C++ / OBJECT ORIENTED PROGRAMMING

Our first aim is writing reusable, maintainable and managable codes.

#include <iostream> 🡪 like stdio.h

using namespace std; 🡪 there will be a complete chapter about namespaces

int main()

{

int numberOfLanguages;

cout << “Hello reader. \n”

<< “Welcome to C++. \n”;

cout << “How many programming languages have you used?”;

cin >> numberOfLanguages;

if (numberOfLanguages < 1)

cout << “Read the preface. You may prefer\n”

<< “a more elementary book by the same author.\n”;

else

cout << “Enjoy the book.\n”;

return 0;

}

Blues are language keywords, they are part of the language.

<< 🡪 stream insertion operator: send string to standardoutput

cout (character out) / cin (character in) 🡪 global variables / true one is object, we’ll see later

In C, we can put “&” in front of a variable and get its address but we can’t do this in C++. If an integer is an integer, use it as an integer ; if sth is an address then use a pointer.

So C is more efficient than C++ (%50).

int i = 7;

char c = ‘a’;

cout << i << c;

<< is smart enough to know what kind of parameter it is getting.

Compilation:

g++ -c hello.cpp 🡪 produces object code (hello.o) , -c means compile only (compiling)

g++ -o hello hello.o 🡪 -o means output hello, executable program (linking)

./hello

./hello > res.txt 🡪 all things not printed to the screen but printed to the res.txt

Compiler creates assembly version of your code, linker finds your functions and links them then creates executable file.

Some compiler switches:

1. -c
2. -o
3. -S
4. -E
5. -g
6. --std=c++11 //to use C++11

g++ -S hello.cpp 🡪 creates hello.s file, produces assembly code

less hello.s 🡪 shows hello.s , hello.s is assembly version of your program.

wc -l hello.s 🡪 shows line number of assembly code

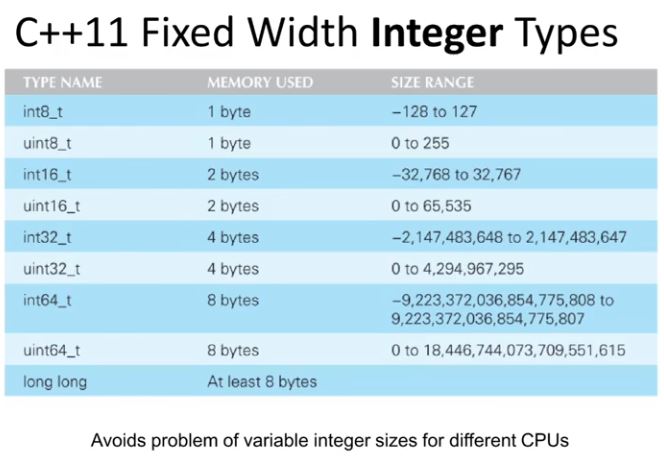
g++ -E hello.cpp 🡪 creates too many lines so direct it would be nice, stop compiler from compiling your program and run only the preprocessor

g++ -E hello.cpp > helloE.cpp 🡪 directs what -E does to helloE.cpp

helloE.cpp is very large. we unfolded #include <iostream>

man gcc 🡪 shows manual page of gcc

\*\*\*sizeof is an operator (for C). It is not a function. It is part of the C programming language.



int uses different memory spaces for different systems but above ones use exact byte of memory. These are not part of the language, they are defined in a header file. They are user defined new types.

These are not C++ keywords. They are defined in header file.

**New C++11 Types**

* auto (C++ keywords)
  + I have a variable x and I don’t know its type. auto deduces the type of the variable based on the expression on the right side of the assignment statement.

auto x = expression;

* + auto determine the type by looking the expression.

auto x = sin(0.5); 🡪 type of x will be double

auto x = 3.0; 🡪 is x a double or float? (double)

75 🡪 int 3.0 🡪 double

75u 🡪 unsigned int 3.14159L 🡪 long double

75l 🡪 long 6.02e23f 🡪 float

75ul 🡪 unsigned long

75lu 🡪 unsigned long

auto x = 3.0f; 🡪 now x is float

* + You can’t do : auto x;
* decltype (C++ keywords)
  + Determines the type of the expression.

decltype(x\*3.5) y; 🡪 x\*3.5 is a double so y is declared as a double.

**Assigning Data**

LeftValues = Right Values;

Rvalues

Lvalues must be variables. Rvalues can be any expression

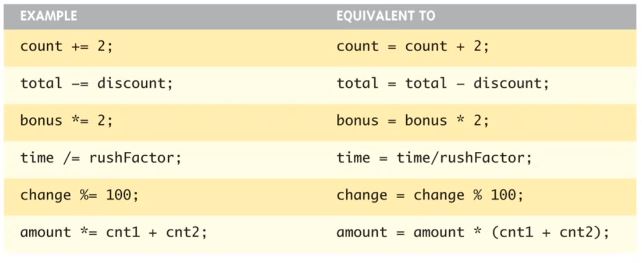
Lvaluesd

\*ptr = x+y; 🡪 we don’t know \*ptr or x+y will become first

z = 7/x+y\*3; 🡪 we don’t know 7/x or y\*3 will become first

z = x = y; 🡪 first y is assigned to x and then z is assigned to them (right to left). assignment is the only operator that happens right to left

2 + 3 + 4 🡪 left to right



- Don’t do x=2; if x is double, do x=2.0;

**Literals**

* They are literal because you literally type them.
* Considered “constants”: can’t change in program (during execution)
* 2 //Literal constant int
* 5.75 //Literal constant double
* ‘Z’ //Literal constant char
* “Hello World” //Literal constant string

**Escape Sequences**

* “Extend” character set
* \ preceding a character

**Raw String Literals**

* Introduced with C++11
* Avoids escape sequences by literally interpreting everything in parens:

string s = R”(\t\\t\n)”;

* The variable s is set to the exact string “\t\\t\n”
* Useful for filenames with \ in the filepath

**Constants**

* const is C++ keyword.

const int NUMBER\_OF\_STUDENTS = 24; //use this instead of define

= above is not an assignment operator, it is initialization operator.

Compiler knows NUMBER\_OF\_STUDENTS

#define NUMBER\_OF\_STUDENTS 24;

Compiler doesn’t know NUMBER\_OF\_STUDENTS bc preprocessor changed all the occurrences of NUMBER\_OF\_STUDENTS with 24.

#include <iostream>

using namespace std;

int main()

{

const double RATE = 6.9;

double deposit;

cout << “Enter the amount of your deposit $”;

cin >> deposit;

double newBalance; 🡪 you can define variable anywhere

newBalance = deposit + deposit\*(RATE/100.0);

cout << “In one year, that deposit will grow to\n”

<< “$” << newBalance << “ an amount worth waiting for.\n”;

return 0;

}

cout prints to the standard output, not screen. cout by default connected to the standard output. For many OSs and shells, standard output is the screen.

* 17/5 🡪 3
* 17.0/5 🡪 3.4
* 1/2 🡪 0

**Type Casting**

static\_cast<double>intVar

doubleVar = static\_cast<double>intVar1 / intVar2;

First intVar1 will be double and then we divide double by integer so result is double.

Explicitly type casting 🡪 you do (do this if you can do)

Implicitly type casting 🡪 compiler does (17/5.5) . This expression causes an implicit type cast to take place, casting the 17 -> 17.0

int a,b=2;

a = b++; 🡪 a is 2, b is 3 BUT increment is done first , assignment is done after. ++ operator increments the value of b first but returns original value of b.

Unary operators (take just 1 parameter) have higher precedence (-a, a++, ++a, static\_cast).

You will be careful about these:

b = ++a - (a+b);

which one will be executed first?

=

b -

++ + 🡪 we don’t know if ++ or + will be done first

a a b so we can’t say certain thing about the result, then don’t write this kind of expression

When you do loops or sth., use “++i” instead of “i++” because “++i” doesn’t store the initial value so for optimization, “++i” is better.

b = 2 \* (n++); 🡪 don’t use this kind of things because there is no meaning for parentheses.

**Console Output**

Any data can be outputted to display screen:

* Variables
* Constants
* Literals
* Expressions (which can include all of above)

Cascading : multiple values in one cout

cout << “Hello World\n”; is same as cout << “Hello World” << endl; (you can think endl as constant)

cout << a << b;

<< What does stream insertion operator (<<) return?

<< b It returns cout object itself.

cout a

a + b + c

+ + returns integer.

+ c

a b

a = b = 2;

=

a =

b 2

**String Type**

Must add “#include <string>“ at the top of the program.

“+” operator on strings concatenates two strings together.

“cin >> str” where str is a string only reads up to the first whitespace character.

string a, b, c = “hello”;

You can do these:

* a = c; 🡪 this is copies content of c to a like an integer
* a += c; 🡪 a may need some new space
* if (a == c) …
* if (a < b) …
* cout << c.size(); 🡪 prints 5
  + I don’t know if this is null terminated string or not.
* cout << c[0]; 🡪 prints “h”

|  |  |
| --- | --- |
| size / length | return length of string / characters - bytes |
| clear | clear string |
| capacity | return size of allocated storage (how many characters can it hold before allocate more space) |
| max\_size | return maximum size of string |
| append / += | append to string |
| insert  (index1, str, num) | insert into string  index1 -> starting index of the original string that we are inserting str  num -> how many characters do you want to insert from new string (optional) |

You can send strings to function just as you are sending integers to a function as value or as address.

