**CONSTRUCTORS**

* Initialization of objects
  + Initialize some or all member variables
  + Other actions possible as well
* A special kind of member function
  + Automatically called when object declared
  + Explicitly used for initializing objects
  + You don’t call your constructors yourself, compiler does automatically whenever you make a new object
* Very useful tool
  + Key principle of OOP
* You call the constructor only 1 time during the lifetime of an object, not 0, not more than once.

Classes are kind of blueprints (binanın planı gibi), objects are instances (buildings).

**Constructor Definitions**

* Constructors defined like any member function
  + Except:
    1. **Must** have same name as class
    2. Cannot return a value; not even void!

*!!!exit() is in cstdlib*

**class** DayOfYear{

**public**:

DayOfYear(**int** monthValue, **int** dayValue);

CONSTRUCTORS (in public)

If private, could never declare objects

*//Initializes the month and day to arguments*

DayOfYear(int monthValue);

*//Initializes the date to the first of the given month*

DayOfYear();

*//Initializes the date to January 1*

**void** input();

**void** output();

**int** getMonthNumber();

**private**:

**int** month;

**int** day;

**void** testDate();

**};**

**int** main(){

DayOfYear date1(2, 21), date2(5), date3;

*//date1 will be initialized with month:2 day:21*

*//date2 will be initialized with month:5 day:1*

*//date3 will be initialized with month:1 day:1*

cout << “Initialized dates:\n”;

date1.output(); cout << endl;

date2.output(); cout << endl;

date3.output(); cout << endl;

date1 = DayOfYear(10, 31);  *//an explicit call to the*

*//constructor DayOfYear::DayOfYear*

cout << “date1 reset to the following:\n”;

date1.output(); cout << endl;

Initialization Section

**return** 0;

}

DayOfYear::DayOfYear(**int** monthValue, **int** dayValue)

: month(monthValue), day(dayValue)

{

testDate(); *//You can leave body empty if you don’t want to test*

}

***OR YOU CAN DO THIS:***

DayOfYear::DayOfYear(**int** monthValue, **int** dayValue)

{

month = monthValue;

day = dayValue;

}

DayOfYear::DayOfYear(**int** monthValue) : month(monthValue), day(1)

{

testDate();

}

DayOfYear::DayOfYear() : month(1), day(1)

{ */\* Body intentionally empty. \*/* }

*//uses iostream and cstdlib:*

**void** DayOfYear::testDate(){

*/\* My customers are not supposed to call testDate, it’s not a useful function for my customer bc testing the validity of date is not my customer’s work \*/*

**if** ((month < 1) || (month > 12)){

cout << “Illegal month value!\n”;

exit(1);

}

**if** ((day < 1) || (day > 31)){

cout << “Illegal day value!\n”;

exit(1);

}

}

date1 = DayOfYear(10, 31);

HERE, you are making an anonymous object (object with no name) with the right side and then you are assigning that object to date1. Don’t use this!!!

What happens if you define date3 like “date3();”?

date3 is like “int f();” so it is like a function declaration. Function that returns a DayOfYear object but doesn’t take any parameter. So you don’t use parantheses if you have a constructor that doesn’t have any parameters.

You can do “DayOfYear date3[100];” to create an array with 100 DayOfYear objects with day 1 and month 1. No parameter constructor will be called for all the elements.

If you don’t define “DayOfYear();” in public area of class “DayOfYear date3[100];” or “DayOfYear date3;” will give an error.

Without constructors, if you define date1 and then try to write date1.output() , we don’t know what it will print.

In OOP we always try to keep our objects in valid states.

If you carry testDate() from private to public, program can run. We are not using private to compile our program successfully, we use it to write object oriented code to hide the information, encapsulation, data abstraction, and principle of least privilege.

If you carry one of the public function to private area, then most probably code wouldn’t have compiled if user calls it.

Constructor with initialization section is more efficient. It is like “int i = 7;”. Without initialization is like “int k; k = 7;”. With initialization section, you are creating object and assign values without garbage values. Without initialization section, you make an object inside my object, later make the assignment, you are creating object with garbage values and then assign values to them.

**Constructor Equivalency**

* Consider:

DayOfYear date1, date2;

date1.DayOfYear(7, 4); ***//ILLEGAL!***

date2.DayOfYear(5, 5); ***//ILLEGAL!***

**Constructor Additional Purpose**

* Not just initialize data
* Body doesn’t have to be empty
  + In initializer version
* Validate the data!
  + Ensure only appropriate data is assigned to class private member variables
  + Powerful OOP principle

**Overloaded Constructors**

* Can overload constructors just like other functions
* Recall: a signature consist of:
  + Name of function
  + Parameter list
* You can overload your constructors as long as your signature is different
* Provide constructors for all possible argument-lists
  + Particularly “how many”

**Constructor with No Arguments**

* Can be confusing
* Standard functions with no arguments:
  + Called with syntax: callMyFunction();
    - Including empty parantheses
* Object declarations with no “initializers”:

DayOfYear date1; //YES!

DayOfYear date(); //NO!

* + - What is this really?
    - Compiler sees a function declaration/prototype!

**Explict Constructor Calls 🡪 DON’T DO THIS, USE SETTER IF YOU WANT TO REASSIGN**

* Can also call constructor again
  + After object declared
    - Recall: constructor was automatically called then
  + Can call via object’s name; standard member function call
* Convenient method of setting member variables
* Method quite different from standard member function call

**Default Constructor**

* Defined as: constructor with no arguments, automatically defined
* Compiler will make a constructor for you if you don’t declare a constructor
* Doesn’t do anything
* Default constructor calls the default constructors of the data members.
  + For example “int month;” and “int day;” are our data members.
  + Default constructor of int will be called and actually default constructor of int does nothing, just gives garbage value.
  + Instead of int, if we have something other (for example another object) than default constructor of them will be called
* One should always be defined
* Auto-generated?
  + Yes & No
  + If no constructors at all are defined 🡪 YES
  + If any constructors are defined 🡪 NO
* If no no-parameter constructor:
  + Cannot declare: MyClass myObject;
    - With no initializers

**Class Type Member Variables**

* Class member variables can be any type
  + Including objects of other classes!
  + Type of class relationship
    - Powerful OOP principle 🡪 COMPOSITION
    - Using object of another class in your class
* Need special notation for constructors
  + So they can call “back” to member object’s constructor

*!!! TRY TO WRITE YOUR CUSTOMER CODE (main) FIRST WITHOUT SEEING ANY IMPLEMENTATIONS. IF THAT CODE SEEMS LOGICAL, START IMPLEMENTING YOUR STUFFS.*

**class** Holiday;

{

**public**:

Holiday(); *//Initializes to January 1 with markets closed*

Holiday(**int** month, **int** day, **bool** closed);

Holiday(DayOfYear d, **bool** closed);

*/\* This 3rd one is not that good because you exposed DayOfYear to the outside world. If you change your mind and don’t use DayOfYear object in this class, you have to tell to user that “you should change your code, I won’t use DayOfYear class” and you shouldn’t do this!!! \*/*

void output();

**private**:

DayOfYear date;

**bool** marketsClosed; *//true if closed*

};

**int** main()

{

Holiday h(4, 23, **false**);

Holiday h2(1, 1, **true**);

DayOfYear d3(8, 30);

Holiday h3(d3, false);

cout << “Testing the class Holiday.\n”;

h.output();

**return** 0;

}

Holiday::Holiday() : date(1, 1), marketsClosed(**false**)

{ */\* Body intentionally empty. \*/* }

Holiday:: Holiday(DayOfYear d, **bool** closed)

*//THIS 🡪* : date(d.getMonth(), d.getDay()), marketsClosed(closed)

*//OR THIS🡪* : date(d), marketsClosed(closed)

{ */\* Body intentionally empty. \*/* }

Holiday::Holiday(**int** month, **int** day, **bool** closed)

: date(month, day), marketsClosed(closed)

{ */\* Body intentionally empty. \*/* }

***OR YOU CAN DO THIS:***

Holiday::Holiday(int month, int day, bool closed)

: marketsClosed(closed)

{

date.set(month, day);

}

//DON’T PREFER THIS ONE, bc at first you are creating an object, initializing it with January first (1, 1) and setting it to sth else. This is not efficient. \*/

**void** Holiday::output()

{

date.output();

cout << endl;

**if** (marketsClosed)

cout << “All the markets are closed.\n”;

**else**

cout << “Markets are open.\n”;

}

If you have “int Day();” in public area of the holiday class and do this:

**int** Holiday::Day(){

**return** date.day;

}

This will not compile. Because holiday is not dayOfYear. Holiday is a customer class. You should do “return date.getDay();”.

If you have “DayOfYear getDate();” in public area of the holiday class, you can do this:

DayOfYear Holiday::getDate(){

**return** date;

}

This will compile but not good because you exposed DayOfYear class.

**Parameter Passing Methods**

* Efficiency of parameter passing
  + Call-by-value
    - Requires copy be made 🡪 Overhead
  + Call-by-reference
    - Placeholder for actual argument
    - Most efficient method
  + Negligible difference for simple types
  + For class types 🡪 clear advantage
* Call-by-reference desirable
  + Especially for “large” data, like class types

**The const Parameter Modifier**

* Large data types (typically classes)
  + Desirable to use pass-by-reference
  + Even if function will not make modifications
* Protect argument
  + Use constant parameter
    - Also called constant call-by-reference parameter
  + Place keyword *const* before type
  + Makes parameter “read-only”
  + Attemp to modify parameter results in compiler error

**const with member functions**

* The const modifier applies to the calling objects the same way it applies to parameters.
* Saying that my function will not change anything about the object.
* Constant functions not allowed to alter class member data.

void output() const;

double getBalance() const;

int getDollars() const;

int getCents() const;

In DayOfYear, if you define month like “const int month;” it is initialized but later nobody can change it so only constructor can change it.

If you do “const BankAccount cb;” you can do “cb.output()” but you can’t do “cb.input()”. Because cb is const object, it cannot be changed.

If you define “void output();” (not const) and define “const BankAccount b1(100, 20);”, “b1.output()” will not compile bc compiler says b1 is const, on the const object, you can only call const function. While compiler compiles “b1.output();”, it doesn’t know the implementation, it just knows the “void output();” (prototype) so it cannot understand if output function is changing anything in the object.

const constructor is not possible!!!

**Use of const**

* All-or-nothing
* If no need for function modifications
  + Protect parameter with const
  + Protect ALL such parameters
* This includes class member function parameters
* USE CONST WHENEVER IT IS APPLICABLE!

*//Data consists of 2 items, an amount of money for the account balance*

*//and a percent for the interest rate*

**class** BanAccount

{

**public**:

BankAccount(**double** balance, **double** rate);

*//Initializes balance and rate according to arguments.*

BankAccount(**int** dollars, **int** cents, **double** rate);

*//Initializes the account balance to $dollars.cents. For a*

*//negative balance both dollars and cents must be negative.*

*//Initializes the interest rate to rate percent.*

BankAccount(**int** dollars, **double** rate);

*//Initializes the account balance to $dollars.00 and*

*//initializes the interest rate to rate percent.*

BankAccount();

*//Initializes the account balance to $0.00 and the interest rate*

*//to 0.0%*

**void** update();

*//Postcondition: One year of simple interest has been added to //the account.*

**void** input();

**void** output() **const**;

*//output cannot call input because input changes the object*

*//but input can call output*

**double** getBalance() **const**;

**int** getDollars() **const**;

**int** getCents() **const**;

**double** getRate() **const**; *//Returns interest rate as a percent.*

**void** setBalance(**double** balance);

**void** setBalance(**int** dollars, **int** cents);

*//Checks that arguments are both nonnegative or both nonpositive*

**void** setRate

*//If newRate is nonnegative, it becomes the new rate. Otherwise abort program.*

(**double** newRate);

**private**:

*//A negative(-) amount is represented as (-) dolars and (-) cents*

*//For example, negative $4.50 sets accountDollars to -4 and*

*//accountCents to -5*

*/\* const functions cannot change these variables so const functions are for protecting them. \*/*

**int** accountDollar;

**int** accountCents;

**double** rate;

**int** dollarsPart(**double** amount) **const**;

**int** centsPart(**double** amount) **const;**

**int** round(**double** number) **const**;

**double** fraction(**double** percent) **const**;

*//Converts a percent to a fraction. For example, fraction(50.3)*

*//returns 0.503*

};

**Inline Functions**

* For non-member functions:
  + Use keyword *inline* in function declaration and function heading
* For class member functions:
  + Place implementation (code) for function in class definition 🡪 automatically inline
* Use for very short functions only
* Code actually inserted in place of call
  + Eliminates overhead
  + More efficient, but only when short!

**Inline Member Functions**

* Member function definitions
  + Typically defined separately, in different file
  + Can be defined in class definition
    - Makes function “in-line”
* Again: use for very short functions only
* More efficient
  + If too long 🡪 actually less efficient!

**Inline Functions**

Functions that you implement in class declaration.

Actually it is possible to implement all functions inside the class declaration but we don’t prefer bc customer shouldn’t see them.

When you make a function inline, when you call the function, there is no function calling in your assembly conversion. Call (i.e. return rate;) will be directly inserted at the position of function call.

Compiler may choose not to inline your functions and call them.

