O3TgfFS3XwTq0RkNr+8gkANsuTJjyyQGJOM4Ar9M/gf8AtRfCPwjoEGma7deILOO2jWK3tY/AWr7Yo1GFUBbTAAAwBXP/APCSH/nv/wCPUf8ACSH/AJ7/APj1X/auI5eW0fuf+Yv9T8t/6ef+BR/+QPYv+G2f2ef+g14j/wDCA1n/AORKP+G2f2ef+g14j/8ACA1n/wCRK8d/4SQ/89//AB6j/hJD/wA9/wDx6o/tKv5fc/8AMf8Aqjlv9/8A8Cj/APIHsX/DbP7PP/Qa8R/+EBrP/wAiVJa/tpfs73c/2YeKdWhkZSVS68H6pCXA67fMtl3Y4zjOMjPWvGf+Ekb/AJ7/AK1yvjnWGuvFeinzQdtved/+uVfJcd8ZZlwtwnic0w0YSnSUWlJS5XecYu9pJ7N9Vqb4bgvLMRWUG6iv/ej2/wAB9Nf8Nd/AL/obb7/wm7//AOMUf8Nd/AL/AKG2+/8ACbv/AP4xXzF9vP8AeH50fbz/AHh+dfzN/wATOca/9AuH/wDAav8A8tPX/wCIc5T/AM/Kn3x/+QPp3/hrv4Bf9Dbff+E3f/8Axij/AIa7+AX/AENt9/4Td/8A/GK+Yvt5/vD86Pt5/vD86P8AiZzjX/oFw/8A4DV/+Wh/xDnKf+flT74//IH07/w138Av+htvv/Cbv/8A4xR/w138Av8Aobb7/wAJu/8A/jFfMX28/wB4fnR9vP8AeH50f8TOca/9AuH/APAav/y0P+Ic5T/z8qffH/5A+nf+Gu/gF/0Nt9/4Td//APGKP+Gu/gF/0Nt9/wCE3f8A/wAYr5i+3n+8Pzo+3n+8Pzo/4mc41/6BcP8A+A1f/lof8Q5yn/n5U++P/wAgfTv/AA138Av+htvv/Cbv/wD4xVO5/bY/Z1t9VTQ4/E2tXV5JbtOtrp/gvVrmTygwUuVitWIXLAZPGSK+bvt5/vD860fgHqZh/aeaXf8A8yHcD/ydt6+98M/HLibjTjLD5RjKFGNOoptuCmpe7CUlbmqSW6102PD4i4Ly/J8pniqU5uUbbtW1aXSK79z6D/4bK+B3/U5f+Gx17/5Co/4bK+B3/U5f+Gx17/5Cqf8At7/prR/b3/TWv6l0PzUg/wCGyvgd/wBTl/4bHXv/AJCo/wCGyvgd/wBTl/4bHXv/AJCr5M/4K9/t1/tffsJ/BrQvj3+zb8BdP8c+HdL1xJPiOs00hubPTB95oo06ZzzJkhMDK4Oa9g/Yr/bx+BX7enwI0v4/fAPxYl9pt9GFvrGRgLnTbnHz206ZyrqfwYcjg0Aep/8ADZXwO/6nL/w2Ovf/ACFR/wANlfA7/qcv/DY69/8AIVT/ANvf9Na8V/by/wCCh/wG/wCCePwC1H49fHfxIIoYVaLRdFgkX7XrF3jK28Ck8k8bmxhByewIB7H/AMNlfA7/AKnL/wANjr3/AMhUf8NlfA7/AKnL/wANjr3/AMhV8uf8Ejv2zv2wv20P2c7r49ftYfA3TfAsOta1NP4HtbSSRZrrSG5hklikyUIztD5G9QG2jOT9Vf29/wBNaAIP+Gyvgd/1OX/hsde/+QqP+Gyvgd/1OX/hsde/+Qqn/t7/AKa0f29/01oA9MooopAFFFFABRRRQAVi/Efw9L4v+HmveE4GCyapo91aIzdAZImQfzraoPIxQB81/wDBObxzpmvfs+eHvARljh1rwDZx+FPEels486wurBBb7ZF/hLxpHKvqsqmvpQHcMivBfjl+w98PviL8QJPjL4L8S+KPAPjWaFIrzxZ4B1r7Dc30aZ2JdRMsltehMnaLmGXbk7cZrlx+y5+1BAuwf8FRvj8q9FVfD/gBsfi3hYk/iSa0kubYD6ior5d/4Zj/AGoP+kpXx/8A/Cd+H3/zK0f8Mx/tQf8ASUr4/wD/AITvw+/+ZWp5GB9RUV8u/wDDMf7UH/SUr4//APhO/D7/AOZWj/hmP9qD/pKV8f8A/wAJ34ff/MrRyMD6ior5d/4Zj/ag/wCkpXx//wDCd+H3/wAytH/DMf7UH/SUr4//APhO/D7/AOZWjkYH1FRXy7/wzH+1B/0lK+P/AP4Tvw+/+ZWj/hmP9qD/AKSlfH//AMJ34ff/ADK0cjA+oqK+Xf8AhmP9qD/pKV8f/wDwnfh9/wDMrR/wzH+1B/0lK+P/AP4Tvw+/+ZWjkYH1FXF+Ef2i/gR49+J/ib4KeDvi1oOoeMPBs0UXinwvb6in2/TDLDHNG0sBO8I0csZWTGw5wDkEDxH/AIZj/ag/6SlfH/8A8J34ff8AzK1+Uv7eH/BG3/gon+1J/wAFSrj9oDwb+0X458KaL4N03T9Oh+NPio6RZ65qlwqNM0mnQeH7PTw6Ri4WLzJljYtHIvmuFVFqNOUnYNj9WPDmqpafFr4qR+Z/zUAn/wApen10X9vJ/wA9a+d/2WrLxj4G0PxP4U+IHxQ1vxjrFj4nEWoeJ/EUdut7qDjTrIeZKLeKOPOMAYXJABYu2529Q/t//pqaiS5ZWA7j+3k/561xf7QXwa+EH7Ufwk1j4H/HDwla654c1y1aG8srqMHb6SI3VJFPKsOQaj/t/wD6a0f28f8AnrSA/KX4WfF79oX/AINy/wBoa3+A/wAcdV1Txl+yx4y1Rv8AhFfFBRpJvDMrn7jD+Ern54+jgb15yK+sv+Cmf/BbD4YfsofCrQ/D/wCzReWfxF+KXxFs4z8PNB0VxdIUnGIryUJnKZPyp1cjnABrzn/guD+3n8BPhj8B7r9knWfhvY/E34gfES3Fn4f8AtGZjE7nbHdy+Wd8ZVjlNpDMw4IGTX53/sXfCf4if8EKv2pvBvxz/b6/Z507UfDPjPSo7bT/ABlYySXkng+4k5K4PypKoOHXBbbnY3UEA/Sz/gkp/wAErPEPwW8V3n7ef7eOtt4x+P3jBmu57jUnE6eHUk58mLPHnYO0sOEHyrxkn9Cv7eT/AJ615r4P+J3hfx/4XsPGvgnxJa6ppOqWqXOn6hYzCSK4iYZV1YdQRWl/b/8A01NAHcDXkH/LWga8g/5a1w/9v/8ATU0f29/00NAHlfhPXfJu/ESBv+Z01o9fXUJzWwPER7t+teZ6Pr3k6n4gTf18WasevrfTVe/4SQ/89P8Ax6vhalblqSXm/wAz+l8Fl/tMHSlbeMfyR33/AAkZ/vfrR/wkR/vfrXA/8JIf+en/AI9R/wAJIf7/AP49U+3On+zPI77/AISI/wB79aP+EiP979a4H/hJD/f/APHqP+EkP/PT/wAeo9uH9meR33/CRH+9+tH/AAkR/vfrXA/8JIf+en/j1H/CSf8ATT9aPbh/Znkd9/wkR/vfrR/wkR/vfrXA/wDCSf8ATT9aP+Ek/wBv9aPbh/ZnkdJ4Rvf+KfhO7+KT0/56NWl9t/2v5Vy/hG+H/CO2+fRj/wCPGtL7ctf5n5lJf2hW1+3L82e5DDe4jW+2/wC1/Kj7b/tfyrJ+3LR9uWuPmXcr6qa323/a/lR9t/2v5Vk/blo+3LRzLuH1U1vtv+1/Kj7b/tfyrJ+3LR9uWjmXcPqprfbf9r+VJJe/u2+b+H2rK+3LTZL0FGAJ+77Ucy7h9WPeP2Cv+RXtf+vOH/0EV9Ha7oNnrlq1tdxblYdK+K/2FPhb8R4fhbqnhNP2ofHn2zWbeN9O8Sf2foH2zQlKg7LVf7L+zuo/6eYbg+pNfRerfCj4rap4f8M6HZ/tc/ETTrnw9Kr6prFnpvhxrnxKo/gvRLpDwxqe/wBkjtT6EV/rdlStlOFs/wDl3T/9IifyTjP98q/4pfmzqdO+EuhWNz56wZ9N3appPhhoTxSRm0H7z73y1hR+CPiFH8Urj4kt+0H4xfSZ7H7PH4Bax0b+ybeTGPtCSDTxfGTvhrto8/wY4rI074QfFuw8Ca34Muf2x/iVeahrF0ZrDxZc6X4ZGoaIhOfJtUj0dbV0HQG4gnfH8RPNehzT7nNodZpvwh0HTrxLyG3+ZD8vHStzW/C1prVsttdpuVe1ef6v8LPinqdh4ZsbP9rL4hadJ4eZTql5Z6b4daTxKB1F6JdJdIwe/wBkW1PoRV628DfEO3+JV/8AEGX9ofxlNpd7ZmC38ESWOi/2XYSYx58Ui6eL0yd8S3MiZ/gxxSvNvcNDe/4Vhom/f9m/hxUn/CudG+x/YxbfLuziuGs/g98XLT4eap4En/bL+JdzqWpXhns/GU2l+GBqWlpnPkQImjLZsnbM1tK+P4yeat6t8MfihqbeGWs/2qvH+n/8I/t/tRbPTfD7f8JLj/n983Sn8vPf7J9l9sUuaQaG14p8LeA/BHh3UvHHiu4js9M0fT5rzUryYHZb28SF5JDjJwqqScZ6Vx1/4++DnxW/ZI8VfEn4GfEXRfFHh3VPBmpSafrGganHdW0y/ZZPuvGSuR3HUHggGrHjnwh8SdCHjT4lH4y+PPEmn3nhu7Wx+GsWl6C1nBIIWIFoTpouZJnPyqtzPNEWYAoRkV+IH7BH/BGf/gol+yfrvjH9ov4lftN+JfhVpviCz1K+vPhn4XvLa6m1G3kjkYRamoRrFDtOCscUhUE7GhYDEzU505LyZ0YOyxdP/EvzR+j3hvxCV8O2C7ullF3/ANgVe/4SI/3v1rzfQvEW3RLNfM6WsY6/7Iq1/wAJIf8Anp/49XwH1g/pl5Zrsd9/wkR/vfrR/wAJEf7361wP/CSH/np/49R/wkh/56f+PU/rAf2X5Hff8JEf7360f8JEf7361wP/AAkh/wCen/j1H/CSH/np/wCPUfWA/svyO+/4SI/3v1o/4SI/3v1rgf8AhJD/AH//AB6j/hJD/wA9P/HqPrAf2Z5Hff8ACRH+9+tYvxG17zvA2qRbvvWrDrXN/wDCSH/np/49Wb4x8QGbwvfR7/vQkda7crrXzKgv78f/AEpHmZ1lvLk2Jdtqc/8A0ln0J/b6f3/8/nR/b6f3/wDP51w39vL/AM9/1FH9vL/z3/UV/RX1TyP4z+sHc/2+n9//AD+dH9vp/f8A8/nXDf28v/Pf9RR/by/89/1FH1TyD6wdz/b6f3/8/nR/b6f3/wDP51w39vL/AM9/1FH9vL/z3/UUfVPIPrB3P9vp/fo/4SCP+/8A5/OuG/t5f+e/6ik/t5f+e/6ij6p5B9YO6/4SCP8Av/5/Oue+KOtJP4NkiD9b6z/S6irG/t5f+e//AI9WV411hbjQDF5uc3lrxn/p4jr4nxKwqj4c5y7bYTEf+mZnu8L1ubibAr/p9S/9Liaf23/a/lR9t/2v5Vk/blo+3LX+MXMu5/cH1U1vtv8Atfyo+2/7X8qyfty0fblo5l3D6qa323/a/lR9t/2v5Vk/blo+3LRzLuH1U1vtv+1/Kj7b/tfyrJ+3LR9uWjmXcPqprfbf9r+VZfii581dPTd/zFIf6037ctc38UdE1jxloFvoHh/4gax4XuptRh8vWtDhs5LqDBP3FvLe4hOf9qNq+o4GcZca5Yrr/eKP/pyJ5eeYflyXEv8A6dz/APSWffX7PfPgW0B/54it7xP4C0vxG266iyw6N6V4t8Ifhp8SdY8I+E7rTP2nPHmjpoKr/adtp+n6CyeIsdrwz6ZIyZ7/AGRrb2xXf2PgP4i2fxA1fxxcftF+NLrTdUtTDZ+D7ix0QadpD4x51u6act2zjrie4mTPVccV/qvU5lUbR/Ka+E3rP4WaFa2jWwts7uue9Nv/AIT6BexrG9r937px0rjYPg38X4fhlefDuT9s/wCJk2rXd59og8dSaV4Y/tWzjzn7PHGujCyMeOMyWryY/jzzV/V/hv8AE3VNV8M6pZftQePNOh8PxquqafZ6foDQ+JSP4r0y6Y8kZPf7G9sPQCp5p9x6HXeF/h9pfhiVpLGLbu+970/V/AmmatffbbmLc3TdXKab4A+I2n+ONe8YXX7R/ja+0/WrUxaf4VurHQxYaExH+utXj05Lp3HXFzPcJnqp6VkJ8GPjAvwpm+Gh/bT+JzaxLffaE+IDaV4X/teKPOfsyxjRfsBjxxlrQyY/jzzSvK9w0O3X4X6GE2G1/izU118OdHupVlkt8lVx0rltW+HnxJ1TxP4Z8RWX7THjjT7Tw9CseqaFZ6foTWviVgP9ZetLprzxse/2OW1X0Ap2kfD74kaX4v8AEnim9/aU8b6hZeILdotL8O3lhoa2fhtiP9bZNFpqXDsOoF3Ncr6qelHNMNDP+L/jT9nL9naz0K7+OPxL0HwjbeJ/EMei6He+IdQjtYbvUZIZpo7YSyEIJHSCUqGI3FdoyxUHh/28mt9K0H4bm2lVlXxxJtYNnP8AxKNQFfNv/BcL9jT9qn9or/gmV/wxp8J/Efjb4weMvFXjzThH4m8Tw+HbNtPt4ne6knvpLbT7SGC1jSAorW0cdw0ksSF5Fd45Pmz9jj9hH9qb/gnv8JPCHw1/aA/bK8R+PbSPxEVsPB27zNC0CQabecWclwhuePmACtDERIxaAttdeTMeZ4CpJ9F+qPoeE4qXEWHh3bX4M+w/+EiP979aP+EiP979a4H/AISQ/wDPT/x6j/hJD/z0/wDHq+K9uf0B/Znkd9/wkR/vfrR/wkR/vfrXA/8ACSf9NP1o/wCEk/6afrR7cP7M8jvv+EiP979aP+EiP979a4H/AIST/b/Wj/hJP9v9aPbh/Znkd9/wkR/vfrR/wkR/vfrXA/8ACSf7f60f8JJ/t/rR7cP7M8jvv+EiP979aydW1f7V4o007vu29z/7Trl/+Ek/6afrRYax9q8S2uWztt5v5pX5v4u1ubw6x68of+nIG2Hy/wBnWjKx2323/a/lR9t/2v5Vk/blo+3LX8F8y7nsfVTW+2/7X8qPtv8AtfyrJ+3LR9uWjmXcPqprfbf9r+VH23/a/lWT9uWj7ctHMu4fVTW+2/7X8qPtv+1/Ksn7ctH25aOZdw+qmt9t/wBr+VT/AAd1JYP2j/ML/wDMkXA/8nLesL7ctRfDnUxB+0EsgbH/ABRtwP8Aybgr9m8AJf8AG0sFr9mr/wCmpnxfiBQ5OFaz84f+lxPpf+3k/wCetH9vJ/z1rh/7f/6a0f28f+epr/RI/nU7LUb3TNY0+fSdWtorq1uoWiube4jDxyxsMMrKeCCOCDwRX5D/ALWv7K/x9/4Ij/tCXv8AwUR/4J06Xcat8JdXuQ/xW+FMbsYbSItlpolGcRjJKuBmI8HKmv1G/t4/89TXjv7cf7cfwG/Yn+Amq/Fb4+6pDJYPbyW1joJVHm1qZlI+yxxtwwYHDEgqqkk+hAMPXP8AguL+w3pH7DEf7dkXxHjuNGuIfJs/DccqjU5dTC5On+VnKyA/ebG0L83IIB+Q/wBib9jP49f8Fd/2hLP/AIKb/wDBTzS5LbwXZyiX4S/CW4VhbLbBt0U00Tf8sujYYbpm+ZvlwD+dun/sa/tFeDryx/4K6az+w/oc3wlPjRtab4T/AGi4PkaazbluDASWEHI2sxIyo3JsIB/fr9j39tD4H/tl/A7SfjN8BfEMVxpNzCsU2ngKk2mTKo3W0sY/1bp0x0IwRkEUAfQ1tqtpZW8dpZqsUMSBI441CqigYAAHQACpP7eT/nrXD/29/wBNDR/b3/TQ0Adx/byf89aQ65GTkyVxH9vf9NDQdf8A+mtAH11RRRQAUUUUAFFFFABRRRQAEAjBFM8iE8mMU+igBn2eH/nmKPs8P/PMU+ii4DPs8P8AzzFH2eH/AJ5in0UXAZ9nh/55ij7PD/zzFPoouAz7PD/zzFH2eH/nmKfRRcBn2eH/AJ5ij7PD/wA8xT6KLgM+zw/88xWfq3hbTNYK/a4FYKc4Za06KfM0B8I+J7mHRPjv8TtOtkVY08aLtUDp/wASyxo/t41g/tDXHirwH+0P8RNQ1f4eeLPsOo+Jo7qx1Cz8L3k9vPH9gtE3LLHGVPzRsvB4K1xP/C4bT/oVvFn/AISN9/8AGqfLKWtgueqf2+a+TP8Agpx/wVR0P9i3wza/DP4YaUvij4teKlEHhTwtaqZGiZztW4mVedoP3V6sfbJr1/8A4XDaf9Ct4s/8JG+/+NV57f8Aw0/Zw1T492v7T2o/ALWJvHVnp/2O28QP4LvjKkeeuPK2lwOA5G4AkAijkn2YXR5X/wAEwf8Agmxr/wAI/FF1+2v+2nqh8WfG7xUxupZ9QbzV0FXH+rjzwJcHBI4QfKvc19efHP4V/C/9pH4Xat8G/jJ4TttZ0DWbcxXdpcLyp/hkQ9UdTyrDkH8RXM/8LhtP+hW8Wf8AhI33/wAao/4XDaf9Ct4s/wDCRvv/AI1RyT7MLo/PT4ZfFT47f8EEvjnD8E/jJqOo+LP2bvFepN/wjPiRlaSTw9I7fdb+7jPzx9CPnXuD+p/