In C++, I don’t care how they represent a string. I don’t care if there is a null character at the end.

#include <iostream>

#include <string>

int main(){

std::string str (“Test string”); //another way of initialization of string

std::cout << “size: “ << str.size() << “\n”;

std::cout << “length: “ << str.length() << “\n”;

std::cout << “capacity: “ << str.capacity() << “\n”;

std::cout << “max\_size: “ << str.max\_size() << “\n”;

return 0;

}

|

|

size: 11

length: 11

capacity: 15

max\_size: 429496729

11 and 15 means: “There are 4 empty spaces available before I need to reallocate something.”

**Vectors**

Arrays of anything. You can use it with “#include <vector>”.

not necessary

vector<int> foo (3,0); 🡪 foo is list of integers, there are 3 integers in it, they all initialize with 0

vector<int> bar (5,0);

bar = foo;

foo = vector<int>();

cout << “Size of foo: “ << int(foo.size()) << ‘\n’; 🡪 Size of foo: 0

cout << “Size of bar: “ << int(bar.size()) << ‘\n’; 🡪 Size of bar: 3

|  |  |
| --- | --- |
| size | Give the number of elements. |
| [] | Give the specified index. |
| push\_back | Add new element, takes 1 argument which will be added. |
| pop\_back | Take last element and erase it. |
| insert | Insert another vector at any place, you need to move elements after position. |
| erase | Erase. |
| resize | Resize. |
| at | Access element. |

vector<int>::size\_type sz = bar.size();

vector <int> a,b; 🡪 size of a and b are 0, they are initialized.

(if you do “cout << a[7];“ this will give an error but if you do: “cout << v.at(7);” this will check the index number and if it is bad, it will not run. But it is more expensive than index operator.)

In all OOP languages, when you declare a variable it is initialized.

vector<string> strA;

strA.resize(100); 🡪 now strA has 100 elements.

vector <vector<int>> a; 🡪 each element of a is another vector

a.resize(10);

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

a[i].resize(10); 🡪 now a is 2 dimensional array (10x10)

int f(vector<int> vect){ 🡪 you don’t have to send size of this vector, bc vector already knows size of it

cout << vect.size(); 🡪 prints size

}

**List**

Two directional, you can go forward and backward.

|  |  |
| --- | --- |
| push\_front | Insert element at beginning. |
| pop\_front | Delete first element. |
| push\_back | Add element at the end. |
| pop\_back | Delete last element. |
| insert | Insert elements, better use lists because insert is very expensive for vectors. |

You can have vectors of lists, list of vectors, vectors of vectors etc.

**Forward List**

Single directional list.

**Formatting Output**

double price = 78.5;

cout << “The price is $” << price << endl;

You might get: The price is $78.500000 or

The price is $78.5

input output system

“Magic Formula” to force decimal sizes:

cout.setf(ios::fixed); 🡪 I am calling setf function on object cout with “ios::fixed” flag

cout.setf(ios::showpoint);

cout.precision(2);

Now you get: The price is $78.50

**Error Output**

* Output with cerr
  + cerr works same as cout
  + Provides mechanism for distinguishing between regular output and error output
* Re-direct output streams
  + Most systems allow cout and cerr to be “redirected” to other devices
    - e.g., line printer, output file, error console, etc.
* If you are not printing a normal message but you are printing an error message saying that “I couldn’t find the file.” or “This name is not available.” etc. I am gonna hold my program, in that case you should use cerr.

cerr << “error”;

When you are prompting user for input, wait in the same line with the question.

Comments: “//” or “/\*…\*/”

Identifier naming:

* ALL\_CAPS for constants
* lowerToUpper for variables
* Most important: MEANINGFUL NAMES!

**#include**

* preprocessor directive
  + Executes before compiler, and simply “copies” library file into your program file
* C libraries are available to C++ programs.
  + to be able to include a C header, for example math.h, you put a c letter in front of that header name so you include <cmath>
  + suggestion is don’t use them in C++

**Namespaces**

* Namespaces defined:
  + Collection of name definitions
    - collection of classes, cin and cout are in the standard namespace for example
    - string library, list library, vector library are all inside this standard namespace
* For now: interested in namespace “std”
  + Has all standard library definitions we need
* Examples:

#include <iostream>

using namespace std;

* + - Includes entire standard library of name definitions

#include <iostream>

using std::cin;

using std::cout;

* + - can specify just the objects we want

or you can do specifically:

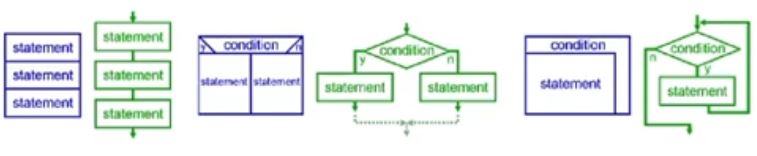
std::cout << a;

Libraries are smaller, namespaces include libraries.

Using your own namespaces will be on EXAMS!!!

string dogName; 🡪 dogName is OBJECT of string CLASS (types that we have created)

int actualAge; 🡪 actualAge is VARIABLE of int TYPE (fundamental types)



Loop structure

Sequence structure

Condition structure

**STRUCTURED PROGRAMMING**

Write your statements that easily replacable.