You can inline your input function writing this “inline void BankAccount::input()” instead of this “void BankAccount::input() {…;}” (in class) in implementation of input function.

class BankAccount

{

public:

BankAccount(double balance, double rate);

BankAccount(int dollars, int cents, double rate);

BankAccount(int dollars, double rate);

BankAccount();

void update();

void input();

void output() const;

double getBalance() const {return (accountDollars + accountCents\*0.01);}

int getDollars() const {return accountDollars; }

int getCents() const {return accountCents; }

double getRate() const {return rate; }

void setBalance(double balance);

void setBalance(int dollars, int cents);

void setRate(double newRate);

private:

int accountDollars; *//of balance*

These 4 are not working on MY data, objects. They need all the data from parameter. So they are kinda bad. You can make them global functions but it is worse. We will see solutions for these.

int accountCents; *//of balance*

double rate; *//as a percent*

int dollarsPart (double amount) const {return

static\_cast<int>(amount); }

int centsPart(double amount) const;

int round (double number) const {return

static\_cast<int>(**floor(number + 0.5)**); }

double fraction(double percent) const {return (percent/100.0);}

};

*//Returns true if the balance in account1 is greater than that*

*//in account2. Otherwise return false.*

**bool** isLarger(**const** BankAccount& account1,**const** BankAccount& account2);

*//You can put this function in public part of class as this:*

**bool** isLarger(**const** BankAccount& other) const;

*and implementation would be:*

**bool** BankAccount::isLarger(**const** BankAccount& other) **const{**

**return** getBalance() > other.getBalance();

**} getBalance is const (and must be bc isLarger is const),**

***but you can call this non-const function in const function:***

**bool** BankAccount::isLarger(**const** BankAccount& other) **const{**

BankAccount b1;

b1.input(); 🡪 OKAY

other.input(); 🡪 NOT OKAY

input(); 🡪 NOT OKAY

**return** getBalance() > other.getBalance();

**}**

**void** welcome(**const** BankAccount& yourAccount);

*//You can put this function in public part of class as this:*

**void** welcome() const;

*and implementation would be:*

**void** BankAccount::welcome() **const**{

cout << “Welcome to our bank.\n”

<< “The status of your account is:\n”;

output();

} **const**

*!!! Global and member functions can coexist.*

**int** main(){

BankAccount account1(6543.21, 4.5), account2;

welcome(account1); *//OR “account1.welcome();”*

cout << “Enter data for account2:\n”;

account2.input();

**if** (isLarger(account1, account2))

*//OR* ***if*** *(account1.isLarger(account2))*

cout << “account1 is larger.\n”;

**else**

cout << “account 2 is at least as large as account1.\n”;

return 0;

}

*//Uses iostream and stdlib:*

**void** BankAccount::output() **const**

{

**int** absDollars = abs(accountDollars);

**int** absCents = abs(accountCents);

cout << “Account balance: $”;

**if** (accountDollars < 0)

cout << “-”;

cout << absDollars;

**if** (absCents >= 10)

cout << “.” << absCents << endl;

**else**

cout << “.” << ‘0’ << absCents << endl;

cout << “Rate: ” << rate << “%\n”;

}

BankAccount::BankAccount(**double** balance, **double** rate)

: accountDollars(dollarsPart(balance)),

accountCents(centsPart(balance))

{

setRate(rate);

}

BankAccount::BankAccount(**int** dollars, **int** cents, **double** rate)

{

setBalance(dollars, cents);

setRate(rate);

}

BankAccount::BankAccount(**int** dollars, **double** rate)

: accountDollars(dollars), accountCents(0)

{

setRate(rate);

}

BankAccount::BankAccount()

: accountDollars(0), accountCents(0), rate(0.0)

{ */\* Body intentionally empty \*/* }

**void** BankAccount::update()

{

**double** balance = accountDollars + accountCents\*0.01;

balance = balance + fraction(rate)\*balance;

accountDollars = dollarsPart(balance);

accountCents = centsPart(balance);

}

*//Uses iostream:*

**void** BankAccount::input()

{

**double** balanceAsDouble;

cout << “Enter account balance $”;

cin >> balanceAsDouble;

accountDollars = dollarsPart(balanceAsDouble);

accountCents = centsPart(balanceAsDouble);

cout << “Enter interest rate (NO percent sign): ”;

cin >> rate;

setRate(rate);

}

**void** BankAccount::setBalance(**double** balance)

{

accountDollars = dollarsPart(balance);

accountCents = centsPart(balance);

}

*//Uses cstdlib:*

**void** BankAccount::setBalance(**int** dollars, **int** cents)

{

**if** ((dollars < 0 && cents > 0) || (dollars > 0 && cents < 0))

{

cout << “Inconsistent account data.\n”;

exit(1);

}

accountDollars = dollars;

accountCents = cents;

}

*//Uses cstdlib:*

**void** BankAccount::setRate(**double** newRate)

{

**if** (newRate >= 0.0)

rate = newRate;

**else**

{

cout << “Cannot have a negative interest rate.\n”;

exit(1);

}

}

*//Uses cmath:*

*//You don’t see any reference to class member data so it shouldn’t be*

*//part of my class*

**int** BankAccount::centsPart(**double** amount) **const**

{

**double** doubleCents = amount\*100;

**int** intCents = (round(fabs(doubleCents)))%100;

*//% can misbehave on negatives*

**if** (amount < 0)

intCents = -intCents;

**return** intCents;

}

You can do self reference in the object functions with the keyword: **this (const pointer to my object)**

**\*this ------->** use when you talk about yourself

“this” is automatically defined by the compiler.

**bool** BankAccount::isLarger(**const** BankAccount& other) **const{**

**return this**->getBalance() > other.getBalance();

**}**

**Member Initializers**

* C++11 supports a feature called member initialization
  + This feature allows you to set default values for member variables
  + We don’t prefer this way, incompatible with previous versions of the C++.

class Coordinate

{

public:

Coordinate();

private:

int x = 1;

Member Initializers

int y = 2;

};

Coordinate::Coordinate()

{ } *//x and y are already initialized*

Initializes c1.x to 1 and c1.y to 2

Coordinate c1;

**Constructor Delegation**

* C++ allows one constructor to invoke another

Coordinate::Coordinate(**int** xval, **int** yval) : x(xval), y(yval)

{}

Coordinate::Coordinate() : Coordinate(99, 99)

{}

* The no parameter constructor invokes the two parameter constructor to initialize x and y to 99 and 99

**Static Members**

* Static member variables
  + All objects of class “share” one copy
  + One object changes it 🡪 all see change
* Useful for “tracking”
  + How often a member function is called
  + How many objects exist at given time
* Place keyword *static* before type

In C we can do things like this:

int k = 0;

int f(){

k++;

return k;

}

int f(){

static int k = 0;

k++;

return k;

}

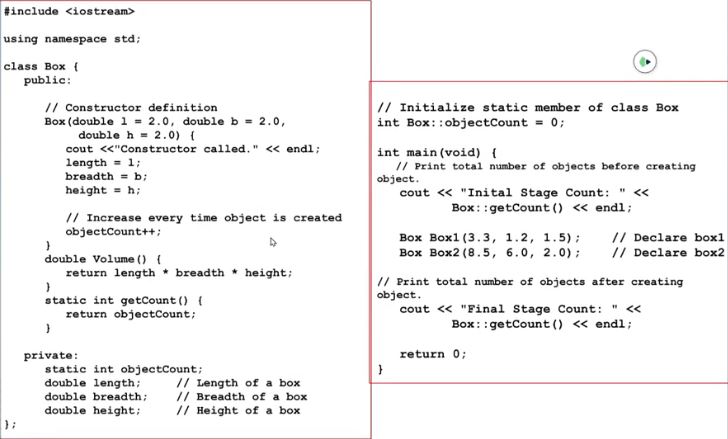
These two has the same effect. Difference is in the left, k is only available in the scope of f but in the right everyone can access k.

We don’t like this static because if 2 different processes call f (in the left), one of them will get affected from the other’s calling.

In C++, static doesn’t have bad effect.

When you say I have a static data member, it behaves like a global variable.

**Static Functions**

* Member functions can be static
  + If no access to object data needed
  + And still “must” be member of the class
  + Make it a static function
* Can then be called outside class
  + From non-class objects:
    - E.g., Server::getTurn();
  + As well ass via class objects
    - Standard method: myObject.getTurn();
* Can only use static data, functions!
* Static functions can’t get const keyword because statics cannot access the variables anyway so it cant change them

What is “Box::getCount()” in main?

We can do it because getCount is static function. This function can work only on the static data members. We can’t do “Box::Volume()” because Volume function needs length breadth and height data members of the class. Volume needs only on the objects.

getCount can’t call Volume function, but Volume can call getCount.

So static functions can only work on the static data members.

Regular functions and constructors can work both the static and non-static data members. So if you erase static keyword in front of “int getCount()” nothing will change, but “Box::getCount()” lines will give ERROR message. So if you are dealing with static data members, make your functions static so that it can be called using the class name.

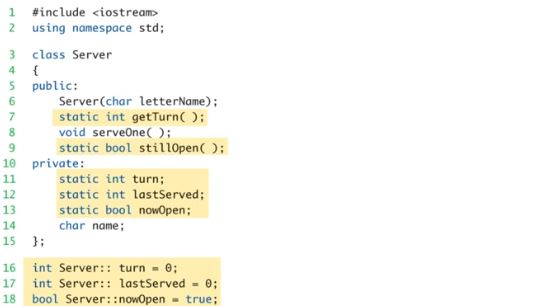
For the lifetime of your program (not lifetime of your object(s)), you initialized the static data members only once.

Static data members are created before the main function starts, but it can be accessed only within the class because it is private data member of the class.

Unlike the globals, static data members can only be accessed within the class. Since they are private, I can only access objectCount only from constructor or getCount function or Volume function.

You can initialize static data members in class like member initializers for C++11 and afterwards.

NEVER WRITE SETTER FOR OBJECTCOUNT!!!!!