hT4k+HPHXhqx8Y+D9attS0vUrVLixvrOQPHPEwyrKR1BBrxP4yQ/CX9oD4d6h8KfjB8JvEOuaDqkey7sLzwff4PoykRBkYHoykEetXvh1rnw++E3gnTfhz8Ofhn4i0nRdHtVttP0+z8HXyxwxqMAD91+vU0ck+zC6Pbv7eNH9vGvK/wDhcNp/0K3iz/wkb7/41R/wuG1/6FbxZ/4SN9/8ao5J9mF0cXHr3la3ry78Z8Takfvet3Kam/4SL/pp/wCPVx88fjSTVtTvIfAHiPy7rV7u4hY6DcglJJndTgpkZBHB5FHleN/+hA8Rf+CK4/8AiK/N8RQxv1idqct39l9/Q/r7KcZkn9lYfnxNNPkhdOcbp8q8zsP+Ei/6af8Aj1H/AAkX/TT/AMerj/K8b/8AQgeIv/BFcf8AxFHleN/+hA8Rf+CK4/8AiKx9hjv+fUv/AAF/5HofXMh/6Cqf/gcf8zsP+Ei/6af+PUf8JF/00/8AHq4/yvG//QgeIv8AwRXH/wARR5Xjf/oQPEX/AIIrj/4ij2GO/wCfUv8AwF/5B9cyH/oKp/8Agcf8zsP+Ei/6af8Aj1H/AAkX/TT/AMerj/K8b/8AQgeIv/BFcf8AxFHleN/+hA8Rf+CK4/8AiKPYY7/n1L/wF/5B9cyH/oKp/wDgcf8AM7D/AISL/pp/49R/wkX/AE0/8erj/K8b/wDQgeIv/BFcf/EUeV43/wChA8Rf+CK4/wDiKPYY7/n1L/wF/wCQfXMh/wCgqn/4HH/M9j8JXefDlqQf+WZP6mtH7X71zXhuPVbXQrWC50PUUkWEblaxlyPbpV3zL/8A6A+of+AMv+Ff575hwzxVLH1WsvrNc0tfZT7v+6c9PNMj5FfFU9l9uP8AmbH2v3o+1+9Y/mX/AP0B9Q/8AZf8KPMv/wDoD6h/4Ay/4Vx/6r8Wf9C6v/4Kqf8AyJX9q5H/ANBVP/wOP+Zsfa/ej7X71j+Zf/8AQH1D/wAAZf8ACjzL/wD6A+of+AMv+FH+q/Fn/Qur/wDgqp/8iH9q5H/0FU//AAOP+Zsfa/ej7X71j+Zf/wDQH1D/AMAZf8KPMv8A/oD6h/4Ay/4Uf6r8Wf8AQur/APgqp/8AIh/auR/9BVP/AMDj/mbH2v3oa7+U89qx/Mv/APoD6h/4Ay/4UGS/Ix/Y+of+AMv+FH+q/Fn/AELq3/gqp/8AIh/auR/9BVP/AMDj/mfUv7Bag+GLUkf8ucP/AKCK+nfKj/u18zfsJW1xaeHLeC5gaN0tYlZXXBBCjivpuv8AVXLYyp5ThoyVmqcE1/25E/j/ABTUsXUa/ml+bG+VH/do8qP+7TqK67nON8qP+7R5Uf8Adp1FFwG+VH/do8qP+7TqKLgNaCNl27a87/aH8I6U3wT8bapLbq0i+EdSKsR0/wBFkr0auV+Oum6hrPwS8YaPpNlJc3V14X1CG1t4ULPLI1u6qigckkkADuamo5ezduzOjCNLFU2/5l+Z+WOk+IdulWq+Z0t0H3v9kVY/4SL/AKaf+PVy914X+KnhqOLT9e+FPiqzmjjVWjuPDtypGB7pVfyvG/8A0IHiL/wRXH/xFfmvscd/z6l/4C/8j+xljMh/6Cqf/gcf8zsP+Ei/6af+PUf8JF/00/8AHq4/yvG//QgeIv8AwRXH/wARR5Xjf/oQPEX/AIIrj/4ij2GO/wCfUv8AwF/5B9cyH/oKp/8Agcf8zsP+Ei/6af8Aj1H/AAkX/TT/AMerj/K8b/8AQgeIv/BFcf8AxFHleN/+hA8Rf+CK4/8AiKPYY7/n1L/wF/5B9cyH/oKp/wDgcf8AM7D/AISL/pp/49R/wkX/AE0/8erj/K8b/wDQgeIv/BFcf/EUeV43/wChA8Rf+CK4/wDiKPYY7/n1L/wF/wCQfXMh/wCgqn/4HH/M7D/hIv8App/49VTXde87SLiLzPvR4+9XNeV43/6EDxF/4Irj/wCIpJLLxzdr9mXwD4gBfjLaJcAfmUr0Mqo4yOaUHKnJLnh0f8yPI4gxWSSyHFqGJpt+yqWSnFtvkdktdz2/+3v+mv60f28P+ev61xf2vxF/0K+sf+Cqf/4mj7X4i/6FfWP/AAVT/wDxNf1l7TB/8/I/ej/PjkxP8kvuZ2n9uj/np+tH9uj/AJ6frXF/a/EX/Qr6x/4Kp/8A4mj7X4i/6FfWP/BVP/8AE0e0wf8Az8j96DkxP8kvuZ2n9uj/AJ6frR/bw/56/rXF/a/EX/Qr6x/4Kp//AImj7X4i/wChX1j/AMFU/wD8TR7TB/8APyP3oOTE/wAkvuZ2n9vf9Nf1o/t7/pr+tcX9r8Rf9CvrH/gqn/8AiaPtfiL/AKFfWP8AwVTf/E0e0wf/AD8j96D2eJ/kl9zO0/t7/pr+tVNb1j7RZLF5mc3UH/o1K5b7X4i/6FfWf/BVN/8AE0qP4iu5ooT4Z1ZR56MzSabMAAGB7r7V8T4lRo1/DnOadGSlOWExCUU0226M0kktW29Elq2e9wr7WnxRgZVItRVak22rJLnjdt9EjuftfvR9r96x/Mv/APoD6h/4Ay/4UeZf/wDQH1D/AMAZf8K/xf8A9V+LP+hdX/8ABVT/AORP76/tXI/+gqn/AOBx/wAzY+1+9H2v3rH8y/8A+gPqH/gDL/hR5l//ANAfUP8AwBl/wo/1X4s/6F1f/wAFVP8A5EP7VyP/AKCqf/gcf8zY+1+9H2v3rH8y/wD+gPqH/gDL/hR5l/8A9AfUP/AGX/Cj/Vfiz/oXV/8AwVU/+RD+1cj/AOgqn/4HH/M2PtfvR9r96x/Mv/8AoD6h/wCAMv8AhR5l/wD9AfUP/AGX/Cj/AFX4s/6F1f8A8FVP/kQ/tXI/+gqn/wCBx/zNj7X71FNP5moaev8A1EI/61meZf8A/QH1D/wBl/wqbTIdRutb09F0m9G28RmaSzdQAO+SK+m4K4b4mpcY5dOpgK0YqvRbbpzSSVSN2242SXVnk59meTTyPFRhiabbpzslON2+V6LU+/v2fEQ+BbXK9YhXfeVH/drg/wBn1WXwNaKwx+5Fd9X+n1X+Iz+TI7DfKj/u0eVH/dp1FZ3GN8qP+7R5Uf8Adp1FFwG+VH/do8qP+7TqKLgQXdhDdwmJl4bivkz/AIKg6Hpvhf4d+B72yhVGbx0dzKMZzpl/X11Xy1/wVb8IeN/Fvwh8JnwP4O1XWprDxktxdw6Rp8lxJFD9gvE8wrGCQu51GemWFcOZ88svqKOrsfT8GTpU+KcJKrJRjzq7bslo92z40/4SL/pp/wCPUf8ACRf9NP8Ax6uTns/H1rK0Fx8OvEkbrwytoNyCP/HKj8rxv/0IHiL/AMEVx/8AEV8D7HHf8+pf+Av/ACP6n+uZD/0FU/8AwOP+Z2H/AAkX/TT/AMeo/wCEi/6af+PVx/leN/8AoQPEX/giuP8A4ijyvG//AEIHiL/wRXH/AMRR7DHf8+pf+Av/ACD65kP/AEFU/wDwOP8Amdh/wkX/AE0/8eo/4SL/AKaf+PVx/leN/wDoQPEX/giuP/iKPK8b/wDQgeIv/BFcf/EUewx3/PqX/gL/AMg+uZD/ANBVP/wOP+Z2H/CRf9NP/HqP+Ei/6af+PVx/leN/+hA8Rf8AgiuP/iKPK8b/APQgeIv/AARXH/xFHsMd/wA+pf8AgL/yD65kP/QVT/8AA4/5nYf8JF/00/8AHq1PBurfa/ESfNnbbyd/da878rxv/wBCB4i/8EVx/wDEV0nwzt/FA1ia41DwhrVuiQ4DXGkzJkk9sqM9K+B8UMDmmK4DxtKjQnKTUbJRk2/3kHokrmdTHZHGN1iqe6+3HuvM9Q+1+9H2v3rH8y//AOgPqH/gDL/hR5l//wBAfUP/AABl/wAK/in/AFX4s/6F1f8A8FVP/kSf7VyP/oKp/wDgcf8AM2PtfvR9r96x/Mv/APoD6h/4Ay/4UeZf/wDQH1D/AMAZf8KP9V+LP+hdX/8ABVT/AORD+1cj/wCgqn/4HH/M2PtfvR9r96x/Mv8A/oD6h/4Ay/4UeZf/APQH1D/wBl/wo/1X4s/6F1f/AMFVP/kQ/tXI/wDoKp/+Bx/zNj7X70fa/esfzL//AKA+of8AgDL/AIUeZf8A/QH1D/wBl/wo/wBV+LP+hdX/APBVT/5EP7VyP/oKp/8Agcf8zY+1+9UvCOpfZvjnHJ6+E5x/5Mw1U8y//wCgPqH/AIAy/wCFYUms6j4Z+J9v4huvC+vSWraDNb+bZ6Hcz4kM8bBSEQkcKevpX7B4D5DxBg/E7B1cTgqtOCjVvKVOcUr0ppXbilq9PU+J8RMwyqtwnXhRrwlJuGilFv449Ez3M6+aP+EgNeV/8LhtP+hW8Wf+Ejff/GqP+Fw2n/QreLP/AAkb7/41X+gXJPsz+bbob+2L+3H8Hv2JPgzffGP4wayscMKmPS9MhYfaNSucfLBEvcnueijk18I/si/sn/Gv/gqR8dLX/god/wAFEdMkg8I2cvmfDL4YzbhbrAGzHLLGf+WfQ88yHk/LwfrH46/Dz9nf9pdNFj+OXwJ1rxEvh/U0v9J+3eDb8+TMvuIhlT3U5U45BrvLf4s6ZaW8dra+D/FMcUaBI44/B98qooGAABDwAO1HJPswuj0+a40u50ptCuNMt5LGS3MElm8IMTRFdpjKdCu3jbjGOK/Ln9pv9m343f8ABHr47Xn7eH7Bek3Gp/C7VrhX+JnwyjZmitYy3zSxLziMZJVxzGTg5U8/fX/C4bT/AKFbxZ/4SN9/8aqK9+Kej6nZTadqPgvxPcW9xE0dxbzeDr1kkRhhlZTDggg4IPBFHJPswujR/ZS/bK+EH7ZHwc0341fBjxAl5p18u25tm4nsbgAb4Jlz8rqT+IwRkEGvSP7eNfNvwC8Bfs+/sv6FqHhv4D/A7XPDtnquqS39/HZeD9QzNO5ySS0ROB0Vc4UAAYFd9/wuG1/6FbxZ/wCEjff/ABqjkn2YXR6p/bxoPiBgeAK8r/4XDa/9Ct4s/wDCRvv/AI1Sf8Lhtu3hbxX/AOEjf/8Axqjkn2YXR+odFFFSAUUUUAFFFFABRRRQAUUUUAFFFFABRRRQAUUUUAFFFFABRRRQAUUUUAc/4x+H+k+MYfs+pwLIv+1XL/8ADNfgT/oGJXpFFaRqVIqyYrI83/4Zr8Cf9AxKP+Ga/An/AEDEr0iin7ap3Dlj2PN/+Ga/An/QMSj/AIZr8Cf9AxK9Ioo9tU7hyx7Hm/8AwzX4E/6BiUf8M1+BP+gYlekUUe2qdw5Y9jzf/hmvwJ/0DEo/4Zr8Cf8AQMSvSKKPbVO4csex5v8A8M1+BP8AoGJR/wAM1+BP+gYlekVmeNPGfhT4deENU8f+PPENppOiaJp819q+qX8wjgs7aJC8ksjHhVVVLEnoBR7ap3Dlj2OK/wCGa/An/QMSj/hmvwJ/0DErs/BXjjwh8R/COl+PfAfiK01bRdc06G/0fVLGYSQ3lrNGJIpo2HDIyMrAjqCK1cj1o9tU7hyx7Hm//DNfgT/oGJR/wzX4E/6BiV6RuHrRuHrR7ap3Dlj2PN/+Ga/An/QMSj/hmvwJ/wBAxK9I3D1oo9tU7hyx7Hm//DNfgT/oGJR/wzX4E/6BiV6RRR7ap3Dlj2PN/wDhmvwJ/wBAxKP+Ga/An/QMSvSKKPbVO4csex5v/wAM1+BP+gYlH/DNfgT/AKBiV6RRR7ap3Dlj2PN/+Ga/An/QMSj/AIZr8Cf9AxK9Ioo9tU7hyx7Hm/8AwzX4E/6BiUf8M1+BP+gYlekUUe2qdw5Y9jzf/hmvwJ/0DEo/4Zr8Cf8AQMSvSKKPbVO4csexzPgz4Z6H4LyulW6xqf7tdNRRUSlKWrGFFFFSAUUUUAFFFFABTZEEiFD3p1FAHFeJ/gp4X8VXxvtTsldvU1mf8M1+BP8AoGJXpFFae2qLqKyPN/8AhmvwJ/0DEo/4Zr8Cf9AxK9Iop+2qdw5Y9jzf/hmvwJ/0DEo/4Zr8Cf8AQMSvSKKPbVO4csex5v8A8M1+BP8AoGJR/wAM1+BP+gYlekUUe2qdw5Y9jzf/AIZr8Cf9AxKP+Ga/An/QMSvSKKPbVO4csex5v/wzX4E/6BiUf8M1+BP+gYlekUUe2qdw5Y9jzf8A4Zr8Cf8AQMSj/hmvwJ/0DEr0iij21TuHLHseb/8ADNfgT/oGJQP2avAZ66bHXpFFHtqncOWPY83P7NXgMdNOjo/4Zr8Cf9AxK9Iope2qdw5Y9jzf/hmvwJ/0DEo/4Zr8Cf8AQMSvSKKftqncOWPY83/4Zr8Cf9AxKP8AhmvwJ/0DEr0iij21TuHLHseb/wDDNfgT/oGJR/wzX4E/6BiV6RRR7ap3Dlj2PN/+Ga/An/QMSj/hmvwJ/wBAxK9Ioo9tU7hyx7Hm/wDwzX4E/wCgYlH/AAzX4E/6BiV6RRR7ap3Dlj2PN/8AhmvwJ/0DEpU/Zt8CxuHXTEyOa9Hope2qdw5Y9jO8OeHbTw7ZLZWabVUYUVo0UVm3fUYUUUUAFFFFABRRRQAVX1PT4tStWtpR8rLjFWKKAPO7v9nXwTe3LXM2nIWdsmo/+Ga/An/QMSvSKK09tU7i5Y9jzf8A4Zr8Cf8AQMSj/hmvwJ/0DEr0iin7ap3Dlj2PN/8AhmvwJ/0DEo/4Zr8Cf9AxK9Ioo9tU7hyx7Hm//DNfgT/oGJR/wzX4E/6BiV6RRR7ap3Dlj2PN/wDhmvwJ/wBAxKP+Ga/An/QMSvSKKPbVO4csex5v/wAM1+BP+gYlH/DNfgT/AKBiV6RRR7ap3Dlj2PN/+Ga/An/QMSj/AIZr8Cf9AxK9Ioo9tU7hyx7Hm/8AwzX4E/6BiUf8M1+BP+gYlekUUe2qdw5Y9jzf/hmvwJ/0DEo/4Zr8Cf8AQMSvSKKPbVO4csex5v8A8M1+BP8AoGJR/wAM1+BP+gYlekUUe2qdw5Y9jzf/AIZr8Cf9AxKP+Ga/An/QMSvSKKPbVO4csex5v/wzX4E/6BiUf8M1+BP+gYlekUUe2qdw5Y9jzf8A4Zr8Cf8AQMSj/hmvwJ/0DEr0iij21TuHLHseb/8ADNfgT/oGJR/wzX4E/wCgYlekUUe2qdw5Y9jzf/hmvwJ/0DEo/wCGa/An/QMSvSKKPbVO4csewUUUVkMKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAK+GP+DkH4dWfxA/4I9fFy9ufE+u6XJ4b0uPV7Y6Hqj2v2p1fyTb3G3/XW7JO+6I8MQuelfc9cX+0T8Cfhz+058D/E/wAAPi5op1Dw14s0iXTtYs1kKM8LjqrD7rA4YHsQDQgP57fjf8X/AI2fss/sefsX/sffBLx58YtR8KfGDSE8R+NIPCPiTzfEOo403Tn/ALK025kK/ZYFMrFY1I2LtxnGDsP+1f8AtyeBP+CY/wC2F8KvEUvxi8L+GfDv9gX/AMI9Y+IuuH/hJtKSa9iF5aS3kEm9sMgKncCVds8NtH6MXH/BA39kWT9n3Tf2V72x8XX3h/wzrT6r4R1y98UzPq+h3DIkeLS7wGhQJGihFG0bRgZAIm8H/wDBDH9l3Q/2cvGn7Kj6L4ik0P4gSwyeMNeuteefWNTeKRHjaS6kBJK7ABx0J9a7OWm3cjU/PfVdO/az8B/tefsq+FdJ/b3+KjL+0f8ADGVPiUs+vGSOAR6fbyyiwRgVtm2S7EkALq0YcsxYisHUv2of2mvhJ/wTt/bk/Zkh/aO8capD8KPiNpEHgfxZqniKdtWsra4vNzRm7DCQ4WJQeectwAcV+s2q/wDBKb4I618S/hL8WtQt9a/tj4IaTNpvgTZqWIhBLbRW7faF2fvW8uFOcrg5Pevmb/gp7/wSKXwp+w5+0Nf/ALJ3w68TeJvHXxl1bStT17RY7kXLXFxBdL/x7x7V2KEeQkZPA9qLUtf66f5j1Pm8+Ovj3+xZ+0d+xv8AH3wN+2x8QPHeofGS+0vTPiB4e8Q+JGvNOv7e5toSz21t92FYw2wNgsThic5z+7fgPxSPFGlLeY+8Pyr8yf2Jv+CMH7PHw08W+B/2grnwN4ibxroHhezFppWva9Nc2Wg3rWyCc29u+RA/mbsgHAboOBX6XfDPwzP4b0VLW4+9jmprRhGIROmooorlKCiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAaYIScmMUCCEHIjFOoouA3yIT1jqOezimjaMr1GKmoouwMrS/CmnaZctcwQqrOfm961QAowBRRTbuAUUUUgCiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKAP/2Q==)