![Graphical user interface, application

Description automatically generated](data:image/jpeg;base64,/9j/4AAQSkZJRgABAQEAYABgAAD/4RDaRXhpZgAATU0AKgAAAAgABAE7AAIAAAAFAAAISodpAAQAAAABAAAIUJydAAEAAAAKAAAQyOocAAcAAAgMAAAAPgAAAAAc6gAAAAgAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAE1lcnQAAAAFkAMAAgAAABQAABCekAQAAgAAABQAABCykpEAAgAAAAM4NwAAkpIAAgAAAAM4NwAA6hwABwAACAwAAAiSAAAAABzqAAAACAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAMjAyMToxMDowNyAyMjo0OTozNAAyMDIxOjEwOjA3IDIyOjQ5OjM0AAAATQBlAHIAdAAAAP/hCxdodHRwOi8vbnMuYWRvYmUuY29tL3hhcC8xLjAvADw/eHBhY2tldCBiZWdpbj0n77u/JyBpZD0nVzVNME1wQ2VoaUh6cmVTek5UY3prYzlkJz8+DQo8eDp4bXBtZXRhIHhtbG5zOng9ImFkb2JlOm5zOm1ldGEvIj48cmRmOlJERiB4bWxuczpyZGY9Imh0dHA6Ly93d3cudzMub3JnLzE5OTkvMDIvMjItcmRmLXN5bnRheC1ucyMiPjxyZGY6RGVzY3JpcHRpb24gcmRmOmFib3V0PSJ1dWlkOmZhZjViZGQ1LWJhM2QtMTFkYS1hZDMxLWQzM2Q3NTE4MmYxYiIgeG1sbnM6ZGM9Imh0dHA6Ly9wdXJsLm9yZy9kYy9lbGVtZW50cy8xLjEvIi8+PHJkZjpEZXNjcmlwdGlvbiByZGY6YWJvdXQ9InV1aWQ6ZmFmNWJkZDUtYmEzZC0xMWRhLWFkMzEtZDMzZDc1MTgyZjFiIiB4bWxuczp4bXA9Imh0dHA6Ly9ucy5hZG9iZS5jb20veGFwLzEuMC8iPjx4bXA6Q3JlYXRlRGF0ZT4yMDIxLTEwLTA3VDIyOjQ5OjM0Ljg3NDwveG1wOkNyZWF0ZURhdGU+PC9yZGY6RGVzY3JpcHRpb24+PHJkZjpEZXNjcmlwdGlvbiByZGY6YWJvdXQ9InV1aWQ6ZmFmNWJkZDUtYmEzZC0xMWRhLWFkMzEtZDMzZDc1MTgyZjFiIiB4bWxuczpkYz0iaHR0cDovL3B1cmwub3JnL2RjL2VsZW1lbnRzLzEuMS8iPjxkYzpjcmVhdG9yPjxyZGY6U2VxIHhtbG5zOnJkZj0iaHR0cDovL3d3dy53My5vcmcvMTk5OS8wMi8yMi1yZGYtc3ludGF4LW5zIyI+PHJkZjpsaT5NZXJ0PC9yZGY6bGk+PC9yZGY6U2VxPg0KCQkJPC9kYzpjcmVhdG9yPjwvcmRmOkRlc2NyaXB0aW9uPjwvcmRmOlJERj48L3g6eG1wbWV0YT4NCiAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAKICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgIAogICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgCiAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAKICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgIAogICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgCiAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAKICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgIAogICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgCiAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAKICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgIAogICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgCiAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAKICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgIAogICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgCiAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAKICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgIAogICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgCiAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAKICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgIAogICAgICAgICAgICAgICAgICAgICAgICAgICAgPD94cGFja2V0IGVuZD0ndyc/Pv/bAEMABwUFBgUEBwYFBggHBwgKEQsKCQkKFQ8QDBEYFRoZGBUYFxseJyEbHSUdFxgiLiIlKCkrLCsaIC8zLyoyJyorKv/bAEMBBwgICgkKFAsLFCocGBwqKioqKioqKioqKioqKioqKioqKioqKioqKioqKioqKioqKioqKioqKioqKioqKioqKv/AABEIAQwCQgMBIgACEQEDEQH/xAAfAAABBQEBAQEBAQAAAAAAAAAAAQIDBAUGBwgJCgv/xAC1EAACAQMDAgQDBQUEBAAAAX0BAgMABBEFEiExQQYTUWEHInEUMoGRoQgjQrHBFVLR8CQzYnKCCQoWFxgZGiUmJygpKjQ1Njc4OTpDREVGR0hJSlNUVVZXWFlaY2RlZmdoaWpzdHV2d3h5eoOEhYaHiImKkpOUlZaXmJmaoqOkpaanqKmqsrO0tba3uLm6wsPExcbHyMnK0tPU1dbX2Nna4eLj5OXm5+jp6vHy8/T19vf4+fr/xAAfAQADAQEBAQEBAQEBAAAAAAAAAQIDBAUGBwgJCgv/xAC1EQACAQIEBAMEBwUEBAABAncAAQIDEQQFITEGEkFRB2FxEyIygQgUQpGhscEJIzNS8BVictEKFiQ04SXxFxgZGiYnKCkqNTY3ODk6Q0RFRkdISUpTVFVWV1hZWmNkZWZnaGlqc3R1dnd4eXqCg4SFhoeIiYqSk5SVlpeYmZqio6Slpqeoqaqys7S1tre4ubrCw8TFxsfIycrS09TV1tfY2dri4+Tl5ufo6ery8/T19vf4+fr/2gAMAwEAAhEDEQA/AO4q5Yac9+sxjOPKQuff2qnWnpeoR2MDA7t7SKTjuo6iqEQ3Gly29na3DYIuc7VHUVJLo5tr6K3uJkQSLkP2+laLavZSz/vQ/lxTK8KgdguMfnVa/wBVt7+3iPkeTLHJuGDkEUANn0IRSTRx3SSyRRmRlUdKo3di1pFAznJmTeBjpWkusRRapd3MRYGWHZGcdGxVPVNQ+3pbMzM0iR7ZCR3oAoY9qMe1FFAB+Ao/AUUUAH4Cj8BRRQAfgKPwFFFAGimlI2li8knWPdnapUnJFSyeH5knRNy7Xi8wN+GcU6HUrf8AsIWbyTxuN2QgG1s+uasr4hiE0gaNnjaIKmeqttxmgCh/ZUUdsr3V3HDI6lkjIySKgl08xG0ywP2lQwx2ycVopqOnuYrm6RzPFGU8sKCr+h9qja7sZbSyZnmWe2UDaFG0/NmgBp0Ty0nea4WNYm2A7c5NRxaNJK9qFbi4RmDY4GM/4Vct9VtY7y6uGlmxIxPk7QVce/pSwa+sBtUUssKIwkQAdTnGKAMFlwxB5waTApWOWJ9TSUAGKMUUUAGKMUUUAAFaFzpaW0AM10i3G0N5GOcHpWf05Fal9dWN4v2otMl2VAMYA25HGc0AVJLC5h8rzImAlGV460y7tZrKby7hCjVZn1m7n8jdIQYPukH9aiv7+bUp/NuTyBgDsKALh0CdZoFLjZNF5gfHA4zimxaJ5kcfmXKxzTAmOMj71X4fEMKyqHUtF5AU8fdbGMiq8WpWRNtc3HmfaLUYRFHyvzkc0AQppELac1y14ilOHQqcq3pUDaXIumLekjy2bGPQetO+2pJps8MufMluBIcDjFX5Nasnt3sxbkQ+VsSTvn6UAVZNFijtYphext5zBUUA8nNImiqrSfarlIUWQxKxH3mphvYmtdPjO7/R5Cz8e46VafUbK7Mkd35qIJzLGyDJIPY0AQNoVwltdSAhjbthlHceoqpfWbWVz5TMGO1WyPcZrTbXgN7xqQxmVgh6FAMYNU9ZvYb7UDPbKVQqoAPbAoAz8UbaKM0AXrXTUltftFzOsEW7apbuaiFjM0Ms0aF4Yjguo4q1bXNpNp4tb5pI/LYsjRjOfY1GmqTW9rNaW8jCCQ8ZPIFAED2U8Volw8ZEbnAbFTW2miaza6lmEUYOBxnJpJtVuZtOjs3fMaHP19qs6VfQWUTmaSU5yDCFBVx70AVoNLludOnvExsiPQ9SO9Pi0iSazt7hCMTSbAMdK0LfW7O2hhtltyYtjB3J5Bbrgd6SHWra3jEMasyJDhMjo/rQBXTQSJrlJrlIxAyruboSelVrrS5LO3aSVhlZTEVHqO9aLa7GkEzwrmeREHzqCMg8mq639tdaa8OoSzCUymTcig5OKAIb7SlsrVJWuFZ3APlgetZ2BW1qeo217YRpHNPujUARlRtyO/rWLQAufpRn6UlFMBc/SjP0pKKAFz9KM/SkooAXP0oz9KSigAPNaq+H7hrq2gyoM8RkB9PasyPHmDd0zziugbXYNspRW3b1ER/upxn+VIDMtdImu/tGwgfZwc+59Kls9IhurOS4e9SPyx86lT8tXV1y0tXJt4DL5kpdixxj6VR+226QahHEGxcEeXntznmgBP7Fk86JVcESQ+aGxxWcRg+tb0WvgCGHcwhW38tl2/xVgn7x+tACZooooAKKKKACiiigAooooAKKKKACrtjp5vI5ZDKsUcQGWaqVaej3UNrMzTXBiXHKbNwcelMBkWlTTxxvA6yLJIYyR/CR3NIdKlNpJcI6uiS+Xx396sWupR251DyyY0nU+UoHQ5/wqbT9VtrW1ghlyyAMZAB/F2pWAgGgzC7khklVNkQkLH0NMGjutxOtzMkUcJAaQ9CT0rRg1u1djJLK8UrRFCwTdj5sjioLjU7C+a4hmMkUUjBxIFySQMZI96AKMukywrcMzAiEBgR/ED3FUK3jqtjcLcQTNLHC0axxsFySB3NYkuwSN5ZJXPBIpgMooooAKKKKACiiigAooooAKKM0ZoAKKM0ZoAKKM0UAFFFFABRRRQAUUZozQAUUZozQAUUZozQAUUZozQAUUZozQAUUZozQAUUZozQAUUZozQAUUZozQAUUZozQAUUZozQAUUZozQAUUUUAFFFFABRRRQAUUUUAA5Nas2iPC8UaTpJNLjEYHIyM1ljGRmuiuNZtjcW1xHO7iPCmHy8YGMHmkBRk0Zg0ZinjlVn8tiv8JqCPTXdZzvUeRIEPvk4rSM1vp9ks1lL56Szh3yMFcdsVHPfWMKSC1kkk+0TLI+5ceWAc4oAqXWlSW6OQ4cxy+UwHY0sejzyaibRWUuqbiewq5Bq1qmq3ss+5oJjvUY/iHSmWer29us88yNNPM/Khtu1frQBnfYZDZzXB4EMgRlPXNWX0dktIZmuIwZQCqd+auS32m3KXkRlkhW4kWQYjzggc02bUrWTT7eJbiQeSAPL8rhsHrmgCrLok8M80crKPKj8zd2IpLvSVtbRbj7XE4kGUVerVfl12GWG9jdWJkTELkdPUVmXlxHLp9lEh+eFCHGOnNAFGiiinYAooooAKKKKQGr/wj+of881/Ol/4R6/7xr+ddfmkrxfr1Q6vZo5D/hH7/wD55r+dIfD+of8APNfzrsMUtH16oHskcb/wj+of881/Ol/4R/UP+ea/nXY0UfXqgeyicd/wj+of880/Oj/hH9Qx/q0/OuxopfXqgeyRx3/CP3//ADzT86P+Efv/APnmn512FFH16oHskcf/AMI/f/8APNPzo/4R+/8A+eafnXYUUfXqgeyRx/8Awj9//wA80/Oj/hH7/wD55p+ddhRR9eqB7JHH/wDCP3//ADzT86P+Efv/APnmn511xlQSbC67vTPNOp/XaovZROP/AOEfv/8Anmn50f8ACP3/APzzT867CmTSx28RkndY0GAWY4AzxR9dqj9lE5L/AIR+/wD+eafnR/wj9/8A880/OuwHIyDS4NH12qHskcd/wj9//wA80/Oj/hH7/wD55p+ddhRS+vVA9kjj/wDhH7//AJ5p+dH/AAj9/wD880/Ouwoo+vVA9kjj/wDhH7//AJ5p+dH/AAj9/wD880/Ouwoo+vVA9kjj/wDhH7//AJ5p+dH/AAj9/wD880/Ouwo5o+vVA9kjj/8AhH7/AP55p+dH/CP3/wDzzT867DBoo+vVA9lE4/8A4R+//wCeafnR/wAI/f8A/PNPzrsKKf16oHskcf8A8I/f/wDPNPzo/wCEfv8A/nmn512FFL69UD2SOP8A+Efv/wDnmn50f8I/f/8APNPzrsKKPr1QPZI4/wD4R+//AOeafnR/wj9//wA80/Ouwoo+vVA9kjj/APhH7/8A55p+dH/CP3//ADzT867Cjmj69UD2SOP/AOEfv/8Anmn50f8ACP3/APzzT867CjFH16oHskcf/wAI/f8A/PNPzo/4R+//AOeafnXYUuKf16qHskcd/wAI/f8A/PNPzo/4R+//AOeafnXY4qm+radHc/Z5L63SfOPLaQBs+mKPrtUPZI5r/hH7/wD55p+dH/CP3/8AzzT867HFGKPrtUPZI47/AIR+/wD+eafnR/wj9/8A880/OuxxRij67VD2SOO/4R+//wCeafnR/wAI/f8A/PNPzrp7bUbO7meK2uoZZI/vKjglfrU6SJIzqjqzIcMAfun3o+u1Q9kjkf8AhH7/AP55p+dH/CP3/wDzzT867Cij69VD2SOP/wCEfv8A/nmn50f8I/f/APPNPzrsMUuKPrtUPZI47/hH7/8A55p+dH/CP3//ADzT867Dmil9eqB7JHH/APCP3/8AzzX86T/hH9Q/55r/AN9V2OKMUfXqgeyRx48P6hjBjX86X/hH9Q/55r+ddfilp/XqoeyRx/8Awj1+f4F/Oj/hHr8f8s1/OuxpKPrtQPZo4/8A4R+/J+4o/Gj/AIR+/wA/cT867DFGKX12qP2aOP8A+Efv/wC4v50n/CPah/zzX/vquwpaaxtQPZo47/hHtQ/55r/31R/wj2of881/76rsaKPr1ToJ0kcd/wAI9qH/ADzX/vqj/hHtQ/55r/31XY0UfXqgvZI47/hHtQ/55r/31RXY0UfXqg/YoKSlorgNTk7nxHqMV5LFHLZBVcqM2dwT19RxXUxMXhRmwSygnAI/nTsD0H5UtUwCiiipGFFFFABRRRQAUUUUAFFFFAHA+W7/ABM1a5e1E8dnDE6O12yGNtrHCxjhs4os/EOuveaU76lZTw6nbSXLQLCA0AVc7VIPI56n0rrJfD+lzauupyWUbXi4xN346fzqgfCVlB4htNT0+CKAxeZ5oVfv7sVsmiDLs/GEm+xN5cx+W+mzXU4RQWDKeOPp2rHk1+91Tw7rllqxLS2slrLEzhFco7qRlVJH4+9dvB4W0W3kZ4dOgVm3ZIXk5GCPpzTrfwzo1rA8UGnwosgAfC/eAOQD9DRzRGUvF93d2Hgy5utOuPs9xGilZNoOORnrVGHUNVudeOhrqnktBp4uzdmFC0xY4AxjGB3xzyK6i7s7e+spLS6iWWCRdrxt0I9Kpt4e0t44Ua0XEKFEweQp6rn0oTQEHhHWp/EHhe01G7iEc0oIcKMAkHGR7HGa2qjggitbdILdFjjQYVVGABUlZPcYUUUUhhRRRQAU12KxsyjJAJAp1FAjibbxlrE2oCB9KRU3EFhFN/UYqXSPFmr3+tQ2l1piwwyEhpBFKMD6kYrselJV8ytsIKWiipGgooopDCiiigAooooAKKKKACuD8Z+IdT0TXIYtPvM21ygW5ym77ApIHnfTnGDXeVC9pbSGUyQRv5y7ZNy53D0NNCHwj9xHiTzRtGJM/e461zmqa7q1r4ns7O30qaW3kDgkSIPMwOvPTFdLGiwxrHGoVFGFA7ChkR3V2QFl+6SORTVhCgnYCwwSMkelcfqNnYa144i09LG2b7ABdXc3lDcXP3F3Yz712Gc1GsEMczyxxKskmN7Act9aEwJCcKccnFcTF4y1qTUVgbSUVDIVLeVNwPXOMV21GaFJdRnGWfi/V59WjtZdMVImk2l/KlGB65IxXZsQAc9ADnPpRQcEEHkHrRdXEcNop07xB4ok1LTJLa2t4Int4Ut3VJJyT8zkDoPSrOj+GdWtLu6kmu5IjNC+yRJi5Eh4VnB4cgY5rpLfSNOs5vNtLKGCTGN0aAGrUgZomVHKMRgMB0PrVc2oHH+E/EGo+Ib9YZ90J0yIw6gpXh7nOCB7ADP/AAKuyrP0bSIdGtZIomaWSaRpppn+9I7dSa0KmW4woooqRhRRRQAjZ2nDbT2PpXH+GvFOoax4kuNMuPJRNPDq8yrxeHdgNH6BcYPua7AgMpDdDVaHS7K3MDQ28atbhhEQvKhuv51SsIzLu615PF1pBa20L6U0RM0hY5B/Kq/jK51eytIZdG1EQTzSrBDA0CuJHZgBknoAMk49K6XiqN7pcd9qFhdSOwNjK0qJ2ZipXJ+maegiDW5dUtdFLaaPOvFwPljDbvU4JA/WnQXN4PDfn30iWl15ZLySIAsZ9SMkfrWnUN3aW99ZyWt3EssEqlXRujCi4HG23iTVbPQv7Wvp1utOhvMPP5IR5bcj/WbRwMN+YrpfD13e6ho6XmobVM7F4kAxtjJ+UH3xVS68LWo0U6XpcUdpbTTI1wijO9ByVH1wK3ERYo1RAAqjAA7U5NW0AdRRRWZQUUUUAFFFFABRRRQAu0+lJg+h/Kr+KMD0rt+q+Zj7QoYPofypQD6Gr232FGB6UfVfMXtCjtPoaNp9DV7AowPSn9V8w9oUdp9DRtPoavYHpRgelH1XzD2hR2n0NG0+hq9gUYHpR9V8x+0KO0+ho2n0NXsD0owPSj6r5i9oUdp9DRtPoavYFGBR9V8x+0KO0+ho2n0NXsD0owPSj6r5i9oUdp9DRtPoavYFGBR9V8x+0KO0+ho2n0NXsD0ox7UfVfMXtCjtPoaNvtV7AowKPqvmHtCiVPoaTB9D+VX8UYpfVfMftChg+h/KjB9D+VXqMe1H1XzD2hRwfQ/lRg+hq9j2pcD0o+q+YvaFDB9DRg+h/Kr+BRij6r5h7QoYPofyowfQ1fxRgelH1XzD2hQwfQ0YPofyq/gelJj2o+q+Ye0KOD6H8qMH0NXse1LgelH1XzD2hQwfQ0YPofyq/gUY9qPq3mHtChg+h/KjB9DV7HtS49qPq3mHtChg+howfQ/lV/AoxR9V8w9oUMH0P5UYPofyq/ijHtR9V8x+0KGD6GjB9DV/A9KMe1H1XzF7QoYPofyowfQ/lV7HtS49qPqvmP2hQwfQ0YPoav4FGB6UfVfMXtChg+h/KjB9D+VX8e1Jj2o+q+Y/aFHB9DRg+hq/gelGBR9W8xe0KGD6H8qMH0P5Vex7UY9qPqvmP2hRwfQ0YPoav4HpRgUfVvMXtChg+h/KjB9D+VX8e1GBR9V8x+0KGD6GjB9DV/Aox7UfVfMXtChg+h/KjB9D+VX8e1GPaj6r5h7QoYPofyowfQ/lV/HtRj2o+q+Y/aFDB9D+VGD6Gr+PajAo+q+YvaFDB9DRg+h/Kr+B6UmPaj6t5h7Qo4PofyowfQ1ex7UuB6UfVfMPaFDB9DRV/Aoo+reYe0CiilrtMjltX8aR2N5DZW9hezXUku0L9nfGwH5mHHOP6100b+ZGrgEbhnBGDXP/APCEaTLrV1qN0s00k2AimdwIh3Aw3c8n8K6FEWOMImQqjAyc02A6iiipAKKKKACiiigAooooAKKKKAOFv/G15pvjLU9Nuo0WyS2H2afH3ZtjsFb67ePpVi28csunW0ktjNctHaRT30sWAsO8A9CeeueKu6r4LstYj1qK9ZnTVo41bHBhZAdrKexBOfwqq3gQrGkFrqs8FvJbRW92iop+0BBgHJHynAAOKvQCzoniG91PxVqdjJZMllbpG8E5I+YMM/rVzxH4jj0CKAeQ1xPcFhFEGC5CjLEk8DAqax0SKw1ie9t5GCTQJEYT0GzofXpVTxT4Xj8SR2biVYbmzlLxO8KyryMEFWBB7H8KNAM9vH0L2qT2emXdyq24uLkKADCp9QTyfpVn/hNbb7Dc3LWsyC3sUvirAZKMSAPrxVTW/CV8+nzy6fq80VxJa+TcbIkJuMdMcfKe3FRXPgh9X0y33alcWK3Gmx2l5CiKWdVGQMkfKck59aegGrqPiy30zWrOwuY+LsqqyCRcqx6ArnP44p+g3lxLDqzSM07QXkixr0OABhayrv4fm71P7UdXlXMkUzgQIWZ0AAO4jIBA6Diuk0vS00xLkLI0huZ2mct2JAGP0oA5bR/G99c6JaSXOkTSahdzTrHbRsvKRtgtnOABwPrXV6TqcOsabHeW4ZUkHKt1U9wfesNvBjwwwf2bqs9pc200zwzBFbakpy0ZBGCM4x9K2tF0qLRdJhsYWZxGCS7nl2PUmloBepaKKgBKKWigDnrXULn/AITPV7eaRjZwW8bouM7SRziprLxZpd9ex2sDzmWQ4UNA4H5kYq6umj+0bm5kmeSO4jCGBgNq49OM8/Wqlp4R0GxvEu7PS7eGeM5WRQciqA2aKKKQBRRRSAKKKKACiiigAooooAK5XXvGE2mXN9Fp2nm9/s6Dz7tzIECKc4Az1PBrqa5jWfAtprF9eTtf3ltHfwCG7ggcBZgOhOQSCM9qpAUb7x9PbzXP2bSHmgs4YJ7iXzAoVZVB4z1Iz0q1deM2j8Sx6TDaxr5kaSB7iYR7w3ZM9SKtSeDbGS21CAyzbb6GGGTkZURgAY49qj1TwcmrvHFeahcmxjKH7KFTGV6YbbuHTsarQDpvrXP3N7dp49srRJCLN7KR3THVgeDW/gAADtVSSxL6tHeefIFSMxmHja2e/TOfxqQM4eLtK+2C2DXBlL+Xj7O+M59cYrdHNYo8HeHheC7Gk24uA/meZjnd61tUgMaLTry38S3uoyXbNaSQKqW5PCsOpFY2k+O5tSvtOWbSXt7TUZJYYJi4JLx5zx6EA812DqHRlPRhg1gWvg+ytI9JSOWY/wBlzSzQ7j94vuzn/vo0wLWm6ZeWuvaneXF28tvdMvkwsciLA5xWtmgcUtIAooopAFFFFABRRRQAVz7+LLePxWmiGEs8kbMkiSK2SoyQRnI+prfI3KR0yMVxmmeADp+qJeHVJJTCJ0iHlIpxKckswGWIPcntVoCW1+INvcSRM+m3kNrKshS4dQASnUY6/jW34f1ltd0/7Z9ka3iY5iJdW3r68H9Kzf8AhETDBZrZXrwyWds0MUjIGwT1YgjBNX/DmhJoGnSQCUyyTStNK+0KCx9FHAHHQUAU/EHi+PRS0CWN3PdMwSJVgYq7H0IHpW5ZXJurKKdo2jZ1BKOpUg/Q1kXvhHT9R13+0b3zZMJhYhM6qrd2GD1rZt7eO1gWGEEIgwoLFj+Z5pAc1davP4ZuktJ4rrVLvU5ppoEiA2oFx8mT0ABHJq5Y3Uviax0zVLWSWyhDmSSFvvP22n2qxfaIL7VkvWmZTHay26KP4S5GWHvwKtaVp8elaXb2MJJSBAgJ6n3oAt0tFFSAUUUUAFFFFABRRRQAlLVPz3o8965/rEDT2bLlFU/Pf1o89/Wj6zAORlyiqfnv60ee/rR9ZgHIy5RVPz39aPPf1o+swDkZcoqn57+tHnv60fWYByMuUVT89/Wjz39aPrMA5GXKKp+e/rR57+tH1mAcjLlFU/Pf1o89/Wj6zAORlyiqfnv60ee/rR9ZgHIy5RVPz39aPPf1o+swDkZcoqn57+tHnv6j8qPrMA9my5RVPz39R+VHnv6j8qPrMA9my5RVPz39R+VHnv6j8qPrMA9my5RVPz39R+VHnv6j8qPrMA9my5RVPz39R+VHnv6j8qPrMA9my5RVPz39R+VHnv6j8qPrMA9my5RVPz39R+VHnv6j8qPrMA9my5RVPz39R+VHnv6j8qPrMA9my5RVPz39R+VHnv6j8qPrMA9my5RVPz39R+VHnv6j8qPrMA9my5RVPz39R+VHnv6j8qPrMA9my5RVPz39R+VHnv6j8qPrMA9my5RVPz39R+VHnv6j8qPrMA9my5RVPz39R+VHnv6j8qPrMA9my5RVPz39R+VHnv6ij6zAPZsuUVT89/UflR57+o/Kj6zAPZsuUVT89/UflR57/wCRR9ZgHs2W6Kqee/rR57+tL6zAPZst0VU896PPej6zAPZst0VU896PPej6zAPZst0VU896PPej6zAPZst0VU896PPej6zAPZst0VU896PPej6zAPZst0VU896PPej6zAPZst0VU896PPej6zAPZst0VU896PPej6zAPZst0VU896KPrMA9myOiijtXAdBVvtStNOj33twkXoCeW+g6mrCMHjV16MMjiubPh+803VJtSsfL1IyuXMV0fnT2RugHtxXSIzPGrMuwkcqe1N2EOoooqRhRRRQAUUUUAFFFFABRRRQBSttVgudUvbBAwlswhkJ6fNnH8qjTxBpMkU8iahAyW5xIQ33awpLTVYfE+teVYO1pqcCRJdrIoERCsMkZz1YVg6b4UvYtDmj1HSbma7SOOEOk0a7thJyoz6889c1qoxIud+Na00tbqLyItc8wjPL1FqWvWWlajY2V3JsmvnKRD6DNckuj66b3Srn7IUvo1CzXHyGLy92drL2bHcd66PXNKkvNW0e8it0m+yXLGTdjIQqRn8DiiyuFxdC8UWWrxBGnhS7ywMAbJABq9FrOnTQyTR3kRjifZI27AU++a5S18L39pbafJaWsMN1DNPI8nH8QO3J71k2nhjXLiz16O9sJMajp4hBmlUlpwTyAvQHPH0o5UB6HZ6nY6j5n2C6iuPLba/ltnaat1gaNozadr1zNHBHDbSWkMa7AAC6g54rfqGrFBRRRUjCiiigAooooArzX1rbzRxTzokkhwiE8sfpU9c1a+GrnStea+t5Uv0nky5u+ZIQf7jdMe3FdMabQgooopAFFFFAwooooAKKKKACiiigArN1LXtN0mZIb65Ecsg3KgUs2PXA7VpVzF9puqWHiybWdOs4tQjuLYQNE8gVoiDnIz2OeaaSZLNK78S6RZXKQXN6iSuqttwTgHoT6Z96nl1rTYFuGlu41W2K+aSfubulcpqfhvWnvNb+x2lvNHriR7pJJADakKFI9wMZGKrar4S1111SzsoreWC+WE+e8mChQAEY9605YiOrn8U6Nb35s5b+NZ1kETJz8rEAgE9s5FV/+Ez0n/hJZ9GaRlnhRWLbSQSe3SsifwleTwa6FWHzL6/triJm67ECZBP8AwE1fn0vULPxm2q2djFdwXFskMg3qrRlT1560WikB1FV7q+tbNo1ubhInkOEVjyx9hU/auZHhm4svEbarbypfLNJl47sZaEf9Mz2HtUIo6SR1jiZ3O1VGST2FY9r4w0G9uY7e21KKSWVtqLz8xxn+Val5CbixnhXrJGVH4iuB0/wNf2lzYOY4F+zyWhZlPIEaSBv1cVcVFrUR6LioYriKaWWOJwzQttcD+E4zS3EXn27xbmTeuNyHBH0rnfD3hZ9I1i+vZL27lEs26NJJdwYbQMsMdajQZ0k29YHaMBnAO0E4BNYeh69d6hrF3p17bQpJbxq/mW8u9Dk/dPoa2pyy28hjj8xwpITONx9K5jw1pN7D4ivtTlsE0u3niCfZlfcXcHO84/KnG1hHVmiiioKCiiigAqm2rWC6iti13ELojPlFuatnocda4eLSNXbR9VsG0oLeyNPImoO6nzCzfKFPUfKce1Wop7kM6uHWtMnaYQ3sLmD/AFuH+59alk1GzhL+bcxIE27izYC7umfrXCXPh7VLq1lu4NIWzZYI7dbRXUPKobLEnOPpmrT6Bqmq2epRajp6QjUL23JUyBtkCEEjjvwfzq+WIXOgvfFmkWWmz3zXkcsMDBH8s5+YnpWml3C9ml0zqkLoHDMcAA1yF14XuHGoGGyi2XGpxSrCMAeQihf6Zrb1zw+2qSW01vOI2tx8tu4zC/8AvAUnGI7s0rPULXUIXltJfMjQkF8EDj+dVZPEmjROUfUrZSq7iDIOlT6U1ybPZf2cdq6HbsiYFGHqPb2rj7bwW8UliZNPhby7y4lkJAPyt93/APVSsgudxFcQz2y3EUivCy7lcHgj1ptvd293arc20qyQsMh1PGKy9D0p4/CcWm6jHt+V1dAcYBYkdPY1Poeh2fhzRU0+1yYYwclznPrUuwFmy1Ox1FpFsbqOcxnDhDnFWq4zw26N421B7W4hvIHgA8yCPasOG+4ccE+9dnRJWGgoooqRhRRRQAUVN9nPrS/Zj/erb2M+xHOiCipvsp/vUfZj60exqdg54kNFTfZj60fZj/eo9jU7BzxIaKm+zH+9R9mPrR7Gp2DniQ0VN9mPrR9mPrR7Gp2DniQ0VN9mP96j7MfWj2NTsHPEhoqb7MfWj7MfWj2NTsHPEhpOvWp/sx9aPsx9aPY1Owc8SGip/sx9RR9mPqKPY1Owc8SCipvsx9aPsx9aPY1Owc8SHNFTfZj60fZj60exqdg50Q0VN9mPrR9