![Graphical user interface, text, application

Description automatically generated](data:image/jpeg;base64,/9j/4AAQSkZJRgABAQEAYABgAAD/4RDaRXhpZgAATU0AKgAAAAgABAE7AAIAAAAFAAAISodpAAQAAAABAAAIUJydAAEAAAAKAAAQyOocAAcAAAgMAAAAPgAAAAAc6gAAAAgAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAE1lcnQAAAAFkAMAAgAAABQAABCekAQAAgAAABQAABCykpEAAgAAAAM2MgAAkpIAAgAAAAM2MgAA6hwABwAACAwAAAiSAAAAABzqAAAACAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAMjAyMToxMDozMSAxNDowNjo1MAAyMDIxOjEwOjMxIDE0OjA2OjUwAAAATQBlAHIAdAAAAP/hCxdodHRwOi8vbnMuYWRvYmUuY29tL3hhcC8xLjAvADw/eHBhY2tldCBiZWdpbj0n77u/JyBpZD0nVzVNME1wQ2VoaUh6cmVTek5UY3prYzlkJz8+DQo8eDp4bXBtZXRhIHhtbG5zOng9ImFkb2JlOm5zOm1ldGEvIj48cmRmOlJERiB4bWxuczpyZGY9Imh0dHA6Ly93d3cudzMub3JnLzE5OTkvMDIvMjItcmRmLXN5bnRheC1ucyMiPjxyZGY6RGVzY3JpcHRpb24gcmRmOmFib3V0PSJ1dWlkOmZhZjViZGQ1LWJhM2QtMTFkYS1hZDMxLWQzM2Q3NTE4MmYxYiIgeG1sbnM6ZGM9Imh0dHA6Ly9wdXJsLm9yZy9kYy9lbGVtZW50cy8xLjEvIi8+PHJkZjpEZXNjcmlwdGlvbiByZGY6YWJvdXQ9InV1aWQ6ZmFmNWJkZDUtYmEzZC0xMWRhLWFkMzEtZDMzZDc1MTgyZjFiIiB4bWxuczp4bXA9Imh0dHA6Ly9ucy5hZG9iZS5jb20veGFwLzEuMC8iPjx4bXA6Q3JlYXRlRGF0ZT4yMDIxLTEwLTMxVDE0OjA2OjUwLjYyMDwveG1wOkNyZWF0ZURhdGU+PC9yZGY6RGVzY3JpcHRpb24+PHJkZjpEZXNjcmlwdGlvbiByZGY6YWJvdXQ9InV1aWQ6ZmFmNWJkZDUtYmEzZC0xMWRhLWFkMzEtZDMzZDc1MTgyZjFiIiB4bWxuczpkYz0iaHR0cDovL3B1cmwub3JnL2RjL2VsZW1lbnRzLzEuMS8iPjxkYzpjcmVhdG9yPjxyZGY6U2VxIHhtbG5zOnJkZj0iaHR0cDovL3d3dy53My5vcmcvMTk5OS8wMi8yMi1yZGYtc3ludGF4LW5zIyI+PHJkZjpsaT5NZXJ0PC9yZGY6bGk+PC9yZGY6U2VxPg0KCQkJPC9kYzpjcmVhdG9yPjwvcmRmOkRlc2NyaXB0aW9uPjwvcmRmOlJERj48L3g6eG1wbWV0YT4NCiAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAKICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgIAogICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgCiAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAKICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgIAogICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgCiAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAKICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgIAogICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgCiAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAKICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgIAogICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgCiAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAKICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgIAogICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgCiAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAKICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgIAogICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgCiAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAKICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgIAogICAgICAgICAgICAgICAgICAgICAgICAgICAgPD94cGFja2V0IGVuZD0ndyc/Pv/bAEMABwUFBgUEBwYFBggHBwgKEQsKCQkKFQ8QDBEYFRoZGBUYFxseJyEbHSUdFxgiLiIlKCkrLCsaIC8zLyoyJyorKv/bAEMBBwgICgkKFAsLFCocGBwqKioqKioqKioqKioqKioqKioqKioqKioqKioqKioqKioqKioqKioqKioqKioqKioqKv/AABEIAZcCPwMBIgACEQEDEQH/xAAfAAABBQEBAQEBAQAAAAAAAAAAAQIDBAUGBwgJCgv/xAC1EAACAQMDAgQDBQUEBAAAAX0BAgMABBEFEiExQQYTUWEHInEUMoGRoQgjQrHBFVLR8CQzYnKCCQoWFxgZGiUmJygpKjQ1Njc4OTpDREVGR0hJSlNUVVZXWFlaY2RlZmdoaWpzdHV2d3h5eoOEhYaHiImKkpOUlZaXmJmaoqOkpaanqKmqsrO0tba3uLm6wsPExcbHyMnK0tPU1dbX2Nna4eLj5OXm5+jp6vHy8/T19vf4+fr/xAAfAQADAQEBAQEBAQEBAAAAAAAAAQIDBAUGBwgJCgv/xAC1EQACAQIEBAMEBwUEBAABAncAAQIDEQQFITEGEkFRB2FxEyIygQgUQpGhscEJIzNS8BVictEKFiQ04SXxFxgZGiYnKCkqNTY3ODk6Q0RFRkdISUpTVFVWV1hZWmNkZWZnaGlqc3R1dnd4eXqCg4SFhoeIiYqSk5SVlpeYmZqio6Slpqeoqaqys7S1tre4ubrCw8TFxsfIycrS09TV1tfY2dri4+Tl5ufo6ery8/T19vf4+fr/2gAMAwEAAhEDEQA/APfdW1yLSSnmwTSBmVSyJ8q5OBk0/UNWFlLDBFBJcXEwLJEhA4HUkmsvxpewJpQtix84yxOFCk8BwSf0p2pT2U62mpQ35tpEDIknllgw7jH4VDk7nJKpJSlFPaxr6ZqMWqWYuIQy8lWRuqsDgg1crA8IWk1tpEjTlyZ55JV3jDYJ4zW/VJ3RtRlKUE5bhWfqOptZz20MESTyTSqjoZVUoh6tg9celaBrj/HFtZw3GkarLbp5sN/CHuAmWWME55HOK0pxUpWZU5NRujq5bmCD/XzRx8bvnYDj1qpqet2Gk6Y1/dzqIBjDKQd2SBx69a5fWbK117x/oLTQ/abF7CdiCp2NymMj+hrDGjJdfCW9iuLHzns7yf7Orx5aNBN/D3xt/Sto0o6XZnKpJN2R6hBcQ3Me+CVJF9VYHFOklSJN0rqi+rHArG8MHRPsLjw/HCkeR5oiQqN2O9UfiJa/bfCf2co7rJd26sqZzt81c9PasJKzsaxd1c6OK8tp2Cw3EUjMu4BHBJGcZpFv7R7k26XULTjrEJAWH4da8Sh0a40fxEZtBspIbhZ9Wih2IRhAuY1Htnke9aXh9NGl8UeDX0KxcTokpv7owMrGQxHKuxGS2c1JR6da6u934guLGGBHtYYVf7Uk6tlySCm0cjHqautfWiSCN7qFXLbApkAJb0x61wVhFpvh/wCNd+kdullHfaZCE8uIhZZTI5PIGM1gap4dtrjR/GepzWDS38erE20rKSyAMnKen1FAHpmu+II9G+zwx28l5fXblLa1iIDSEckkngKB1Nadu0r20bXEYilKgugbcFPpnvXI6g4sviToV3etst57CW2id+iykq2M9iQD+VdlQBka/rcuhQpdvYSXFkuTcSxMN0I/vbe49cVqQzR3FvHNA4eORQ6MOjAjINUPEN5bWPh2+nvWUQrA+Qf4uMY/GqXgWzubDwFo1rfAieO0QOG6rxwPwGBQBv0UUUAFFFFABRRRQAUUUUAFFFFABRRRQAUUUUAFFFFABRRRQAUUUUAFFFFABRRRQAUUUUAFFFFABRRRQAUUUUAFFFFABRRRQAUUUUAFFFFABRRRQA13WNC7sFVRkknoKzv7XebmwsZ7lO0gwin6FiM02+QX+qRWMn+oRPOmT/npzhVPtnkj2qO98Qw2WqxWIjMm7AYofuk9Bik5KO5E5xgryZZi1dPNSK8gltJHOF80fKx9Aw4zWgDms20v9I8TabIbG6gvrZiUcxOGwR/I0aNNK0MtvcsXktpWiLnq4HQ/iCKZZpUUUUAFFFFABRRRQAUUUUANZVP3gD9RQUQrgqMemKx/EUbJZS3Q1C4tTEh8tYiAGftxjn6VlX2q3IXT7fUbmSyBtfPuniHzZ4AA9OTUuSTOedaMG00dcOOlLWP4bvpbyxlWeTzXt5mhMhGN4HQ/kRWxTTujWE1OKkgpGUMMMAR7ilrk/G95um0fSYb14Zb6+jSWKCTbK0PO4jHIHHJplnV7RxgDjpxQFAGMDFR21utraxwIzssahQXYsxx6k9aloARUVPuqF+gpSAeozRRQAm1Qc4H5Uixov3VA5zwKdRQAhRS2SoJHfFG0c8DnrxS0UAVdQ02z1aza11G3juIG6o4zz6j0PvU1vbx2ttHBCCI41CqCScAe5qSigCjfaNYandW89/bLO9sS0Qckqp9dvQnjvV6iigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigDKuHFnr8M8pxFcx+RuPQPnKj8eR9frWZf+HriXxAl3bBTEzB3LHp6iujubaK7gaGdQ6MMEGswWmrWh2WV7FLCOiXURZl/4ECP1BqZRUlqZVaUaqtIyPBfw90P4e211Jp7StLPlp7md/wCEHIGOgArZ0LfNHcXjqyi6mMqKRgheAv6AH8aYNLvL1gdXuhKinIghTZGfrySfzx7VrogRQFGAKt6muw6iiikAUUUUAFFFFABRRRQBgazpWpXmr291aTWxhgT5Ip1YgP8A3uO+Kkv9KvZ57S+t3t/tkMZjkVwdjg9fccirt/rFjpsipeziNmGRkE4HqcdKfc6pZWdstxdXMcUTDKszAZHt61Nlc5nCm3K7IdF01tNsjHK4klkkaSRgMAsTk4rRpsbrJGrocqwBB9RTqaVjeMVGNkFcr41snL6TqltYyXEtjfRyStAm6URc7gO5HPIrqqKZRFbTi6tY51SSMSKGCyLtYexHY1LRRQAUUVHcXENpbSXF1KsUMSlndzgKB1JNAElFZOj+J9I16WSPSrwTvGu5h5bL8vqMgZH0rWoAKKKKACiiigAooooAKKiuLmC0t2nupkhiQZZ5GCqPqTUGmatYazam50u6juoA5QyRHK7h1GaALlFZw13T31w6RHMZL1U3uiIWEY/2mAwpPYE0lp4g0691690a3mLX1iqNPGUI2hhkc9D+FAGlRVM6tYDWBpRu4hftF5wt93zlM43Y9M1DfeINO07WtP0q7mKXeo7/ALMmwkPtGTz0HFAGlRWe+uafHra6TLP5d46b40dCBIP9luhPsDmtCgAooooAKKhu7uCws5bq7kEUEKlnduigd6paP4j0rXhL/ZV4s7QkeYm0qy56ZUgHHvQBp0Vnya7YR61HpPnF72Rd3lxoW2D1YgYX8abH4g0+XxHNoSTMdQghWd49hwEPQ56GgDSorDfxn4ej1M6fJq1ut0svkmNiRh/7uemas6n4h07SNQ0+yv5mjn1KUw2yhCd7AZxkdPxoA06Kz59d0+11iHTLmfyrqdd0QdCFf2DYwT7ZzWhQAUUUUAFFFFABRRRQAUUUUAFFFFABRRRQAUUUUAFFFFABRRRQAUUUUAFFFFABRRRQAUUUUAFFFFABRRRQAUUUUAFFFFABRRRQBia/eqcaXbqr3d1GRlsYjToWNV9ZsLa08EXEJCyG3smRHYAnha077Q9N1G4E15apLKF2hmHOPSludF0+8t44Lm2SSKJdqKc4A9Khp6nNKnKTk3bXREmlkHSrXBz+6X+Qq3Vay06106Ix2UQiQnJAqzVLY3gmopMKKKKZQUUUUAFZ3iDSrfXPD95pl7K0MN1EY2kU4K57itGoL2yt9RspbS9hWaCVdrxuOGFAHC2uva54bvJtB1pbfULqPTpbqzvbdNrSLHj5ZE7HkdODVHwVJ4mvL/RdVZ7qWzvYma+e4vEkjclcqY0H3cHjHp1rudJ8MaPokkkmm2UcLyLsZ+WYr6ZOePamWXhHQtO1H7dY6bDBcbiwZMgAnqQOg60Actoum6p4j1PWLifxDfQR2OrtHbwQsAgVdhIb+8D0x2qrd+INT02+1bwo13I+qXl6v9mSscsIJckkeyAMPyr0S1sbax877JCkXnymWTaMbnOMsffgVHLpNhNq0Opy2sTXsCGOOcr8yqeoBoA841268Q6l4v1bSdLkvt2mW0ItRBdJF87JnzHDcuM8enWtTW5davNY8JabLqMunyXsM3277KwyxVFJAPbnPPbNdTqvhbRdbuUuNT0+KedF2rKchsemR1HtVoaTYrLaSC1jD2SlLdsf6sEYIH4CgCPRNNk0nTFtJr2e9KuxE05y+CcgE98dKzvGsGr3Ph/y/D8/lXJmQuqyCN5Y8/Misc7WI710NUtU0ew1q0FtqlstxEGDBWzwR3BFAHOeDbqw1PT9R0+4S+eWzuQl1bapIJjE5AIAboR3pnw22pomqLEFGNVutqjH981uReFtFg0iTS4tPhSzlbdJEAfnPqT1J4puk+EdC0O6NzpOmw2spBBaPPOevegDH+F483waLybm9urqd7tj94yeYwIP0wBXSQaVaWuqXeoxx/6VdBVlkJzlVHAHoKqWfh6PTtbnvrC5khhumLz2gAMbyH+Mf3T6461rsodSrDIIwRQB4te6nKnidfGn9nXgSPVPKN7tHlfY8eXjOc43ZbpXrtxpVnealZ6hPH5k9mG8hieF3DBOPXFDaNp76OdLa0jNiU8swFfl2+mKtxosUaxoMKoCgegFAHI/FBUXwPPdKdl3azQy2jj7wl8xQoH1zj8a6a41C20+wW51O4itYwFDvKwVQT2yfeqN/wCHotU1iC81C4kmgtmEkNmQBGsg6Oe7H0z0p3ibw9a+KNDk0u/4gkdWfCgngg8enTrQBrAggEcg9Kw/GMGrXPhmeLw/crb3rMm1i4Qsu4blVj0YjIBrbVQihVGABgVV1PS7LWLI2mpW63EBIbY3qOhoA5Hwtrmn2VjrUWqSahbSaaFlvYtTnE3kqVyCrDgg4qz4Ts21LX7zxZOot2vrdILe1BG5IVJIZ/8AaOfwFbEPhLQoNJudMi0yAWl1/r49ufN/3j1NO0fwtougTPLo+nxWjyLtcx55HpQBifDjM+m6reXPN9NqtyLhj94bXKqv0CgYrpU0u1j1iXUxHm7liWIyE9EHIA9OTVOLw+lr4gk1SxupbcXB3XVsoBjmbGA2Ox6cjritigDxXUxcxab4jubk2j6RBrxluYcYuHC7DhGPHXHbPWvW7nTLPU7iwvLmIu9o/nQBjjYxXGceuDVJ/Bnh2XUmv5dJtnumk81pGXOX9cdM1uUAct8SY4W+HuqzTYWS3i86CT+JJVIKEH1zirWsiS68En7Rq7aJLJAhe9BAMLcZ68deKm1bw7HrWoW8moXMslnAVcWWAI3cHIZu5xxx0qbxFoVv4k8P3Wk3hxBcrtc7QcDPb3oAu2Y22MA87z8RqPN/56cfe/HrU1R28CWtrFbwjEcSBFHoAMCpKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAy9Yn1K1ge4sjbCKKMu4lzlsdvami71C/wBJtbvTxDD5sYkcTgnGRnHFUfEP9oXGowW6WM1xpyrvlETKPMbspyenejXpNQksrO3s7Gb7PL/x8rCQHVR/AOe/T6Vm92ccp2lJ66GnoWpNq2kx3UiCNySrAHIyDjI9uK0aqaZj+z4gtq1oAMCFsZX8qt1a2OmnfkVwoooplhRRRQAVFc3MVnbPcXDFY4xuYgE4H0FS0Hkc0AYmg6wL7S7q9mvoLmGOeTEkSMgRB/CwPOQOtPsvFWi6jfJa2d+ks0ib0ABwwxng4waxNEtNRgsfEGnT6dNGZ7i5mhmYrskDn5QOc02y0C8gfwgRaiMWETi5xj5CY8fjzXS4Qu9f6sc/NOysXtN8daZf65e6dv8ALNvKsUTbWPmkjPpxzxzXT1yNlBe6R4y1aVtMmuLbUJInimi2lUwu0g5ORXXVnUUU1ymlNya94w9U8Z6Bo11LbalqMcM8IUyR7WYqGGQTgdMd6vRazp8+oJZQ3Ub3EluLlEU/ejJwG+lctdeHryXxD4wu/sYZb/TkgtXOP3jCJgQPTkisyz0vXtC1rRb6LR5L0DQ0sJEjkUeVKGz8xJ6e4rI0OquPHfhq1hgluNWgRLhC8THPzqG2k9OgNZGtfEGyj/sqbRr6CW2k1VbO8kZThF2Mx6/Qc1x6eB9ek0O0hn0z99Hod1bsu5TtledWC9e6g1d1jwrqqaobmPRnu7aLWra68iPb88SQbSQCccNQB6DYeLdD1Kwury01GJoLM4uHfKeV3+YHBFSaT4n0jW1nOm3iym3AMqlGRlB6HDAHHvXBat4O1rxDpniXUFtP7PudTa3+z2RkAZkhOfmI4DNz9OK1fDOhsmq3uoS2Otx3ktkbdpNSnR1IzkKu3370Ab2n+N/Duqahb2Wn6nHPPcqWiVEbDYGTzjGcDpnNb9ct8PtCk0bwLpNpqNmkF7boS6kAlGJOTkd8Grun6/PeeMNV0aSyMUVjHFIlxvyJd46Y7EYNAG5WNqni3RNHvhZ396EuCAxiSNnZR6naDgfWtmuKjj1Lw/4v1q6OiT6lb6m8ckU9sVLJtUKUYMRgcZH1oA3NU8W6Jo0kceoXyxySJ5ioqM5Cf3iADge5q3FrWnTXVrbw3cckt3EZoFU58xBjLA+nIrk7u21TSfGOp6vFok+pQ6pZRRosTJugdAQUYE9DnORWZY+H9c8NXHhi8XTZNRazs57eeKB1BiaRgw6kfKOlAHZz+MNBtrSO5n1KKOGWZ4EY5+aRM7lHHUYNVLrx3okfhW61y1vFuLeDeuArAmRQTsIxkHjuK5TSvCmseX4bOoacqta6zdXdym4MI0cPtPvyRWzZ+Hb8WfjW3ktxH/adzK9pkjDhogoPtzQBu+GvFOn+JtPinsZMymJJJY9rDy9w6ZIGfwq7pmsWGsxzPplytwsEzQSFc/K69V/CsnwZPdjRbWwv9IurCW0t0jZptu1yBg7SCc9Kk8JaoNUttQYaYmn+RfSwlUIIlKn7+R60Ab9FFFABRRRQAUUUUAFFFFABRRRQAUUUUAFFFFABRRRQAUUUUAFFFFABRRRQAUUUUAFFFFABRRRQAUUUUAFFFFABRRRQAUUUUAFFFFABRRRQAUUUUANZ0TG9lGemT1pHdEGXYKPUmsPxRbafJbZuIBNeyKY7ZQfm3e3pjrmi709v+Ebto72zOqTwxgFNw5bHJ5qb62MJVGpNW2N4EEZHSlrF8JMG8OW4WRn2llIbqhyfl/DpW1TWqNKcueCl3CiiimWFFFFABRRWdr+rLoeh3F+yeYYwNqA43MTgD8zTSbdkJuyuy8ZYxMIt6+YRuC55I9cU+vPZL3UdN8arqGuNbyNDo00vl2ykbQGU4yev1rSsvEGsw3ejyastq1trB2okIIaBipZQSfvcDFbOi+hjGsmdhRSd6WsDcKKKKACiiigAooooAKYsUaO7oiqznLkDBb60+igAooooAKKKKACiiigApscaRLtiRUXOcKMc06igAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigDG1Lw99v1RL6O/uLaVI/LHl4wBnPcGrNzp9zNaQxQ6jPC8Yw0iqpL/XIxRqWqNp+SLKedFXe7xgYUU99WtY9H/tJ5MW/l+YG9R2/GpsjC1NSf4i6Xp0WlWK2sBZgCWLMcliTkk1cqrpt9HqenQ3kAYRzLuUMMGrVNbaGsOXlXLsFFFFMoKKKKACqWsaXBrWkz2F3u8qZcEqcFT1BHuDV2imm07oTV1Y5qz8F28N813f3tzqEz2rWsjTkfMjEcYHTpS6d4Nhsr20mn1C6vI7AEWkMxG2HIx2GSQOOas6J4hfXL69SHT5IrS1meAXTuP3jo21gF6jvya26v2k+5Hs49gooorM0CiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooA5rxLrEcd3HpDzfZknQtPOQflTptX3NM1TT7qe1sP7Jigm0+3j3LHJIV3ED5T0OcfzrpWjjc5dFY+4pwAAwAAPSptc55UXNvme5h+DftA8LWguo1jYLhQrZ4rdpAABgDA9qWmlZWNaceSCj2CiiimWFFFFABUdwksltIttIIpWUhJCu4KfXHepKKAOS+HQddEv1nYSTrqlys0wGBM4kOXA7A+ldbWVp3hzT9K1W7v7FZYpLslpY/OYxliclgmcAk9xWrQAUUUUAFFFFABRRRQAUUUUAFFFFABRRRQAUUUUAFFFFABRRRQAUUUUAFFFFABRRRQAUUUUAFFFFABRRRQAUUUUAFFFFABRRRQAUUUUAFFFFABRRRQAUUUUAFFFFABRRRQAUUUUAFFFFABRRRQAUUUUAFFFFABRRRQBjeIklispbxNSmtBFGdqRhcM3bOQc56YrP1W91EaLpckkk9o0hBungj3FBtJ6YPerOtaZqt5q9vc2bWr28C/LDcbsb/73HWtC6/tT7BGLWO1e5IxIHZgo47cVDu2zklGUpS3Q/SXWTTIXS7N4rDImbGW/KrtZ2h6a2laVHbSOJHBLMQMDJOTj25rRqlsdFO/IrhRRRTLCiiigAooooAKKKKACiiigDndR8Vm312TSNL0y51O7gjWW4EJVViVumSx6n0pl/wCMDbajFptnpN1fai1uLmW2iKgwITgbiTjJOQB7VXvtD13T/FV5rPhp7GVdQjjS5tr1mUBkGAysoPbqKivdA8QW/iJfEGjy6e19cWS2t5BcFxFlSSrIQM8ZPB60AXbDxxp2o3GjRW0cxOrCbZuGDE0X3lYeueKzPEHjyeLwfrOpaHYTSTadPLbMzBSEZDguRnlaqjwHq+mW+hXOj3lrNqWmyzyTm5VljlM33yMcjB6Cren+B75PBOvaNqN7FJc6rcTzeeiHapk5HH1oA6Dw7q15q+nrNfabPYtsUjzSp8zIzkYJ4rXJx1rF8Nx67BZC31+KxTyUVImtJHbcAMZO4DFal5aQ39nLa3SeZDMpR0JxkGgDLh8U6deawNN05nvZFYiaS3XdHBgfxP0B9utFx4q02PV00u2d729ZwrxWq7/JB/icjhR9apaT4Um8OzRWujXg/sU5WSwnTJjBB+44569jn602w8HN4evBJ4XvPslrJNvuLKZd8bgn5ip6qfzFAHU1yWsaFr13daw9jqLwpc/ZvsoEhHl7GzJ9MiutooA4K88ax6P4113TtVvzHFDZwvax+UWw5U5PA9cdaXQvHE7+HdCha2n1bWL+0Ny0cW1cIDguScADPFbcHh64j8U67qbSx+VqNtFDGnOVKqQSfzrAs/Bet6D/AGPfaLcWM1/ZWBsbiK53LHKu7cCGAJBB9qANCL4laQ+3zoriAfZp55BInMZhYK6Ef3gSK2dI8SWet3RisQ7qLaK58wjAKyAlR9cCuRf4a3UyaTJPewm6iv5bvUGVSFlWUgsij0yq9fSt7wV4Vn8MRagtzcJcGe5JhKjGyEfcQ/QE0AafiLXoPDmlfb7qKWWPzUi2RDLEuwUYH1Nc/p3xLsL2/W3urG6sFKz7pZwNqtD99eD2BzmtvxTo02uaVDa28iRtHdwTkv0ISQMR+Qrk7v4b3d6yJLdwpEXvy5UHIFxjbj6d6ANey8fR3eqabbSaRe20OqBzZ3EwUCQKu7O3ORkdM1DY/Em1vZbBl0u9js726Nol26gIJckBeuSMjr0qrY+FPE0+saBea5d2BTR0kiEdsrfvAY9gc579OOgqa38D3sPhnQtNNzCZNN1MXkj84ZRIz4HvhqANQ+NbBdM1a6kimR9LuDbzQEDez8bcD/ayMV0UbF4lZlKFlBKnqPavOZtFg174uC609phYwRLJqalCscs8bERDJ6kZJP0FekUAcl451Q2S6fDMmowW8t7Apu7KRFwxfARs87T3xV/xB4pj0K/sLEWVzeXWoeZ5EcAHJQAkHPTr1qPxtoN74h0OG30ySCO5gvIblDPnYfLcNg455xSS6LqV9r+g6revbRyWCTi4jiLEMXAA2k9uO9AFzw34gi8R6Y91FBLbPFM8E0E2N0bqcEHHFP8AEWuxeHNHfUrmCSW3idRKYxkxqWALn2Gcmq3hbQp9Ch1FLiRJDdX810uzPCucgH3q14jm0+Hw/djWM/Y5YzFIAhbIYYxgUAUtU8Z6XpNzcpcMzR2ll9tuJUGVjQnCj6tg4FVrHxxHNfQ2mo6XeabJc273Nr5+3EyqMsODw2CDg1geG/AbX3wyvbDVZ5heawnzTSr88aLxCCP9lQOPc1o2vhfXtU1mxu/E9xZKmmW8kNutmGJlZ1CmRt3TgdBnrQBpweNrKey8PXKwTBNek8uAEDKHaW+b8FqrP49A1XUrGw0TUL19MnWG5eJRtXIU5GTz97p14rF0rwN4kt5PDVvf3mnmx8P3JaMQh98ybWUE56HkcV1Hh3QbjSNa8Q3k8kbpqd8LiIJnKqI1XB98rQB0AOQDXN6z4wTTNQnsrPTrnUZrWHz7nyNoWFDnGSSOTg8V0leYeMftGja9rL2Wq28KavaqJoJLd3m3BSo8rHBJ6c9KAOmu/Hdrbadot1HY3Vw+tLm2hiUFs7d2DRN40a3s7Tz9HvE1G9neGDTzt3sU+82c42471T0nwveGx8FzSMsTaPBmeN87iWi24H4mtHxRoF9f6lpesaLJAuoaYz7I7jPlyo4AZSRyOg5oAhHj2yXTWuLi0uIJ4b5LGe2cDfFI54zzgjkHIqfVvGljpEusR3EMzHSbVLqbYB8ytnAHvxWFdeBNWvtDv5bi7tV1q8v4r/CqxgUx42p64wOTWb4l8N67F4f8W6vqs1tcXOoadHEsFqjYQoTwM8kc9aAOltvGJ1e9/suKyutMnvrR5tPubhAVlwOuM8YyDg9qzNPGtWXjix0+z1291eOKMnVxcqhjhJX5dpAGGJ/h54rQ0DQtXutWsNZ8RSWqGytfJtLa1DELuA3MzHvgAYrZ0PTJ9PutUkuEtl+13ZmQwAgsuAAXz/Fx2oA164+/8P8AiC41TUprbU3jgnvbSW3QSEeXEmPNX2zzXYUUAZmq6w2mXmnwLYXV2Lybyi8CZWDjO5/QVp1geI59bh1LRU0TyzDLd7b1WQk+VtPIPbFb9ABRRRQAUUUUAFFFFABRRRQAUUUUAFFFFABRRRQAUUUUAFFFFAFO91ax051S8uUiZhkBj2p9xqFra2wuLieOOE4w5bg1m+ILlGUadBEkt5doVG4cRp0LH2GfzrJ1KzmttS0DTraRCkcThZJRuG4KBnHc4zUOVmctStKLdtTq7W6hvIBNbSCSNujCpqytAvZb2zlE6oJIJmiZoxhWIPUVq1Sd0bwlzRTCiiimWFFFFABRRRQAUUUUAFFFFABVPU9Ws9Ht0n1CXyo5JViU7ScsxwBx71ydw+oeIfH+p6T/AGzdaXa6bBE0cdoVV5mcZLkkHIHTFO8awX+m2ukXltrN4PKvLa3ki+XZOGkAJYY64PbFAHbUVxPjG41SXxj4c0jTNTl0+G/W5E7xKCxCqpGMjr6GrXw9v7670fULfU7uS9ksNSntEuJQA7ojYUtgAZxQB1lFZmv/AGldLaWz1OHTZITv86dA0eB1DZxwfUEGsDRPEt/4o0u9jVTprwD93qsS7raUjum/GR6/zoA7KiuQ8KeLrzWdTm0+e2S7jgBH9q2IP2aQ+nP8X0yPeuvoAKK8kOp+IBpr6+2vXOLfXTaJZhV8tojNsIbjJODxzxWtqniS/tNI8czG/MT2E4S1YkDysopAH4mgD0Wqj6pZx6vFpjzAXk0TTJFjlkBAJ/MivLNQ1nxJP4pZLTXJreA31paCIRqQBLBlm6dc8j3qHSG1XxD4k8PLLq80E50y7jmvEC+aVScLxkYBOBzigD2SivI38Va//wAI/pOnx3NxdPd6rPZfb4CiyzRR5IKlvlDNjr7GtG41HxVpHwx8QT6i9xb3NrJ/oFxcMjTGMsuN23gkZI+lAHpdFc94c0fUrCZrvUNcudRW4gUtFMFCxv1JXA4GCBj2qz4bguLe0uhdayNXZrqRlkG390pPEfH93pQBsUUUUAAAHQUUUUAFFFFABQQD1GaKKACiiigAooooAKQqCckAn6UtFABRRRQAUdetFFABRRRQAUUUUAFFFFABRRRQAUUUUAFFFFABRRRQAUUUUAFFFFABRRRQAUUUUAFFFFAGbfaBp2pXQuLy33yhdofcQcenFSXGkWd1aRW00WY4ceXgkFceh61FrFxqNpC89kLfyooy7eaxBOO3HSom1C/udJtbzT4YFEsYkcXDldoIz2FTpc55Omm00aNpaQWNuILZAka9AKnrN0G+udR0tbm8jSNmZtuwkgrng8+taVNGsGnFOOwUUUUywooooAKKKKACiiigAooooAyNW8LaRrd0lzqFruuEXYJUco230ypBI9qlvfD+m6jo6aXeWwltIypWMseCvIOevFM1TxNo+i3CQalfRwzOu4R4LNj1wATj3pt/4p0XTIreS+1COIXC74hgkuvrgDOPegCWHw/psEtjIlsDJp4dbZ2YkxhhhuT61YsNMs9MWcWMCwi4maeXb/E7HJb8aij13TJpLJIb2GRr9We12tnzQoySPoDWVr/jnSNE8P3upi4S4+yu8RjUkFpV6pnHBoA2r/TbPU4Vi1C2juY1cOEkXK5HQ4p11YWt7ZNaXUCS27ABomHykemKqaH4h07xDaefplws20KZAARsJGccitOgBkMMVvCsUEaxxqMKiLgAewp9VpdQtILyK1muI0nmOI42Ybn4zwKW5v7SyaMXdxHC0rhIw7AF2PQAdzQBU/4RzSvsDWZs4/s7XH2kx9jJu3bvrnmquoeCvD+qahNe3+mxzTzKFkLE4fAwCR0zjvW7XJaxq/iS2utYXT7JJY7f7N9kJQnzN7Yk+uBQBsnw3pJuPPNlH5vnJNu/20Xap/AcVSufAvh66WATaeuLdXWPa7LtDNuYcHoTUFz4wXSPEV7p+uolrBHafa7W4DcTKB86/wC8D275qaDxZbWmh2d74lePTZrwFo7cks23qOAM5xjPpQBeufDekXejx6VPYQmyiwYogMCMjoRjofeo4fCujw6TcaaLTfa3JBmSR2ff9STntU0XiDSZo7J4r+F0v2KWpVs+awGSB74Bom8Q6VBb308t9EsWnv5d05PELccH8x+dAGiqhECqMKowB6VkeHPDsHhy0uobY5+03Uly+BgBnOcAVJpXiXR9buprfSr+O5lgAaRUB4B4ByRyOO1aE88drbyTzuEjjUszHsBQBJRXKeGfiDo3iSaeGCdY50uZYI4zkmRUP3xxwCOa2l8QaU9nbXS3sRgu5hBBIDxJJkjaPfIIoA0aKxNR8ZeH9JuJrfUNUhhmhIEkZyWXIyMgD0ov/GXh7TJniv8AVYIZEKhlJJK7hkZwOBjvQBt0Vlan4n0fSPJ/tC+jiNwu6JQC5dfUBQTj3rQtrmG8tY7i2fzIpVDIwGMigCWiq2oalZ6VZtd6jcR20CdXkOB9KoWfi3Q7+xvLu11CN4LEZuXII8oYzyCPTmgDYorGsfFuhandSW9jqUM0kcRlbbnAQdTuIwRT9K8UaLrd09tpeoRXE0a7yi5BK9NwyOR7igDWoqnb6vYXep3WnW90kl3aBTPCOse4ZGfrVygAorkdW1jxLb3+rx2FikkNubb7IxjJ8ze2JO/OBXQajrOn6S1qupXUdu13KIYQ/wDy0c9FHvQBeooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKAOa8Rvez6jBaixuLjTwu+XycfvGzwpyenf3qxrSXl3o9taWNu8QunSOboDFH/F+nH41uEgdcUZA61NjB0ruWu42GJIIEijUKqKAAOwp9AIPSiqNlorIKKKKBhRRRQAUUUUAFFFFABRRRQBxFxHfaB491PVX0e51O01GCJY5bVA7wlBgoQTwD1zUF3HqGm+OT4jOh3d9a3umpbiCFVaS2dWJwVJ6HPJHpXdzTxW8ZknkWNB1Z2AFEcqSxh4nV0PRlOQaBXR5ZaaDrmgL4X1JtJmujaT3bT2lqVZ4BPyoGTggdD6VcsvDmr3Xw68VWd5p/2e+1C7upoIGYHIYgrg+9elUUDMPwtfyXempFLpN5pzwRojC5iCbyBg4wea1b2CS6sZoILh7WSRCqzRgFkPqM8VPTfMTzPL3LvxnbnnH0oA4rw74d1Dw9qHk3lomovPuU62HzODg8uGzj/gPHtTNM8N6loPiD7XewL4h86bi/mfFxbgn+6fl2j/AGcGu6ooAKKKKAOP8feH5NduPDzRWQuha6pFJMcfci53E+3So9ftrzTfH1n4gj0yfU7NbB7Ro7ZQ0kLl9wYKeoI449K7SigDypNB1nTbTRdV/siaTyNanvpLCAqZII5VYAAZxxkZx60+40LW7/wb42SfSpIbnVLwS29tuDF1wnfp2NepUUAVLCxgtbeEx28cUiwrGSqgEAdvpmrZAIwRkelFGc0AcN4Uju9F1HVNKu9Gux9p1K4uIbxIwYfLc5Uls8emK56w0zXxpmgaJJoN1H/ZmtCe4uWK+W0fmswZeeRhufSvWqKAOQ0bQHHjzxRfahYKbe8ECwyyKD5ihCGA9s1n3vhy6mvfG8i6fuN9aRxWjbR+8xEQQv4139FAHm+u6TcR6fo0sFjrUOrWtgsUd1poRtpwMxurHBGR3rtfDv8Aaf8Awjtl/bwRdQ8oeeIwAA34cVp0UAch47sr6WfQtQs7GTUoNOvvOubOLBZ1KEBgDwdpOcViLo2tXul+O5rjSntpdWhH2SDcCX/c7QOO+evvXpVFAHJaj4YkvvhnJpFnDFa30mmfZkO0LsYqMrkdASKwfBmgyx+I7C91G18QJeWVo9uDemPyIwQMqpUDcOOK9LooAyNPmkfxJqcb6MbVECbb/wCX/SuPbnjpzWvRRQAVzvivVItLk0gzaSNRFxfJArHH7hm6Pz/SuiqOSCKaSN5Y1dojuQsM7T0yKAJKKTcu7buG7GcZ5paACiiigAooooAKKKKACiiigAooooAKKKQkKpLHAHUntQAtFAIYAg5B7iigAooooAKKKKACiiigAooooAKKKKACiiigDn/FNtYPa+Zco8l2y+XbKjkMX7YA/nWbrouoNN0e3uFe/mUZntY2IaUBeTn2NbOpaDJfatHfw6hLbSRx+WoVVYDJ5IyODT7zRZLlrWeK9kju7ZSom2g7weuR07Vm022cVSnOblZWIfCMgk8PxlZGf5m4YnMfP3Dn06VuVR0nTE0u0MKu0jO5kkdurMepq9Vx2OijFxppMKKKKZqFFFFABRRRQAUUUUAFFFFAHAeNbXUtQ8YaHZbLCe0kaVo4bkMVZgnVwODjtV5NQ1O11GHw3otrp8FxBa/aJ22sIUBYgKqjnkg10F7o8V7rOn6i8jLJY79ijo25cHNVNU8Nm91ZdUsL+XTr3yfIeWJFbzEzkAg8cHoa6VUi4qL6fmc7py5nJGRP4yvj4c0+8tLGNry4vvsUkDv8quCwOD6ZFdDorau0Uw1xLYSLJ+7a3ztZce/PXIqjH4QtIdL06yjll22NyLreTlpH5yW+pJNdBUTlC1oouCle8irqd5/Z2k3d6V3i2geXb67VJx+lcl4cW10XwU/i/UVNzqFzZ/bLq4PLMCNwRfRRwAK7SaFLiCSGZQ0cilGU9wRgiua0PQLyx0qfw5qccV1o6xNFBMHIcxnjy2X2BxkViajdG1TxHLpses62unwWElubg28O9pY127lG48E468Vl6f4y11oNF1bU7SzTSdauEgijiZvOg8w4jZieDnjIHTNbOjeFLvSYorKTXbq80uGMxR2k8aE7CMBS+NxAHSqlh8P0tJtPim1a6utM0ubzrKxkVQI2GduWAywXPANAELeN7seFptTFtF5iat9gCZONvnBM/XHNVda8Z+Iba58QPp1jYmz0J1MrzO26ZSoYqoHQ8nk1Zn+G0c8ssZ1m8Wwa+F8lmAu1Jd4Y89SMjpWtc+ELa5t9ehe4lA1sgykAfu/kC8flQBheItT8Sv4u8MjRprOK1vRIwimL/OfKLEPjsO2O9dxZ/afscf2/yvtG3955OdmfbPNYur+FjqMelPZ6hNY3elnMFwiK/BTaQVPBBFbNjDPb2ccV3cm6mUYeYoFLn1wOBQBR1rxDaaD5X2uG6k83O37PCZMY9cdKXRfEFrrqym0huoxFjP2iEx5z6Z61Jqug6Vrfl/2tYw3XlZ2eYM7c9f5UaXoel6KJBpVlDaCTG/yxjdii4GP8R9SutL8CX8+nytBcPshWZTgx73ClgexANc74baPRviFaabpdve2lne2twJ4Lpid8kLLiYZJ+9uPPeu81rSbTXtFutL1BS1vdRmN9pwRnuD2I61k6H4VbTtWOqanqc2qXyW4tYZZUVPLizkgAdyQMnvildAaGmz6xJq2pJqdpBDZRuospI5NzSrjksO3NalYGiaVqFl4i1u8vb2Wa1vJUa1heTcIgF5wOwJ7VvZp3A5GHXPEGu6xqMegJYQWOnXH2ZpLvezTyAAtgL90DOM81BfeJteudW1iHQbex8jRVXz/tLNunfbuKrj7oA7nvV6TwjPb6xd32h65c6Yt84kuYEjSRGfGNw3A7SQOcUzU/BX2vUb25sNYutPXUkVL6OJVImwMbgSPlOOMildAGk+MG1bWLSGCBVtrrSRqClj8wJbG2siLxzreoWfh1dOsrQ3Wsm5DGV2CReWeDxyR7Vq33gSCS6sZdK1K60r7LafYWW32nzIOu3JHB9xzUul+B7LSl0MQ3UzDRhMIt2Pn8zrn6UXQHO6v4n8Q3vwx1q6gNraajp001vcSRM+DsON0fcHp1rsfDTa0+mxvrj2chaJDG1tvyfl53bu/0qBfCFj/Ymr6ZLLI8GqzSyynIBUydQPpVrQtLvdKt/JvtXk1FFVUi8yFEKADH8I5PuaLoDRurqGytZLm6kWKGJSzu3RQO9YGleKpfEN9E+hWDS6Tk+ZqEzbFf/rmvVue5wK6JgrqVcBlPBB6GsW38LWFjrK6hpby2BJ/fW8D4hm/3k6A+4waLgblear8QNfa1GqHTrNdMi1Y6fKDI3mSAy+WGXsMZHXrzXpOa5U+BbP8A4R19I+1y+W2o/wBob8DO7zfM2/TPFF0BBc+M7qCw8XTrbxE6E+2EEn958gbn86ydT8f69b6wLew060lg+0W9tudyGLzRbh+APX2rT1j4dxard6qU1m8tbTVsNdWsW3azgYDBuo6DI6GrkvgezlvFuDdShlu4LrAx96JNgH0IougOPttf1rXfFHh/UNOtbYajc6Zco4kZhDGVm2lvUj5eB71qSfEi5h8O2TXEVtbapc30ti5cs0MbR/efjkjGMD3rQT4fGzawfSdbuLKWzhlhEixo5ZZJC56jrk1an8B2J0SwsrO6ntbiwmM8F4CGk8xs7mbPB3ZORRdAZ9p47vJfA+tarJbwyXWlM6howwiuMAEMueQCDW14evPEl5P5+tWtlBaTQCSJYHLPGx/hYnrx3FEvhue+8N32k6rq0t2bxSpmEKR7B7BRityCIQWscKnIjQKCfYYougM/Qp9Znhujr1pBbOtwywCCTcHi/hY+hPPFavSsDwnpWoaTbaguq3sl01xfSTQ+a+4xxnG1c/h+tb2RRdAcdoWv+IfE8kmo6XFYW+kJdPBGtxuMsyo21nyOF5BwKz7zxxrjWmt63ptlaNo2i3DwyRys3nXAjx5jKei4ycA9cVsWXg+fSL2Y6JrlxZ6fNcG4ex8pHVWY5YKxGVBPb3qrffDuK6bULe31W6ttK1Sbzr2wjClZGON2GPKhscgUXQEsPjKa4vvEKQwRmHTLGK7gYk5ffGz4b8hWevjLxBqd5p1lo1jZedeaOmovLO7bIicZGByRzgVf1T4fQX+p3lza6pd2EN/bJbXdvAF2yKgIXBPK4BxxWhpfhC10q/tbqKeV2ttMXTlDYwUBB3fXimByHiXxVr+ofDHStb0l4LGW4uYkuPmbcG84JhSP4SQc57Guxl1ubQtHefxVc2EE5LeUYSwRsDIHzc5qrJ4Gs5PAqeGTdTLFG/mR3AxvRxIZA3pwTWnZaVcjS5LXWb/+05HyBM8CIVBGOg4oAy9B8eaPqnhq31K5v7aJzCj3Ch+ImYdPzpPDPi3SvGOmy20kttLNI00b2yndujVyoJB7EAH8a2dI0S10fRbXTYkWSO2iWIM6jLbRjJqHS9Fj8P6TLDp0azyh5ZU34UszsX257DJxQBi+GGfR/GereGImL2EFvFeWiscmFXLKY/plcj612Nc/4d0K5s9R1DWdYeN9T1AorrEcpDGgwsak9epJPcmugoAKKKKACiiigAooooAKKKKACiiigAooooAz9S1Q6cCxtLiZFXc7xrkKKjuNdt47e2kt0kuWul3xRxLklcZzWf4k1iOO6j0hrhbYToWmmb+FOmF9z+lF9b2ltDp13ZX8VoIIikRkXcroQOOo9AahtnJOpLmkovY2tO1CHUrNbi3ztJIIYYKkcEH3q1XPeDYJotIleZmbzriSVCwwSpPBx2z1roaqLujajJygmwooopmoUUUUAFFFFABRRRQAUUUUAFFFFABRRRQAUgdWJCsCR1welcv8SGvl8B3x0wzCT5PNNvnzBFuG/bjvtzXP3Q8PQ+DtXPw9eH+0TYn/AI9XJkK8ckf3v1oA9HV0fO1lbHXBzil3LxyOenPWvJ9NTSo/E2ir4GcuklhP/aYjZiCvljYZM/x7/wAetVtD1yC703wFpsNxJJeW98RdxjdmMhXGH/8Ar0Aep2+tWNzrN3pcMwa7s0R5kx90Pnbz+Bq/XlMOjeGNN+LutDWIYbYSwW01oZXZQ75bcVOeTnHFerUAFFcf4tn8RRajGLdZY9D2fv59PUPdKf8AdPRfdcmq2oT6vHoOnnwcZJ9KIP2mcEyXgGeSqyYBPrnn2oA7G7YhFwe9Yeta3BoWnfa7wyMDIsUccY3NI7HCqB6k0aA9g+ihtOnupwZT5r3hbzd/cMG6fTpVTxbb6NdaC0PiKYW9q0ibZi5QxyA5Vg3YgjrXjYqT9tY6qfwlvStWm1JJDLYXli8ZA2XKgbvcYJzV8SbvuvnHoa8ru9Z14+HvFGm6TqTaxDZWsbWupRp+8+Y/PGSPvMFGcj1q54Ds4ofETXNhq+mNbzWG17KxMhLsCCJGDHhhkg/WseV73KPR/NHH7wc9Pmpd5xndx65rzn4e+FNPvvDdjq9+k0l+lxMyyPI2QN7rtx/dwelV9JF/PrMXga7E3laXeG5kn5xLaj54hu/3jg+y0rPuM9O8whgC2CegJ60NLtzucDHXLV41f291f6/rH2/VdO03UItQzbzXRkE0cYIKeXg7dpHp75rqdQ0S2174oTW+qo89oukIxi3EIzeYRkjvTaa6gd55nygh+Oxz1pS5A5bH1Ncr4mstAsfDenWetS3ENlBcwx25iZtxcHCAkc4q74wGlHwneDX5JotP2jzXhLBgMjGNvPWs7t21HZG7ub+9+tAcn7rZ+hzXDeNbiKLTPD8UkssWhT3CJeyqWB8rYSgYjkAnGa5G4klj0Xxl/wAImkkFqLy0CqAw2wFV8xgOuCMnjnFaRjJrcls9nDlhkPx6huKzrPXrbUtHfUdNMt3CpZQsS/MxU4IAPuK4rwVpFpNqeoxrqGl3Wn3lsgk0+wMmwMD/AKw7jwSMA/SmfDGx0Kxt/KAhi1qK4uEaIuRIF3n+E+2KGrX1BHR23jdLrUpbCLSNW8+Ep5ytAB5e7oTzXS7+M7uOnWuX0RSPiJ4lJBwY7XB9fkNcPNrMSeFP7K+0Sf2iviMF4Ru3on2kHJ9FwaEm+oz2Aybc5bGOuT0oMmBkvgeua4G50G08Q/FDXLbVllmtE0+3Kxb2CbiXG7A7iqWpaBaX/ifxDBeLNLb2OkQ/Z4zIwUMFYBuOpGBSs+4jv9W1i20TTZL6/lKRJgcclmJwFA7kngCpNOvZ72xSee1ms3fP7mUjco7ZxXBTysfCvgS81Jma3juLdrp5OxKEKzf8CIr0Wpk3FblFTVNQn06xNxBZzXhUjdHCRu29yAev0p2marb6xpsN9p83mwTDKsPyII7EHjFWMhQSxAA6k1yfw+Umx1iaEYsp9WuJLT0KZ5I9i26km7XuFkddvb1NG5vU1HN5vkP9nCGXadgc4XPbPtXEaZcajJrkS+NHu7e983/RobfIsmPbDLyx/wB/H0pK76gd3ub1NXrUkwjNeaTT6o2stH41e5tLLzv9GGn5Nu4zx5jj58+xAFel2n/HuMV3YS/tNzKr8Jma54ij0O/0m1ktpJjqd19mRkIwh2lsn24rZqGezt7mSF7iFJGgfzIiwzsbGMj8DU1escxl+Jdcj8N+G73WJoXnjs4/MeNDgke1XrS4F1ZQ3AUoJY1faeoyM4pbq1gvrSW1u4lmglUrJG4yGB7Gq2rJaR6Ddpebo7NbdhL5eQVTbzjHtQBeBBGRyKKxfB50s+ENN/4R+WSXTfJH2d5SSxX3J5raoAKKKKACiiigAooooAKKKKACiiigAooooAKKKKAGPDFI2ZI1Y+pXNI8EUihZIkZV6AqDisTxRdNY2ouBqrWOFIRAgbzH7DkVTv8AWboLY29xdrp5a28+5nAB29BgZ46mpbSOedWEZNNHVKAuAowKWsnw9fy39jJ9oYPLBK0TOBjdjofxGK1qad0awkpR5kFFFFMsKKKKACiiigAooooAKKKKACiuf1LxYlnrT6VYadd6neRRrLOlsFxCp6bixAyfSmah4wisr6HT4NNvb3UJLf7S9rAo3Qx9Mtk4HORjvigDo6K52y8a6XqE+jxW3msdWWUw5XGwxj5lb0I6VmeIPHptvCGr6notjPcTafPLbNlAVR04LNyPloA7XrUcdtBCWMUMcZb7xVQM1m+HtYn1mwWa5065sW2Kf36gB8jOVwTxWtQBHHbwwsxhiSMscsVUDP1oW3hRiyQxqxO4kKMk+tZY8T6bLrg0mzka7ugf3ot1LrAP9thwv0PNLd+J9NtNWi0tZWuL+RgDb26mRowf4nx90e5oA05baCdlaaGOQr90ugOKkorl9V8dWemXl3ELK7uoLAqL25gQFLfPPPOTgHJx0oA6ikACjAGB7Vy+q+O7TTtSmsLfT73ULiG2W7ZbVAR5TZ+bJOO1SzeNrH7Jpslhb3N/PqcXnW9tboC5TuxycADIGTQBt3a/Ku0d+cCqcluJUKSxh1PUMMiuGgvdS1j4U6lqkupX9peWk13JG0cgVxsZtqNwcgeld3pV058LWV5cbpZDZpK5AyznYCfxNcdXCqpLmuaxqcqsMitlhj2Qwqif3VUAfkKSO0jiYtFbohbqVQAmrGi6mutaPb6glvPbLOpIiuE2OvJHI7dKv4rL6iv5iva+RlrDsXaibR6AYpPJw5cJ8xGC2OTWrijFH1Fdw9s+xkPapK6vJAjuv3WZASPxp/lHdu2fNjGcc1qUUfUV3D2z7GU8O8YkjDAHIDLnBoaHepV49ynqGGa1aMUvqK7h7Z9jntV0yfUbLyLe6msWDAiSFFJ47EMCCKg0Pw7HocdyRNPd3F5J5txcT4LSNjA6DAAAwAK6iiq+paW5he18jIjtY4STDAkZPXYgGaBaxiUyiBRIer7Bu/Oteil9RXcftn2MoRYYsEwx6nHJpv2SPeXNum5iCW2DJxWvRil9RX8we2fYyvJw5fy8MwwWC8mjyssT5fJGCdvUVq4oxR9RXcPbPsY1zp8N5ZyWl1bLLbyLseJ1ypHpio9O0uPS7FLS1Evkx52iRy5A9MnnFbtFP6iv5g9s+xz+q6NDrFn9lvBN5JYFljkKb/Ykc49qswWqW1vHBbwiKKNQqIi4CgdABWvRR9RXcPbPsZexvQ/lR5ZPVc/hWpRS+oLuHtn2Mvyyeqk/UVetRiEZ45qbFFbUcMqUr3IlU5lYKKKK7DMKjuIVubaSByQsilSVODgjFSUUAVdN0620jTLewsIxFbW6CONB2Aq1RRQAUUUUAFFFFABRRRQAUUUUAFFFFABRRRQAUUUUAZWtpqM8LW9ja288cqFWaaQjaT7YOazJPD13aQ6e9qIbuW1gMEsc5wJFOO+D0IrdvNTsrBlW8uY4S/KhzjNPmvra3gE088ccRGQ7MADUtJs5506c5NyepV0PTn06xZZypmmkaWTZ0BJ6D+VaVNV1kRXQ5VhkEdxTqaNoRUY2QUUUUygooooAKKKKACiiigAooooA5G80jXdL8XX2s+H4rO9i1GKNZ7e6maIo6DAZWCnIx2qC90XxFaeJ18R6PDYT3d1Yra3lrNMyorKSVZH2kkDJ4I5rtaqX2qWGmqraheQ2wY4XzXC5PtmgDhB4H1zSodAvdLls7zUdPmuJbhJ2aONzPyxUgE8HoO9W9P8ABWp/8IP4h0fUri3N3qtzcTLLEDsBk5GR25ru6KAMXw2daSxEGu2VpamFFSM21wZd+Bgk5UY/WtS7tYr6zltrgFopVKOAxBIPuOamooA5XR/Ct14anjtNGu430VyfNtbhP3kYIPKSDk8/3s/Wm2HhCfw1fmXwvdJHazzb7q0ul3hgT8xV/vA+xyK6yigArzbVPAFyPE2p3sGkabq9tqUgmIvLh4mhbGCOAQy8Zr0ms278RaPY362V5qVtDctjETyAHnpQBkR+G7tPEOqXwECQ3elx2kSKx+V1DZ7dPmFYdp4R8R6HJoN5pBsJrq003+z7mO4dgijduDqQMnB7cZr0XOaKAOK0vwjqVn8N9T0K5mhlvbs3JWQHCkyMSCeOOtbGgprFp4WWDVLe1gurWARReRKZVYKgAJyAeo6VuEhVJY4A5JPaqenavYassrabcpcrE+x2jyQG9M9D+FAFTwpd6pfeF7K516JItQkQmZUUqAcnHB6cYrYqvfX1vptlJd3soigjGXcgkKPwp9rdQXtqlxaTJNDIMpIjZDD60AS0UUUAFFFFABRRRQAUUUUAFFFFABRRRQAUUUUAFFFFABRRRQAUUUUAFFFFABRRRQAUUUUAFFFFABRRRQAUUUUAFFFFABRRRQAUUUUAFFFFABRRRQBkeJIrmTSpvscduzeWwczDouO1ZV5DZ3fw7M6xBgli3lGQZZflre1DSLXU2U3YkO0bcLIVBHoQOtNu9Esb20S2niPkou1UViox6YFQ463OWpSlKTflYm0w50u2x/zyX+VWqq2GnW+mwmK1DBCc4Zy386tVZvBNRSYUUUUFhRRRQAUUUUAFFFFABRRRQB5/41vNdsNdN3Jc6nbaDFAGM2lpG5jfnc0qsCSuMdKT4kW+n3vw/GqKkdzMfs3k3TINxUyKcj0zmum1TwjpGs3j3GoQSSNIoWRRO6q4HQMoODTdY8G6LrscUepWrSRRIESNZWRQAcjgHHFAG08scMJkldY41HLOcAfjWF4yOuP4dJ8LN/pRkQuYyocxZ+bYW+XdjpmrB8LaU/h+bRZYGlsZs+ZHJKzZ/EnNT32hWOo6bDY3EbiCDb5YjkZCmBgYIOelAHATeM7nS/AGqz2F5qF3qdrdR20ianCiy2jyFQAQAAwwcg9PetXwtP4ot73U49XF9Jpy2glt7i/8rzRLzuX93xtxgjNdHbeE9FtdJutOSxR7a7JNwspLmYnjLE8k0aX4X0zSPM+xRy/vI/LIknd8L6DcTgUAcp4KtvEuv+HNL1698TT77m2bfbJEgj5BCt0zuBwc9Pak0XxXqWtHSNGSfy9Ut7iRdWYKMqkJweO2/K/ma7rTNNtdH02DT9PiENtbrsjjBztFQWmgaZY61eata2iR3t6FE8w6vjpQB5xa694v1nU5NR0tL5kh1I2/2ZRELYQq+1t2fn3Yyc1bs4LWfwp45l1RY3uBfXYlZwCVCj93yemBjFdhJ4N0V9TkvhbPHNK4kkEUzosjDuyg4Jov/Buh6nqLXt3Z7pXIMoWRlWUjpvUHDY96AOItLnxHqbtYprc+nR2mg2103lIpcylWzywPBxzST+LNe1W28MWFs14kuoaaby5l08RCV2GFwN/AGTk4r0f+xrD7dcXYgAmuYBbysD96MZwP1NUbnwfot1p1nZSWm2OxXbbNG7I8Q6cMDkUAcpqt/r3/AAqN/wC3lkt76SdLaZzhWMTShdx28AlTzivQLK1gsrKG3tI1ihjQKiIMAACqX/CO6edCm0homa0mUq6u5YnPfJ5zS6Hp15pdkbW81Fr9EOIZJIwrqnYMR94+/FAGkQGUhhkHqDXI+Df9F8Q+KNNtxtsra8R4VHRC6bnUenPOPeum1CK6nsJYtPuVtbhhhJnj8wJ77cjNVdC0ODQrFoYpJJ5ZZDLPcSnLzOerH/DtQBHpUF7Fq2qPd6ul7DJMDBbKqg2q4+6SOT681r1i6N4bh0fWtY1KNwZNUnWV1AwFwuPXrW1QAUUUUAFFFFABRRRQAUUUUAFFFFABRRRQAUUUUAFFFFABRRRQAUUUUAFFFFABRRRQAUUUUAFFFFABRRRQAUUUUAFFFFABRRRQAUUUUAFFFFAGZrN5f2MJns4beSKNC8nmyFTx2HFNe+v7nTbe5023izLHvYTuV28ZxwKq+Kfss1r9murG6uXZCYjBGxw31HT8ao39xq1roOm2b29y7SoBdy2ybmQAcge56ZqG3c46lRxm9dLG7omp/wBr6XHdmMxsxKsuc4IODz+FaFUdH8n+y4RbW0ltGowsUi7WXHqKvVS2OmnfkVwoooplhRRRQAUUUUAFFFFABRRRQBl6l4m0fR7pLbUtQht5nG4RseceuB0FVte1DXLa3+0aDZ2NzAkBmdrm4ZN2Odq4B6jueK5TxpBdWviSbUPD9vrCau0CRqYbNZ7W6xnCvnhcZwTkVL4v1DWJRpek3ejajNY3EAk1N9Mi37j/AM8Qc8AnOT6cUAdj4d1mPxD4csdWhiaFLyFZRG/Vc9qs6hqNnpVm93qNzHbQJjdJI2AM9BWbJqZ07wet9pujXLeVCDFpyxhJQOgTb2IrP8a21vq3hO3+32epHMscyjT03zW0g5Dbe+DwetAG1Z6/pV/p8t9a30L2sOfMl3YCYGec9KNN8QaVrEUsmmX8NysP+sKN9z6153daf4q8R/DnV7SaCZnW6jezaWBYJ7qFSrMHToG4IGevcVoeEtOD6rqepH+3nu5rD7O/9o2awJwSQAFAyeetAHVxeMPD9xcJDBq1rJI8ZlVVfOVAyT+XNXjq1gtpb3RuovIumVYJN3Ehb7oHrmub+Hvh2Kx+H+jwajpscF5FAfMWSIB1Zshs+5BxWN4Z8P6tF4mTSNStnGj6BNJNYTt92ffzGP8AgALD8qAO1fxLo0erDTH1K3W8LbRCX5z6fX2rnr3xnqpm1S70jTILnS9HlaK5eSUrLKy/f8sYx8vv1rkIPDl7FqVzputnXiW1Q3cX2S0R4ZcyblbzduVx3ye1bL/2joNn4j0FdIvbubU7qeaymhi3ROJv779F2knOaAOwm8Y6DbW0c91qdvAskSTASPg7HGVOKtX3iDStOsYry9v4IoJv9U7Pw/GePWuL0zwnNY61frcWXnLF4fgtIpimQ7qrBlX9KxJdA1qzsvCV9ImpwwWemtbXCWUKyzQOSDkowOQQMHHNAHpd94j06y8Nya2J1ns0j3q8J3eZ2AHuTxTtCm1a6sftGtw29tJKQ8cELFjGpHAYnq3rjiuKn8PvF8J5rXS4b9zDci7EV7EElkCyiRhtHTODgV3ek6rZ6xpsV5p06TQyKDlT90+hHYj0NAEl+Lw2Mn9mGEXWP3ZnB2Z98c1Q8Oa6dbtJxcwfZb60lMF1Bu3bHHoe4I5BrVmmit4mlnkSONRlndgAo9ya5TwWv2zV/EOuRbvsmo3SC2JGA6xrtLj2Jzg98UAdHa6pZXt1c21pdRzTWjhJ40bJjbrg+lW657QNStbvxFr1rBpP2Ka1nVZp9oH2klchsjrx610NABRRRQAUUUUAFFFFABRRRQAUUUUAFFFFABRRRQAUUUUAFFFFABRRRQAUUUUAFFFFABRRRQAUUUUAFFFFABRRRQAUUUUAFFFFABRRRQAUUUUAJn1oOB16VgeK4Yv7Pa4e2uZ5FRghgYjyz/eODSSrFceFbWa5SfUdkQI+zty5x16ip5tTCVW0nG3S50A9qWsbwpI0vh23Z5jM3zAknJXk/Kc9x0rZpp3VzSnLnipdwoooplhRRRQAUjMqKWchQOpJxilqK6tYL21ktruJZYZV2ujDhh6UASKwZQVOQehFLXFfDyaC1OsaNGzB7TUZikGCUiiLfKFbp07Z4rsLqR4bSaSJQ7pGzKp7kDgUAS0gYNnaQcHBwelcMvxG82DbFZfv2tElQlvkaViAY/XI3A/jWfa65q2jSeIby2tYJrO31M+eZJCGIO0EIPbPeuhYeb3MHXh0PSSyqRkgEnAyetNaSNW2s6g+hNcpr9xP/wAJPoH2m0tpbSW5xBIZGEkT7Cc46HpR47sbZ7fTr5oVN1HqFuiS4+ZQZBkZqFTu1fqU6mja6HXUVzPizxUPD8tjaQmAXV6W2PcsVjRVHLHAz3Ax71Z8I+IW8R6Q9zNCsUsMzwv5ZJRiv8Sk9QaXs5KPP0K5483L1NuRtkZb0qp9tOcYGfSjWbsWGiXt4V3C3geUr67QTj9K4bQNJubrwa+rRT/8T7VbMyfanJPllxlVHooyK8vFVJwa5WdNOKe53X2twPuUfbG9BXm3gy403TLtLTVLO90zXktm85Lmd5FuQuCzoxOG6Z9RTNH+Jkmp61pyGGzNnqU5hijhkZp4eu1nBG3Bx26ZrldWt0ZryRPS/tjf3RS/bG/uiuDPibxFf63q9loelWckel3CRPLPMQZAVBwoHfk9eKtS+MDZ/wDCQR39uiT6UwMKKxPno4/dn6k8fWl7av3Dkidl9sb+6KT7Y390V53r3j670y+j02JdPhvYrNbq5+1yOF3N0jTaCSeDyat3njW8OkeHbrSdNWafW32LDPJs8s7CeTjoCKPbV+4ckDuvtjf3RUNv5Fq0htraKEytvk2KBub1PvWFbXXiEaBdyXdjatqkRcQRRSHy5cfd5PIzWhaS3r6RFLeW6RXzQhngVsqr4+7n0z3qfrFbuHJHsXrl4ry3aC6gjmib7yOMg/hUiXIjRUjjVVUYCqMACucGq6tbeDZtT1LSwNShgeRrG3ffuYZwoPuAKzfCviPUvEFrPKJtHlcRbkht5JN8T/3ZAwyB70/bVrbhyROyidIHkeGBEaVt0hAwWPqaf9tO7b8ucZxnmvPPA+p+K9Qu9Q/tT7BJaxalPC7CVy8e0/dQYwVHbNXNJmuP+Fk6hBqVlapdiwR0uLeVzuiLsApU8A8dqftqq+0HJE7YagC5UFCw6jcM/lQb/D7Pl3em7muDvtPtLL4uaJPaQJFLd2l007Lx5hGzBNN8X2FrF448JX8UKpdTaiY5JR1dRC+AfahV6t1qLkj2PQPtjf3aPtjf3RXB3NnH4m+IGpadqbymy060hMUKSsgLyZJfgjJGABWXpXizWLaz0rSrS3j1G6uLu7tFmuZSMCE/KzHvwefpT9rWtuHLDseofbW/uij7a390V51P8Qp9P0G4bVLW2i1WHUf7OKCU+RvIyH3Yzt289M9qrH4lXcPhrVbxrO1urrTbmGLNs7eTOsjAAqWGQeenqKPa1+4csOx6d9sb+6KQ3pyBgc9Oa5HQ/EWq3HiW80XXbG2t5oraO7ia2kLjYxI2tkD5gR24qt4JiXWoT4pvt0t3dSSLBuY4giDFQijoOmT70vbVluw5IdjuPtjf3RR9tY9FGKr1yM8S+G/HGmrYMyWmstJHPb7iVEoXcJAD0PBBpLEVX1HyRO1F6x6KDS/bHJ6CuI8RRL4f1jT9bsGeI3V7HbXsQY7J1fgMR/eBxzW5q9zq1tJZDRrGK8ElwqXJlk2+VF3Yep9qPrFXTUOSPY6YHKg0tIv3RS17a2OQKKKKYBRRRQAUUUUAFFFFABRRRQAUUUUAFFFFABRRRQAUUUUAFFFFABRRRQAUUUUAFFFFAGdqdjeXq7La++zRspV1EYYnPoT0ph0qe20y3s9Lu/sywrsy0YfcPxrUopWM3Ti3co6Tpkek2C20btIdxZnbqzE5Jq9RRT2LjFRVkFFFFAwooooAKhu5JobSWS1g+0TKpKRbgu8+mT0qaigDnPBek6hpWm3h1VI4pry+muhCj7/KDtnaW7muiIypB7ilooA5G3+H1jb29nGtxKTa3zXYYgZfP8B9hgflVyXwjBLpurWbXEm3Urgzu2BlDkcD8q6KitXVm92ZqlBbIxdc8Ptqw097e8a0msJvNjcIGydpXkH2NP1rQ31nR4rR7xopo5Y5ROqA/MhyDjp1Fa9GKlTkreQ+SOpz+peGX1JLCeS/ddSsS3l3QjX5twwwKdCDx+Vael2lzZ2pjvLoXUhbO8RLGB7YFXaKTk2rDUUncgvokuLGaGVQ0ciFHU9wRgiuN0PStX0vTZNDaQLa28ZSy1BGBcL/AAqyHuB36HFdwyh1Knoai+yx+lcGIozqP3TeElFanGQ+Fbi51m11HX9UbUXskdbeMQLEqlxtZjjqccUmjeFbrQ3ht7TWHbTYJC0dtJbozKuc7PM645+tdp9kj9KPskXofzrm+q1jT2kTnNK0OLStS1W8ild21K4Wd1YcIQoXA/KqGseDbPWfE+n6zNPLG1oMSQp9y4AO5A/+6eRXZfZIvQ/nR9ki9D+dL6pVvcPaROP1TwxJc68dY0zUGsbx4RBNmFZUlUHIyrdxk8ip7jw99sm0ea8u3km0uUyhxGq+aSpXkDgde1dT9ki9D+dH2SL0P50fVK3cPaRMjUbWS9024toLmS1kljKrPH96MnuPem2tpLb6TFZyXUs0qQiM3L/fY4xuPv3rY+yR+/50fZI/f86X1OqHtYnO2el3lp4cGnHVria7CkC/kUGTJOQSOhx0qho/hRrHxFJrmo332y/e3+zhkgWFQmc8hepz3Ndj9ki9/wA6X7JF7/nT+qVQ9pE5DTfDd3pGsXU9jqrCxu7l7qW0eFW+dvvAP1AzUk/h6ZvGC67a6i8BaBLeaDylZZEVi3U8jr2rqvskfv8AnR9kj9/zp/VKoe1icrrPh2bUte07VbPUXs7ixSSMYiVw6vjIOf8Ado8R+HpNcl02e3vnsrjTrg3EUixq+W2lcEH2JrqvskXv+dH2SP3/ADpfVKoe0ichqHhu6m1RdU0zVHsdQa3FtcSiFXWZRyCVPAIOcH3rEv8Awhe2epeGINAnlgisftJmvGRZCGcDlg3Usc16X9ki9D+dH2SL0/WqWFqoPaROGk8CWkuiC0N5cC9W7+3fbzgyGf8AvEYxjHGOmKde+ErnV9An07WNWM/myxSB4rZI9mxw2MDrnHeu2+yR+/50fZI/f86X1Wt3D2kTnYtEii8Vy64JWMktmlqY8DbhWLZ+vNUdA0u/8P3c2nRRxzaRJK80EgfDwbjkoV7jJOCK7D7JH7/nS/ZI/Q/nR9Uqh7SJRFc9Hpd/qPimHVNWSKCCwV0s4EfeWZuDIx6DjgAV1/2SL0P50fZYvSpWDqoPaxOP1HStQ1zXbUXyRQaVYTCdFD7nuZAPlJ/uqDz6mpfE1hqt++lf2NdyWvk3ySXBRgFaIZ3K3c59B3rq/ssfofzo+yx+h/OqWEqJoPaRJV+6KWgcCivVWxzBRRRTAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKztV1K405PMhsJLqNVLyMkirtA9j1p0msW0Wi/2o7EQGMSAY5OegA9TSuiHUim0+hfoqppd+uqaZBexoyLMu4K3UVbplJqSugooooGFFFFABRRQTgZNABRVeDULO5leK2u4ZZE+8qSBiv1Aoh1C0uZnit7qGWSP76JICV+oFAFiio/tEOxX81NrNtB3DBPpWYvibS28QT6P9pRbqCJZX3MAuCSAM565HSgDXoo60UAFFIrK2dpBwcHBoDAkgEEjqM9KAForL8S6tJoPhm/1SG3Ny9pC0ohBwXx2rFk8cG0ttcub3TpFh0i2huDsbJl3puIHuOlAHXUVz1r4y0++k0IWYeWPW43eCQdE2LuIb37fWtlNQs5bh4I7qF5kGWjWQFl+ooAsUVH9oh8rzPNTZt3btwxj1z6U9WDKGUggjII70ALRUF1fWtioa8uYrdWOAZXC5/OsHxb4nu/DumRanZWUN9ZblEr+ftK7mCgqMHPWgDpaKjnuIbaEy3EqRRjq7sFA/E01by2e1+0pPG0GM+YHG3H16UATUVWi1Kynt3nhu4JIU+9IsgKr9TVXRtTuNQs57i9t47VUmdUKzrIHQHh8jgZHbtQBp0VBFfWs0xhhuYnlChiiuCceuKRtQs0uRbtdQiYnAjMg3Z+lAFiiiq1zqNnZuqXd1DCz/dWSQKT+dAFmiobi8trWNXubiKFWOA0jhQfzpzzxIiu8qKrfdJYAGgCSiq8GoWdyFNvdwyhiVXZIDkjqBTp7uC3VzLNGmwZO9wMZ6Z9KAJqKz9EvrnUdKjub62S1lZmBjSYSgAEgHcODkc1Yh1GzuLhoILuGWVfvIkgLD8KALFFQR31rLctbx3MTzJy0auCy/UVPQAUVl6VrP2/UNRsZovIubGUKyE53Iwyjj2PP4g1dnvrS1z9puYYsYzvkC4z060AT0Vi6rr0un6zp1lDaGeO9SVjKG4TYm4fn0qXQNa/tfwza6tdQiz8+PzGjdvufU0AatFIjrIgdGDKwyCDkEUtABRRRQAUUUUAFFFFABRRRQAUUUUAFFFFABRRRQAUUUUAFFFFABRRRQAUUUUAFFFFAHPeK5rFrFra8u57eVkLRrDuy/twOfpVWSw1S80rTJPLgXyIdzwSAgb8cHj09K6oqG6gGlqeW7OedHnk22YnhBLiPwxaJdoqOF4Az0989626OnSimlZWNYR5IqPYKKKKZYUUUUAFYPjeC/uvBOqw6QHN29uwjEZwx9QPfGa3qKAPPNPuPD154au7PwZara6wNOdUT7K0csZ24wxIHOfU9ax9C/s641DwlD4aspINRtM/2o3kMjInlkOshI5Jfp19a9a2gHIFLgA5oA8YtNQ3aHo2hpBdm/tPEAe4QwOBEvnMQS2MYIIrb1O08O6N8VLq71+wgitbyxjMMslsXV5hIS3IB+bkV6ZgelIQD1GaAEjKtGpT7pAI+lct4xtPEl00I0iY/wBnD/j6gtWEd1IPRXbgD8j711dFAHDXFtqk3hW3i8DI+nRrIRdQ3Klbk/3grPkBv9o5+tbHg9LKGwmitLG9s7hZP9JW+3GR3x94uSQ31HFdDRQAjKrqVcBlIwQR1rnfHcG/wLqywx7naDGFXJbkV0dFAHkF3oep6R8RdPsdMgkFhLFd3dk6g7YJXhwye3zfMPrUOkxWNxb+FrDSbKWLxDb3SnU2MDK6Jz53mMRyG7cnORivZaQAA5xQB4OtjrI08t5d2I47htAWHa2PLZjmTHpkjn2r3aKJYYUijGERQqj0Ap9FAHnviBtNsviQ114ttzLp0lgqWUssJkhjcMd4IAIDEY5qfx+9tP8ACqSTSom+y7oWjSOIjCCRTwuM9Pau6IBGCM0uARigDzjxhfWeqXnhjV5RJd+Go7iX7XiJiofbiNnTGdoOe3cVlCy0+/8ADniUiW50vw9c38T2EiWzFQyhdzeXj/Vlx6YNet4G3HajAxjtQBwfw/nXVrXVrK50ywezjdEF5a2phivcjn5GHUdD2rN8M3drF4R8U6JCrx3UMt8624hZcISduOMY9MV6eAAMDgUYHpQB5NpeiLpd34BuNMs2huZrKUXMoU7nY2+QHP8AvetZFvJZeR4ZhOm3C64mtxHUrqe3ZWEhc5G9hznjAB6CvcMVz8fg+xGrxX9xcXt08EhlhiuLgvHGxzyF/HigDoK8r8cvaad4suNRt5be7v2gRG0u+sHmE+OgicD5Sc/T1r1SjAJzQB5P4wFwfFlhc6sLaz02XTAkf2yza4ihlJ+ZcKflbGOT6VPd6Mk3hDwdpzzy6parqSB5fLZN6YfqDyF7c9q9RIz15oxQBxHizwta6b4d+3+G7FYLvTbpdQjjgXG8qMOoHumRgU/wnYHXdK1DXdcs/m1p/MS1uEz5UC8RqQe+Bu+prtKKAPLvC11Nd/CbU9E0NpItZtoblUj8tkKMzvtAJAHTpir/AIUvPDMmkQWWgWq2uuJZMuyS2ZJo5Avzb2I9fU816FjFGBnOOaAPEvBdip1Tw+t9qUVlq1jcH7RbLp0i3EzEEMskmSGU5znp0r1n+3rf/hKP7C8m4+0fZ/tHmeWfL25xjd0z7VqYGc459aMc5oA5bxPGNG1ix8TQghYiLW+x0aBjwx/3W5+hNYGraHa6r408Uz39kLpU0qLyDIhZQ2H5UdM9OetehXdrDfWctrdxiSGZCjoejA9RToII7a3jghGI41CKCc4AGBQBjeClc+BtENyreaLKMN5g+YHaM5zUviltMt/COotrVs0+mpAxnhjU5ZO4AHNbFQ3lpDfWctrcrvhmUo65xkHqKAK2hSWcvh+wk0uMxWTW6GBDnKptG0c+1X6ZDFHBCkMKBI41CqqjAUDoKfQAUUUUAFFFFABRRRQAUUUUAFFFFABRRRQAUUUUAFFFFABRRRQAUUUUAFFFFABRRRQAUUmQOpxS0AFFFFABRRRQAUUUUAFFFFABRRRQAUUUUAFFFFABRRRQAUUA56UUAFFFFABRRRQAUUUUAFFFFABRRRQAUUUUAFFFFABRRRQAUUUUAFFFFABRRRQAUUUUAFFFFABRRRQAUUUUAFFFFABRRRQAUUUUAFFFFABRRRQAUUUUAFFFFABRRRQAUUUUAFFFFABRRRQBz/iuKIWDXD21zPIiMEMDkbP9o4NVb2K5vNA0mSNpr2NMNObeQq0g2HnOR3rZ1OxvL5dltffZo2Uq4EQYnPoe1N/suW20qCy025+zrCoXcUD5H41Fnc5J05Oba2sP0Se3uNJheyZzFyB5hJYHPIOe+a0KpaVp0elWCWsTMwUklm6sSck/nV2rWx0U78q5twooooLCiiigAqtqN0bLS7m6VQxhiaQKe+ATirNQX1qL3T7i1ZiomjaMsO2RjNNb6ie2hzWjeJdYvdPh1bUNNgt9MkszcFklLSKQoPI6YPNUdC+ILarrVnaPHamO/VjEIJGZ4SBkCQYxyPSuosNGhsvDkGjsxlhitxblmGCygYqho/hu70iSCNdWeaztxtjheBd23GAC/U4rdSpO+noc7jUVrMxtB1TxPceLtZt5orN4ILiJXVpm/dKVz8gxzxzz3rQn8XSw6Rrd4LZC2mXRgVd33xleT6dat/8ACOXEHiG51PT9Sa3W7ZGuIDEHDlRjgnkcVSvvA/2yTUETVJ4bO/lE8luqD7/HO7rjjpVOVOUrvyBKpFaE2peINVTX49J0iwgnle0Fx5k0hVU+bGDjrVa58S64usW+k2umWxvZLH7TJ5kxCRtuwRwORW4mjIviBdVErbxai22Y4xuzmkOiRnxN/bPmt5n2X7N5eOMbs5qFKC6FOM+4eG9YOveH7XUXh8h5lO6MNnaQcHn6iuf+Id7Or6FpcbzRw6lfbLnyCQ7xpGzlARzltoFdHoWkJoWjw6fFI0qRZwzDBOST/WoPEfh6PxBaW6+e9rc2k63FtcxgFopB3weoIJBHoaylbmdtjWN7anOeA7508Qa1pUIuUsYo4Lq2guifMtxIp3RnPPVc4PTNdRoF3qt5ppk12xjsbkSsoijk3goD8rZ9xVfw/wCHP7Fmvru5u3vtQ1CQPcXLoF3bRhVCjoAKTwjYapp2itDrl5Jd3DTyOrysGZULfKCRx0qSjcooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKAM7U9WTTMGS3uJV2lmaKPcEA7mkudctLaxgusvKtxjyUiUs0meeBVTxS8R02SNtVFg/ls23cv7wY6YPOPpWDcb5G8OXFy50+2MLIzKduxiowMnp0qHJpnFVrThJpf1qdnZ3X2y2WXyZYcnGyVdrD8KsVjeGbma50+UzSmdEndIpj1kQHg1s1S1R1U5c0UwrF1bxGujarZ213Y3H2a7kSFLxCpRZGOApGc/jjHNbVcj8QfM+xaR9mKLdHVIBbvKMxo+TguByR7DFMs66io7cTC2jF0yNMFHmGMEKW74B7VJQAUUUUAFFFFABRRRQAUUUUAFFFFABRRRQAUUUUAFFFFABRRRQAUUUUAFFFFABRRRQAUUUUAFFFFABRRRQAUUUUAFFFFABRRRQAUUUUAFFFFABRRRQAUUUUAFFFFABRRRQAUUUUAFFFFABRRRQAUUUUAFFFFABRRRQAUUUUAFFFFAEM1pb3DK08McjL90ugOKdLBFNH5c0aSJ/dZcj8qKKBcqHJGkaBI1CqOgAwBTqKKBhWTqHh2z1TVrW+vXnkNqweKDzSIg4JIYr3IzRRQBrUUUUAFFFFABRRRQAUUUUAFFFFABRRRQAUUUUAFFFFABRRRQAUUUUAFFFFABRRRQAUUUUAFFFFABRRRQAUUUUAFFFFABRRRQAUUUUAFFFFABRRRQAUUUUAFFFFABRRRQAUUUUAFFFFABRRRQAUUUUAFFFFABRRRQAUUUUAFFFFABRRRQB//2Q==)