**PROGRAM:**

#include <iostream>

#include <stdlib.h>

using namespace std;

struct node {

int data;

struct node \*next;

struct node \*prev;

}\*head,\*tail;

void insert(int data){

struct node \*new\_node;

new\_node = (struct node \*)malloc(sizeof(struct node));

new\_node->data=data;

new\_node->prev=NULL;

new\_node->next=NULL;

if(head == NULL)

{

head=new\_node;

tail=new\_node;

}

else

{

tail->next = new\_node;

new\_node->prev=tail;

tail=new\_node;

}

}

void deleteAtBeg(){

if(head == NULL)

{

cout<<"Underflow";

}

else

{

struct node \*temp;

temp=head;

head=head->next;

temp->next=NULL;

head->prev=NULL;

free(temp);

}

}

void deleteAtEnd(){

if(head == NULL)

{

cout<<"Underflow";

}

else

{

struct node \*temp;

temp=tail;

tail=tail->prev;

temp->prev=NULL;

tail->next=NULL;

free(temp);

}

}

void DeleteAtLoc(int pos){

if(head == NULL)

{

cout<<"Underflow";

}

else

{

struct node \*temp;

temp=head;

for(int i=0;i<pos-1;i++){

temp=temp->next;

}

temp->next=temp->next->next;

temp->next->prev=temp;

}

}

void display(){

struct node \*temp;

temp=head;

while(temp!=NULL){

cout<<temp->data<<" ";

temp=temp->next;

}

}

int main(){

int choice;

insert(3);

insert(4);

insert(7);

insert(9);

insert(10);

insert(66);

cout<<"The doubly linked list is:"<<" ";

display();

while(1){

cout<<"\n\nSelect an Option";

cout<<"\n1.Deletion At Beginning\n"<< "2.Deletion At End\n"<<"3.Deletion At Specific Location\n"<<"4.Exit\n";

cout<<"\nEnter Your choice:";

cin>>choice;

switch(choice){

case 1:{

deleteAtBeg();

cout<<"\n The doubly linked list after deletion at beginning: "<<" ";

display();

break;

}

case 2:{

deleteAtEnd();

cout<<"\nThe doubly linked list after deletion at End: "<<" ";

display();

break;

}

case 3:{

DeleteAtLoc(1);

cout<<"\nThe doubly linked list after deletion at specific Location: "<<" ";

display();

break;

}

case 4:{

cout<<"exit";

exit(1);

}

default:{

cout<<"\n Incorrect choice.";

break;

}

}

}

return 0;

}

**OUTPUT:**

The doubly linked list is: 3 4 7 9 10 66

Select an Option

1.Deletion At Beginning

2.Deletion At End

3.Deletion At Specific Location

4.Exit

Enter Your choice:1

The doubly linked list after deletion at beginning: 4 7 9 10 66

Select an Option

1.Deletion At Beginning

2.Deletion At End

3.Deletion At Specific Location

4.Exit

Enter Your choice:2

The doubly linked list after deletion at End: 4 7 9 10

Select an Option

1.Deletion At Beginning

2.Deletion At End

3.Deletion At Specific Location

4.Exit

Enter Your choice:3

The doubly linked list after deletion at specific Location: 4 9 10

Select an Option

1.Deletion At Beginning

2.Deletion At End

3.Deletion At Specific Location

4.Exit

Enter Your choice:4

exit

**CIRCULAR LINKED LIST**

In a circular Singly linked list, the last node of the list contains a pointer to the first node of the list.

**Structure of node in circular Linked List:**

struct node

{

**int** data;

    struct node \*next;

};

A node can be added in three ways

* Insertion at beginning of list
* Insertion at specified position of list
* Insertion at end of list

**Insertion at beginning of list**

**Algorithm:**

**Step 1:** Create NewNode and allocate memory for NewNode

**Step 2:** SET NEW\_NODE -> DATA = DATA

**Step 3:** SET NEW\_NODE->NEXT=HEAD

**Step 4:** SET TAIL→ NEXT = NEW\_NODE

**Step 5:** SET HEAD = NEW\_NODE

**Step 6:** EXIT

A picture containing diagram

Description automatically generated

**Insertion at end of list:**

**Algorithm**  
**Step 1:** Create NewNode and allocate memory for NewNode

**Step 2:** SET NEW\_NODE -> DATA = DATA

**Step 3:** SET TAIL -> NEXT = NEW\_NODE

**Step 4:** NEW\_NODE->NEXT=HEAD

**Step 5:** TAIL=NEW\_NODE

**Step 6:** EXIT

A picture containing text, device, screenshot

Description automatically generated

**Insertion at specified position of list:**

**Algorithm:**

**Step 1:** Create NewNode and allocate memory for NewNode

**Step 2:** SET NEW\_NODE -> DATA = DATA

**Step 3:** SET TEMP=HEAD

**Step 4:** SET I=0

**Step 5:** Repeat Step 6 and 7 until I<POS-1

**Step 6:** SET TEMP=TEMP->NEXT

**Step 7:** SET I=I+1

[END OF LOOP]

**Step 8:** SET NEW\_NODE->NEXT=TEMP->NEXT

**Step 9:** SETTEMP->NEXT=NEW\_NODE

**Step 10:** EXIT

Diagram

Description automatically generated

**Program:**

#include<iostream>

#include<stdlib.h>

#include <cstdio>

using namespace std;

struct node{

int data;

node \*next;

}\*head,\*tail,\*temp,\*newNode;

void create(int n){

int data;

for(int i=0;i<n;i++){

cout<<"Enter data of node "<<i+1<<":";

cin>>data;

newNode=(struct node\*)malloc(sizeof(struct node));

newNode->data=data;

newNode->next=NULL;

if(head==NULL){

head=newNode;

tail=newNode;

}

tail->next=newNode;

tail=newNode;

tail->next=head;

}

}

void insertNodeAtBeginning(int data){

newNode=(struct node\*)malloc(sizeof(struct node));

newNode->data=data;

newNode->next=head;

tail->next=newNode;

head=newNode;

}

void insertNodeAtEnd(int data){

newNode=(struct node\*)malloc(sizeof(struct node));

newNode->data=data;

tail->next=newNode;

newNode->next=head;

tail=newNode;

}

void insertNodeAtLoc(int data,int pos){

newNode=(struct node\*)malloc(sizeof(struct node));

temp=head;

for(int i=0;i<pos-1;i++){

temp=temp->next;

}

newNode->data=data;

newNode->next=temp->next;

temp->next=newNode;

}

void display(){

temp=head;

while(temp->next!=head){

cout<<temp->data<<" ";

temp=temp->next;

}

cout<<temp->data<<" ";

}

int main(){

int n,data,choice,pos;

cout<<"Enter the total number of nodes: ";

cin>>n;

create(n);

cout<<"The circular linked list is:"<<" ";

display( );

while(1){

cout<<"\n\nSelect an Option";

cout<<"\n1.Insertion At Beginning\n"<< "2.Insertion At End\n"<<"3.Insertion At Specific Location\n"<<"4.Exit\n";

cout<<"\nEnter Your choice:";

cin>>choice;

switch(choice){

case 1:{

cout<<"\nEnter data to insert at beginning of the list: ";

cin>>data;

insertNodeAtBeginning(data);

cout<<"\n The circular linked list after insertion at beginning: "<<" ";

display();

break;

}

case 2:{

cout<<"\nEnter data to insert at End of the list: ";

cin>>data;

insertNodeAtEnd(data);

cout<<"\nThe circular linked list after insertion at End: "<<" ";

display();

break;