mPrR7Gp2DniQ0VN9mPrR9mPrR7Gp2DniQ0VN9mPrR9mPrR7Gp2DnRDRU32Y+tH2Y+tHsanYOeJDRU32Y+tH2Y+tHsanYOeJDRU32Y+tH2Y+tHsanYOdENFTfZj60fZj/eo9jU7BzxIaKm+zH1o+zH1o9jU7Bzohoqb7MfWj7MfWj2NTsHOiGipvsx9aPsx9aPY1Owc6IaKm+zH1pfsx9aPY1Owc8SvS1N9lP96l+yn+9R7Gp2FzxIKM1P9lP96j7Kf71HsanYfPEgzRU/2U/3qPsp/vUexqdg54kFFT/ZT/eo+yn+9R7Gp2DniQUZqf7Kf71H2U/3qPY1Owc8SCip/sp/vUfZT/eo9jU7BzxIKKn+yn+9R9lP96j2NTsHPEgoqb7Kf71H2U/3hR7Gp2FzxIe+e9Gan+zH1o+zH1o9jU7D54kFGan+zH1pPsp/vUexqdg54kNGan+yn+9R9lP96j2NTsLniQUVP9lP96j7MfWj2NTsPniVkRYwRGiqCcnaMU6p/sx9aPsx9aPY1Owc8SCip/sx9aPsx9aPY1Owc8SCip/sx9aKPY1Owc8SxRRS16hzEF1d29jCZryeOCIdXkYKP1qVHWSNXQ7lYZBHcVwV40Gma9Pe3Vzb6viQlYp3Iktx/dTt+ld5DIJYEdRgMoIB7UAOopaKQCUUtFACUUtFACUUtFACUUtFAEaTxSTSRJKjSR43oDkrnpkdqkrh01WPT/GPiaESKt5LBE1rEwP7xwj8D8cVzOk3ep3fhy7uX15raR44zLFLO7NvJ+bLY+TPTA6Yq7AeujmmSTxRyJHJIqvISEUnliBnj1rzVNZdb7Rbrz7prdkWP7F57eYGL43/AO2v17V0PiywW48T+GbuSW4jSG7dD5bkKCUOMgeuMUWA6Sxv7fUbQXFo5eMkqDgjkHBqyORXl+jJdaKthfLc3swmnuRJb7zs2jJAC9vrVax1/Up9O8Qra3sqMLEX1rskaQxSAnK5b/gOV6U7AesUZ4Ncr4YiurPXLyymvbm7i+zQz5uGLEO2d2D6cdKwU1C5bxAjjUbptTbUvIkssnYsBPJx0xjJzSsB6QOVzS1y3gOO7bSbm61C9uLuaS6ljHnn7iRyMigD6Dr3rqalgJS0UUgCiikOe3WgBaSuMM2uHXm/twzxJvP2NLIfuW9N5659jxS6PceL21qFdUVxZ7j5mUjHGOOgzVWA7KloopAFFFFABSUtFIAooooAKKKKACoZLu3iuY7eSZFmlBKITy2OuBU1ef8AjPR9Pbxrouo6l9oS1xLHNLHI4VTt+X7vTmmgO0utShtLy1t5UlL3TMqFIyyjAz8xHT8at9ce9cTrxg0/xP4a1OGSdYJGeKVwWZSu0bcj+tcrrNpqNxq+pXCXGoAGW48sJIwXaq5TA+tVYD2E0leQl9Zu9fb+0L77FPG9s1pJIJMmPYu4AA4OTuBzXrq/cGTnjrS6gOpK467n1r/hJAmt+dDpnmAW/wBg5V+ePMPUfyqK3ufGZ1VFmST7L5vzHZH93P0zQB2x4HSsvS/EFpq1/eWUCTxXFkVEsc0RQjdnBGeoODWpziuO0y4ig+J+u29xuRrqC38rKnD4DZ5/EUAdPa6jDdX1zaoswktiA5eJlU5GflJ4P4Vb7151Ex0y+8WaXBJcoXQSWiks38AyVP1rL8NadfW/iiwEs1+8MkpjcSyMV2tbEtnP+1inYD0601GK8lukiSVTbSmJzJGVBI7qT1HuKmtrqC8h821lWVMkbkORkdRXHeFI4EuPEOhSvPhb2QxRuzZERAxtY/41W+Gtjp+n20luDcJqKSSCSKV3IC7uODxQ0B39FFFSAUUUUAJUE19awCUzTxp5KB5MtyinoSPwqxXmklvJFqmu3trLci9vtUhsFDMSEjAViQOmMbqaA9JRhJGrodysMgjvTu9eZ3EeqX+oFo9Zv4Uk1MwQBGIVIl+/x3/Gs+51TVYoLTTZLy4FncPcuLiadkYhWCou8cnnJx3xTsB65UF3e21jD5t7cRwR9N0jAAmodJgnt9FtYbu5N1MkQDzsMFzjrXHO0Gla3Ld3Fxb6yWk485/3sPsoPFIDvVYOoZTkMMg1RGs2BuJ4fPG+CVYnBB4cjgVdEgaIP0BUNz2rzrTp7o3UEwaSMXVzc3s5AIyi5CCnYD0eoZby3huFhllVJHUsqk8kDqa8vWDVJY47i61rUv8AkGz3VwFkK8Z/dqMdD7jmrtw+o3Gi38d1c3LNaaRDE7KSC0r8s2R3waLAejRypNGJIXV0bkMpyDT6zPD9imm+H7O1hd3RIlwXYk9PU1pVLAWiiikAUUUUAJS0lLTArvYWjuXe1hZickmMEmpwABgDApaKQBRRRQAUUUUAFFFFABRRRQAUUUUAQm2gacStDGZR0coNw/Gk+xW21x9nhw5y48sfN9fWuD1LVtYn8fvZRXzWUcE8IjVpVWOSM8t8uMuT06jFUrTW7+fUtJYa/Mt5e3skd3p+FxEqk4UDGV6de9aAelfZbfzEbyYg0Ywh2DK/T0pt5cW9paPcXZVYYRvZmGduO9cJp/i1jfaVb3GotJKZrkXEQ+9hVbap9DnGPWsm08RXOpWmv2k1wZLe40wXcEck/mtGWZgVLADBxt+XtSA9S32ot1mPlrHjcrEAAZojtLZFIjgiUMOcIBnNcl42sZL34dIEnngMaxOTCcHAxnNZOpa1eWl3eTRazOl1YS28dlY5BW7RtuSwxlicnkdMUAekKiq2QoBxjOO1N+zwifzhFGJTwX2jd+dPXlRkYOKWlqA1UVBhFCjOcAYp1FFIAooopAFFFFACUtFFMAooopAFFFFABRRRQAUUUUAFFFFABTXRZFw6hh6EZp1FADSikAFQQOgI6UbFz90flTqKdwGtHGxBZFJHQkdKWlooAKSloouAZpvlpv3lV3euOadRQA0xoW3FFJ9SKAig52j24p1FFwG7FDFgq7j1OOaAiBiwRQx6tjk06ii4BRRRSAKKKKAEpphj3Z2LkndnaOvrT6KYEYhjGMRrwcj5e9Ne1t5AokgjYIcqGQHafapqKLgJjjHaq7afZs+9rWEtnOfLGas0UXATHGO1M8mPGPLXGMfd7VJRRcCMQxYI8tMFdp+Xt6UvlJz8i/N97jrT6KLgIAAMAYHtS0UUgCiiigAooooAKKb5i+oo3r/eFLmXcdmOopnmL/eH50vmL/eFHMu4WY6imeYvqKPMX1FLmXcLMfRTPMX1FHmL6ijmXcLMfRTPMX1FHmL6ijmXcLMfRTPMX1FHmL6ijmXcLMfRTPMX1FHmL6ijmXcLMiksrWW6S5kt42mj+5IVBK/jVC+8P2t5qVnfIiQzW0/ml0QZk4xgn8a1PMX+8KPMX+8KrnXcLMhXTrNZmlW1hEjNuLBBkn1pItMsYVcQ2kKB8ltqAZz1qxvX+8PzpN6/3h+dLnXcLCtGjR+WygoRjaRxioG0+0e5juGtojLGMI5QZUexqbzF/vD86PMX+8Pzo513Cw6jNN8xf7wo8xP7wo5o9wsx1LTPMX+8KPMX+8Pzo5o9wsx1Gab5i/3h+dHmJ/eFHNHuFmOpaZ5if3hR5i/3h+dHNHuFmPopnmL/AHh+dHmL/eFHNHuFmPopnmL/AHhR5i/3hRzR7hZj6KZ5i/3hR5i/3hRzR7hZj6KZ5i/3hR5i/wB4Uc0e4WY+imeYv94UeYv94Uc0e4WY+imeYv8AeFHmL/eFHNHuFmPopnmL/eFHmL/eFHNHuFmPpKb5i/3hQZF9RRzR7hZjs0ZqvDf2s800UM8byQMFlRWyUJGQCO3BzUwkX1FHMgsx1FJvX1FG9R3FHMu4WY6imean94fnR5if3h+dHMu4D6Kb5i/3hR5i/wB4Ucy7hYdRTfMX+8KPMX+8KOZdwsOopvmL/eFHmL/eFHMu4WHUU3zF/vCjzF/vCjmXcLDqKb5i/wB4UeYv94Ucy7hYdRTfMX+8KPMX+8KOZdwsOopvmL/eFHmL/eFHMu4WHUU3zF/vCjzF/vCjmXcLDqKb5i/3hR5i/wB4Ucy7hYdRTfMX+8KPMX+8KOZdwsOopvmL/eFHmJ/eH50cy7hZjqKb5ieo/OijmXcLMo0UUdq8fU6haSsC/wBeudBZpdZgVrMthbiBslc9mXr+Wa3UdZYlkjOVcBgcdjT1AdRRRSGFFFFABRRRQAUUUUAFFFFABRk1yN7Pe6jrOrf8TaXTLPSUU/ulB3EgsWbPbA6U658Yy2l7eJ9jElnZ2q3LXRfG9WA24HqTVqLJudZRXG2njiefRbu7mitYmjdFgcykxSlv4QwHJHtUT+K9R1RPDV7pUSJa308sdysjYIKBgQPb5SarkYXO3o5riLLx7PfPdvaWCzwQxPIpjlyVC9n/ALpPatqHVNRv/BkuppaLbXMlu0sETyZwNuQSR374qXFoLm4Tg89aBzXm9trviHdpN8LeK5nk0c3E6tNtQAEnP1IrZufHMdveaWoSJkvhH5kYcmSIucDjpj8afIwudfR+NcjP4v1GHUrmNtLQWlneraTymXkl8bSo7jDDNdfUuLQ0wxSUtFSMSj8aWkYMVOzG7HGfWgApc1z9n4nH9tLo2q25t75+U8s+Yjj1yOn44roKewgzRmiikMM0ZoooAM0ZoooAM0ZoooAM0ZoooAM0c0Vx/iC91+Hxvo9ppctsLe4SU7JSfmKrnnA/KmrsR2GQCAWAJ6ZPWg1y+rXF1D4v0CK6igeOZnCsrsGRwoJ46EVi6p461u11O9htLK1MFvJKiu8hyfLGSSMelWoMk0N1jaaxrl6l7Fa6pDdBgrHieIRQjYw7jLDnsTn1rp9Lv11TTo7tI5Id/WOQYKmvMNR1aDWtXufs+lFh9pt1ujCjmWXfGobDAYChXwRnsa9YQbYlVc4VQBmt6sk4pW1JW46isD/hJ1tNch0rVrcwT3DEW7xHzFf645X8a6CubU0GJIkmdjK2OuD0pQQWKgjI6jPSuU0a3S0+I2tRQbkja1ifZuJG49TSWOpTweIPExuoYftFlbRyh43JDrtYgEHp92qsxHW0gIZiAQSOoB6VwGmeOtZutWtbW5srVI55UQsrkkb03L/PmtnwzPdzX2vxSRQreQ3KqWV2KOduQeen4UcrA6c8UVx/gK/17ULKeXV5LeWJbqeMMpO8FZCMdOnFdhUNNDCiiikFgooooCwUAZorh59f1QX+vvcKv2CGaGyt1ik+cyMRkjj0YflVxTYjuDxRXFXfjHVILl4LPSBOsN2LPc0wBkY9x9O9Ml+IDw2aRy20EWoPLKgR5f3eI8ZOcc5JwAO9NwYHcUVV0+5nvdHguJoGtbiaIMYpP+WbEdDWSfEM2mXkNnrsASSd9kUtud6se2R1H41NmGh0H14oPFYHiiZvP0iyjkMbT3odyDj5I1LN+HSsePxZeww24s7Brhr2OW6VpZgFjVSevtjpVKLYjt+tGK4iPxvenT7mbUNN+yqNPN1GElBbk7QD6ZJ4q9pd3q8njB7EhTp1pZR+YWk3OZG5yeOuKHFoZ1NFFFQOwUUUUhhmiiigAoFFFPUWhnQ6DYRX73rxtcXLNkS3DFynsueFH0rRNFFGoBRRijFIAooxRigAooxRigAooxRigAooxRigDD1fwpZ6tdSXDXF1atPH5VwLeXaJ0/usPxPPXmpn8N6dItyrRtsubdLd13cbF6YrWoqrsLI5z/hDbY2KW7ahqDPFIJIJzP8APCQMALxgDHtViz8JabZWFraRNcFLW5NzEzylmVzndyexycj3NbdGKfMwsjCtPCdnZfao4Lm6FtcBh9m8z92m7rgVrW9pFbadHZIMwxxCIBu6gY/lU9FJybCyMa28LWFrFHFG0xWO0a0Tc+SIznj9ao3PgLTLnaDcXsUe2MNFFPtVyn3WOO4rp6MU+ZhZGTN4csp/tXmGTN1cx3Mh39XQKB/6CK1qKKltsLBRRRikMKR1DoynOGBHBxS4oxTAp6fpNjpaMtjbJEXOXYDLOfUseT+NXKMUUmIKKKKBhRRRQAUUUUAFFFFABRRRQAVm6toNnrLW8lyZo5rdiYpYJmjdcjBGQehrSoprQRl6j4dsNUSzF2Z99k26GVJ2V1PfkHnNQy+EtImeV3gZmlLl8yHneMN+lbVFPmYaGL/wiWmLe/aYGurZ22CRbe4eNZNowu4A4PAxmtnAC4HFLRRdsLIpWOkWOmySS2luqzTMTJMeXb6seau0UUtQM1tBsW13+18TLebAhZJmVWUdioODUU/hjTrjV7jUnEyz3MXkzBJmCyLgjBXODwTWvRTuwMiHwtpEFxFPHbESQurod54KrtH6VNbaFZWmtXGqW4mW4ujmUec2xjjGdvTNaNFHMwsZVh4dstMv5rqya4iEzs7Q+cxi3MckhOgJPNatFFIAooopDCiiigAFZLeG7Bpnk/eYe8W9ZN/ymQLgHHpWtRVJtCMmLw3YxPG48wslw9yCX6u3Umqj+CtPZ7Z4Li6tpIBIvmQyYZ1c5YE49e45roaKfMwGtGGjKbmAIxnPP1zVLT9EsNNdpLeHdOxy08rF5D/wI81fopXYGff6Na6hexXU+/zYYZIU2tgASABj9eOtVl8LaekSxoZVVbT7IuH+6mc8e9bNFHMwsYs3hXTpxIJBIRJHDEw38bY23KPz61fg023t9Tur6PcJroKJBu+X5RgcVboouwsgoooqRhRRRQAUUUUAXPKQdqPLT0qSivX5Y9jkuyPy1/u0oRfQViXniKTSJ3Ot2pgtN2Eu423JjtuHUGtuN1ljWSM7lYZB9RT5Y9guxfLX0o8tfSnUUcsewXY3y19KPLX0p1FHLHsF2N8tfSjy19KdRRyx7BdjfLX0o8tfSnUUcsewXY3y19KTy19KfSUcsewXY3y09KPLT0rAvvEF+dWnsdF05bz7Iqtcu8uwDPRR745q2/ifSob17Oe5WO5iTzJYiD+7GM5PtT9nHsF2anlr6CjYvpWRb+LNJurCe7hmdktyPMURncM9Djrg1XvPGWnwXWjJBunj1aQpHIinC4B6/iMUci7Bdm/5a/3RR5a/3RWMvjDRDczw/bADApZmKnaQOuD3xTrnxPYp4YudbtS1xbQIzAqp+fA7e3vS9nHsF2a/lr/do2L/AHa48eP7OHUoPtzmGzuLGO5T92SwLMQc+gGBXQSeIdOj1G3snmKy3KhoSVO18jIwfWnyLsF2aHlr/dFHlr/dFY0XjHRJtSWwivA07OYsbTgOP4Sex4PFblLkXYLsZ5a/3RR5a+gp9FLlj2C7GeWvoKPLX0FPprtsjZiCQozxT5Y9guxPLX0FHlp6CqVhrmnakrm2uFLJ99H+Vl+oNOh1rTLi4WGC/t5JWOAiyAk0ckewXZb2J6Ck8tP7oqQiilyR7BdkflJ/dFHlJ/dFSUUcsewXZH5Sf3RR5Sf3RUlFHLHsF2R+Un90UeUn90VJRRyx7BdkflJ/dFHlp/dFSUlHLHsF2M8tP7oo8tP7op9YWteL9O0SSWOdZpngi86YQpu8pPVqfJHsF2bXlJ/dFHlL/dFYN/420ux6ieYLEk0pijyIlfG0t6ZzS3vjbSLC78iaR22hDK6rlYt33dxp+zXYLs3fLT+6KXy0/uinA5AI5Bqha65YXl1JbRzhZ42KtFJ8rfgD1pckewXZd8tP7oo8tP7oqp/bemCbyvt9v5m7bt8wZz6VdJp8kewXY3yk/uijyk/uiuak1bWLLxhaadM9vd295vJSNMNAoGQxPp2qTSNa1W48X6jpWpxW6R29vHNEYc5IZmHJP+7T9nHsF2dD5Sf3RR5Sf3RT6Kjlj2C7GeUn90UeUn90U+ijlj2C7GeUn90UeUn90U+ijlj2C7GeUn90UeUn90U+ijlj2C7GeUn90UeUn90U+s19dsU1oaS8pS7ZC6qykBgOuD0p8kewXZf8pP7oo8pP7orDj8aaHJqC2a3mZmJVPkOH9cHvVnTvE2l6rqUljZTl541L42kBlBwSD3GaORdguzT8tP7oo8pP7oqvqEt5Dbb9PgS4cHmNm25HsfWqul+ILXU7qW02vBeQjMlvIOVHrmjkj2C7NLyk/uijyk/uis648Rada6xHpk8rLcSMFXKHbuIyBu6ZwKt2V/b6jb+fZyCSPcV3D1BwRT5F2C7JvKT+6KPKT+6KfRU8sewXYzyk/uijyk/uin0UcsewXYzyk/uik8tP7oqSko5Y9guxnlp/dFFPoo5Y9guxaSloqhHITaNqb6pLcXGnxagDITF9ouvlRe2Exj866yMERqGUKcchegp9FMAooopAFFFFABRRRQAUUUUAFFFFAHI6toWtR6hqkmhm3eLVoRHL5rlWgcArvGBzwentTLnwXPcR6khnQNdafFbJN/FuQdTW7eeI9JsNQSxu7xI7hyAFIOBk4GT0GfepLXXNOvb+WytbpJbiHPmRrnKYOOfSrA45fCGsLoawpFBHMJo2mQXLZulUYwz44+gq3pHhHUbDRtItppLcy6beyS4QnaYn3ZAJ5yN36V0Q8RaSdVbThex/aVJBQ5xkDJGemcdqhtvF2hXUvlQahGz4Zgu0jcqjJYccj3pgc3pHgm9sY763mWCVHikjt7l5SzYY8Dbjj3rp4NGK+Dl0aRlVvsX2ZmUcZ24zVtNWspGCxzKzGHzgAM5T1rLv/E9v/Yf27SpFmIuobdwykFN8iqcg8g4bNIDPk8J3s9vL9oaDzZNKSyyvQOCTn6c1mXPgrWpNYsp1mt5EtJYJY5JJW+QIBlAoGOfWup17xPaeH5LBLxXY3s4gTaCdue54qZPEeky6sdOiu0a6B27MHGf7uemfamBit4Tuf3xQwqZNaF/x/cxjH1rrqSipbAWiiipAKa2djbcBscZ9adRQBzdv4RifUW1PVLlrm/ZSqui7EjB9AOv41FpvgTT9L1OK+hubhpI2yFbZg/kK6mimAdqKKKQBRRRQAUUUUAFFFFABRRRQAhrjNe8G6lqN/qr6dfwwW+rWwguBIhLLgEfKffNdpRTTsB5Z4h0HUbH7dpmmyTzf2lFbiRVtXbLoqrw+MBcLk5rYm8AudYkvols50ukj8+O63kqyjGVxwfxru6SnzAIg2oBjoMcVz0nhRdQ1ZNQ1q4NzJC2YI4hsWP0yepro6KVwOW/4QHTjqX2w3Nxv8zzNvyYznP8AdrqDwOKWii4HK6HoOuadrVzdX13ZXMdxIWL7G8wL2UdsCktNC16HxlcazJdWJiuI0geIK24RqWIx7/NXV0lO4C0UUVIBRRRQAUUUUAFFFFACNnaccHHBrzf/AIRDWItSk1TWLqBFhhukkuTMzMRIMK4XGFwOMV6TSHkEHoaaYHl+hWV54huhAhs/s2nWDW8NzatuUyMMZzjrjt2rp/BXhq40K3/4mEMInSMRLKspkYqOvUDA9q6hI0iXbGiovoowKdTbAy/EFre3mm+VprFZCw3bZNhK9xnBqv4fsLixZkm0y3tVI/1iTGR3PuSM1uUUrgcpe6VrV540hu54rabSbchoI/NKsr45dhjk+lanhrSptG0lrW4ZWczySZXphmzWvRTuAUUUVIBRRRQAUUUUAFFFFABRRRQAUUUUAFFFFABRRRQAUUUUAFFFFABRRRQBw+t+G9VudQ1WC1gt5bXVmiLXDt89vt64Hfjp71uaHop0zUtXuWSMfbLkSKyjkrtA5/EGtyiquM8/tfBd8mt3wuo1ltppJZIbkzf6reuOF9feqkOk6mPEej6bqFnbW/k6Rc2kUsRyZMBF3H07HHua9LpnlRmUSFFLqCFYjkA9efwp8wjz+38PeKVbzALa1aKwNnGUkyxIOQ2fek0zwhrEVnfxSQxQi7u7W4AM28r5bqWyfcCvRKKXMBi+ItKm1KGxa1VGktbtJsP0IGQf51g2vhjVI9QgtJIoFsYNQN6LsN+8fnO3H6fSu4op8wBRRRUAFFFFMAooopAFFFFABRRRQAUUUUAFFFFABRRRQAUUUUwCiiikAUUUUwCiiikAUUUUwCiiikAUUUUwCiiikAUUUUwCiiikAUUUUwCiiikAUUUUwCiiikAUUUUwCiiikAUUUUAFFFFMD//Z)