class BankAccount

{

public:

BankAccount(double balance, double rate);

BankAccount(int dollars, int cents, double rate);

BankAccount(int dollars, double rate);

BankAccount();

void update();

void input();

void output() const;

double getBalance() const {return (accountDollars + accountCents\*0.01);}

int getDollars() const {return accountDollars; }

int getCents() const {return accountCents; }

double getRate() const {return rate; }

void setBalance(double balance);

void setBalance(int dollars, int cents);

void setRate(double newRate);

private:

int accountDollars;

int accountCents;

double rate;

static int dollarsPart (double amount) const {return

static\_cast<int>(amount); }

static int centsPart(double amount) const;

static int round (double number) const {return

static\_cast<int>(**floor(number + 0.5)**); }

static double fraction(double percent) const {return

(percent/100.0);}

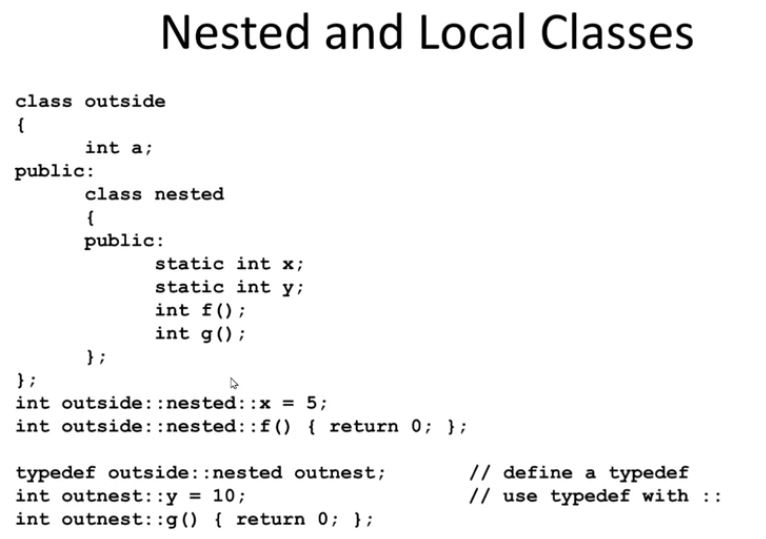
};

What changes if we do 4 private functions static?

I am saying to compiler that these 4 functions will not access 3 data members (accountDollars, accountCents, rate). If I access them by accident, I will get a compiler error.

Still I don’t like these 4 functions because they do nothing with my class data.

**Nested and Local Classes**



🡪 private

If you have a nested private class, nobody can make object of your classes outside of your class.

**VECTORS**

**Vector Introduction**

* Recall: arrays are fixed size
* Vectors: “arrays that grow and shrink”
  + During program execution
* Vectors are as efficient as arrays but they are more flexible to use
* Formed from Standard Template Library (STL)
  + Using template class

**Vector Basics**

* Similar to array:
  + Has base type
  + Stores collection of base type values
* Declared differently:
  + Syntax: vector<Base\_Type>
    - Indicates template class
    - Any type can be “plugged in” to Base\_Type
    - Produces “new” class for vectors with that type
  + Example declaration:

vector<int> v;

**Vector Use**

* vector<int> v;
  + “v is vector of type int”
  + Calls class default constructor
    - Empty vector object created, v has zero element in it
    - v has space for maybe 5 integers but I have zero element
* Indexed like arrays for access
* But to add elements:
  + Must call member function push\_back
* Member function size()
  + Returns current number of elements

**Vector Efficiency**

* Member function capacity()
  + Returns memory currently allocated
  + Not same as size()
  + If capacity same with the size it’ll take more time to push back new element to allocate new space and delete old one and etc.
  + Capacity typically >= size
    - Automatically increased as needed
* If efficiency critical:
  + Can set behaviours manually
    - v.reserve(32); *//sets capacity to 32*
    - v.reserve(v.size()+10); *//sets capacity to 10 more than size*