}

case 3:{

cout<<"\nEnter position : ";

cin>>pos;

cout<<"\nEnter data to insert at specific location of the list: ";

cin>>data;

insertNodeAtLoc(data,pos);

cout<<"\nThe circular linked list after insertion at Specific Location: "<<" ";

display();

break;

}

case 4:{

cout<<"exit";

exit(1);

}

default:{

cout<<"\n Incorrect choice.";

break;

}

}

}

return 0;

}

**OUTPUT:**

Enter the total number of nodes: 3

Enter data of node 1:56

Enter data of node 2:78

Enter data of node 3:90

The circular linked list is: 56 78 90

Select an Option

1.Insertion At Beginning

2.Insertion At End

3.Insertion At Specific Location

4.Exit

Enter Your choice:1

Enter data to insert at beginning of the list: 23

The circular linked list after insertion at beginning: 23 56 78 90

Select an Option

1.Insertion At Beginning

2.Insertion At End

3.Insertion At Specific Location

4.Exit

Enter Your choice:2

Enter data to insert at End of the list: 34

The circular linked list after insertion at End: 23 56 78 90 34

Select an Option

1.Insertion At Beginning

2.Insertion At End

3.Insertion At Specific Location

4.Exit

Enter Your choice:3

Enter position : 1

Enter data to insert at specific location of the list: 28

The circular linked list after insertion at Specific Location: 23 28 56 78 90 34

Select an Option

1.Insertion At Beginning

2.Insertion At End

3.Insertion At Specific Location

4.Exit

Enter Your choice:4

exit

**Deletion in Circular Linked List**

A node can be deleted in three ways

* Deletion at beginning of list
* Deletion at specified position of list
* Deletion at end of list

**Deletion at beginning of list**

**Algorithm:**

**Step 1:** IF HEAD = NULL

Write Underflow

Go to step 7

[End of If]

**Step 2:** SET TEMP = HEAD

**Step 3:** SET HEAD=HEAD->NEXT

**Step 4:** SET TAIL->NEXT=HEAD

**Step 5:** SET TEMP->NEXT=NULL

**Step 6:** FREE TEMP

**Step 7:** EXIT

Diagram

Description automatically generated with low confidence

**Deletion at end of list**

**Algorithm**

**Step 1:** IF HEAD = NULL

Write Underflow

Go to step 9

[End of If]

**Step 2:** SET TEMP = HEAD

**Step 3:** Repeat step 4 until TEMP->NEXT! =TAIL

**Step 4:** SET TEMP=TEMP->NEXT

**Step 5:** SET TAIL->NEXT=NULL

**Step 6:** SET TEMP->NEXT=HEAD

**Step 7:** SET TAIL=TEMP

**Step 8:** FREE TEMP

**Step 9:** EXIT

A screenshot of a computer

Description automatically generated with low confidence

**Deletion at specified position of list**

**Algorithm**

**Step 1:** IF HEAD = NULL

Write Underflow

Go to step 9

[End of If]

**Step 2:** SET TEMP = HEAD

**Step 3:** SET I=0

**Step 4:** Repeat step 5 and 6 until I<POS-1

**Step 5:** SET TEMP=TEMP->NEXT

**Step 6:** SET I=I+1

**Step 7:** SET TEMP->NEXT=TEMP->NEXT->NEXT

**Step 8:** FREE TEMP

**Step 9:** EXIT

Diagram

Description automatically generated

**Program:**

#include<iostream>

#include<stdlib.h>

using namespace std;

struct node{

int data;

node \*next;

}\*head,\*tail,\*temp,\*newNode;

void create(int n){

int data;

for(int i=0;i<n;i++){

cout<<"Enter data of node "<<i+1<<":";

cin>>data;

newNode=(struct node\*)malloc(sizeof(struct node));

newNode->data=data;

newNode->next=NULL;

if(head==NULL){

head=newNode;

tail=newNode;

}

tail->next=newNode;

tail=newNode;

tail->next=head;

}

}

void deleteNodeAtBeginning(){

if(head == NULL)

{

cout<<"Underflow";

}

else

{

temp=head;

head=head->next;

tail->next=head;

temp->next=NULL ;

}

}

void deleteNodeAtEnd(){

if(head == NULL)

{

cout<<"Underflow";

}

else

{

temp=head;

while(temp->next!=tail){

temp=temp->next;

}

tail->next=NULL;

temp->next=head;

tail=temp;

}

}

void deleteNodeAtLoc(int pos){

if(head == NULL)

{

cout<<"Underflow";

}

else

{

temp=head;

for(int i=0;i<pos-1;i++){

temp=temp->next;

}

temp->next=temp->next->next;

}

}

void display(){

temp=head;

while(temp->next!=head){

cout<<temp->data<<" ";

temp=temp->next;

}

cout<<temp->data<<" ";

}

int main(){

int n,choice,pos;

cout<<"Enter the total number of nodes: ";

cin>>n;

create(n);

cout<<"The circular linked list is:"<<" ";

display( );

while(1){

cout<<"\n\nSelect an Option";

cout<<"\n1.Deletion At Beginning\n"<< "2.Deletion At End\n"<<"3.Deletion At Specific Location\n"<<"4.Exit\n";

cout<<"\nEnter Your choice:";

cin>>choice;

switch(choice){

case 1:{

deleteNodeAtBeginning();

cout<<"\n The circular linked list after deletion at beginning: "<<" ";

display();

break;

}

case 2:{

deleteNodeAtEnd();

cout<<"\nThe circular linked list after deletion at End: "<<" ";

display();

break;

}

case 3:{

cout<<"\nEnter position : ";

cin>>pos;

deleteNodeAtLoc(pos);

cout<<"\nThe circular linked list after deletion at Specific Location: "<<" ";

display();

break;

}

case 4:{

cout<<"exit";

exit(1);

}

default:{

cout<<"\n Incorrect choice.";

break;

}

}

}

return 0;

}

**OUTPUT:**

Enter the total number of nodes: 5

Enter data of node 1:34

Enter data of node 2:89

Enter data of node 3:74

Enter data of node 4:62

Enter data of node 5:53

The circular linked list is: 34 89 74 62 53

Select an Option

1.Deletion At Beginning

2.Deletion At End

3.Deletion At Specific Location

4.Exit

Enter Your choice:1

The circular linked list after deletion at beginning: 89 74 62 53

Select an Option

1.Deletion At Beginning

2.Deletion At End

3.Deletion At Specific Location

4.Exit

Enter Your choice:2

The circular linked list after deletion at End: 89 74 62

Select an Option

1.Deletion At Beginning

2.Deletion At End

3.Deletion At Specific Location

4.Exit

Enter Your choice:3

Enter position : 1

The circular linked list after deletion at Specific Location: 89 62

Select an Option

1.Deletion At Beginning

2.Deletion At End

3.Deletion At Specific Location

4.Exit

Enter Your choice:4

exit