In C these return zero or non-zero.

In C++ these return true or false. Zero or non-zero still works for C++ but we don’t prefer it.

**Boolean type (bool)**

bool b1;

b1 = false;

b1 = true;

bool, false and true are C++ keywords.

true, false are language consts (constants).

if (a==b || c==d)

||

== == 🡪 In this case compiler evaluates left side first, right side second.

Short

Circuit

a b c d If left part is true, compiler doesn’t evaluate the second part.

If we use && instead of ||, then if left part is false, compiler doesn’t evaluate the right part.

If we have this:

if (a==b || ++c==d)

Then if first part is true, c won’t be incremented and vice versa. Don’t do this.

**PRECEDENCE OF OPERATORS**

|  |  |
| --- | --- |
| :: | scope resolution operator  std::cout 🡪 scope of cout is within this std namespace |
| .  ->  [ ]  ( )  ++  -- | Dot operator  Member selection  Array indexing (has 2 parameters, for a[1] a is first parameter, 1 is second parameter)  Function call  Postfix increment operator (placed after the variable)  Postfix decrement operator (placed after the variable) |
| ++  --  !  -  +  \*  &  new  delete  delete[ ]  sizeof  ( ) | Prefix increment operator (placed before the variable)  Prefix decrement operator (placed before the variable)  Not  Unary minus (not a-b but -b)  Unary plus (not a+b but +b)  Dereference (pointer dereference)  Address of  Create (allocate memory)  Destroy (deallocate)  Destroy array (deallocate)  Size of object  Type cast |
| \*  /  % | Multiply  Divide  Remainder (modulo) |
| +  - | Addition  Subtraction |
| <<  >> | Insertion operator (console output) - Left shift operator  Extraction operator (console input) - Right shift operator |
| <  >  <=  >= | Less than  Greater than  Less than or equal to  Greater than or equal to |
| ==  != | Equal  Not equal |
| && | Logical and |
| || | Logical or |
| =  +=  -=  \*=  /=  %= | Assignment  Add and assign  Subtract and assign  Multiply and assign  Divide and assign  Modulo and assign |
| ?: | Conditional operator |
| throw | Throw an exception |
| , | Comma operator |

for (i = 0, j = 1, k = 3 ; k < 5; ++i) 🡪 that’s where we use comma operator

Comma operator takes 2 parameters, evaluates them and returns the rightmost one’s result.

for

, < ++

1 = , k 5 i

i 0

= =

j 1 k 3

2 3

Operators that tells you which side (left) it is gonna evaluate first:

* ,
* &&
* ||
* ?: (Ternary operator)

Plus operator etc. doesn’t tell you “I am gonna evaluate this side first.”.

For all operators, associvity is left to right except assignment.

For assignment, associvity is right to left.

x + 1 > 2 || x + 1 < -3 SAME AS (x + 1) > 2 || (x + 1) < -3

left + will be done first because || evaluates left side first

**RULES OF OBJECT ORIENTED PROGRAMMING**

* Information hiding: You hide all the information from customer as much as possible.
* Encapsulation: Putting everything together that is needed to use a library.
* Inheritance: Code reuse example.

**Strong Enum**

C++ introduces strong enums or enum classes.

* Does not act like an integer.

enum class Days {Sun, Mon, Tue, Wed, Thu, Fri, Sat};

enum class Weather {Rain, Sun};

Days d = Days::Tue;

Weather w = Weather::Sun;

Don’t do if (d==0) do if (d == Days::Wed)

or I can’t say increment d etc.

**Branching Mechanism**

* if-else statements

needs a Boolean expression (int/non-int works for the if else statement but we don’t prefer it in C++)

== 🡪 returns a Boolean (true or false)

= (a = b;) 🡪 returns value of b. = operator has a side effect which is changing the value of a.

if ( x = 12 ) 🡪 this will return 12, 12 is non-zero so this code will Do\_Something

Do\_Something

bad coding

else

Do\_Something\_Else

Prefer 12==x instead of x==12. Because if you forgot one of the”=”, it becomes 12 = x and this gives an error message.

There is a option in g++ that will give you a warning message (not error) if you type if (x = 12)

**Nested Statements**

if (speed > 55)

if (speed > 80)

cout << “You’re really speeding!”;

else 🡪 belongs to second if.

cout << “You’re speeding.”;

You can type like this too.

Indentation is for you, not for the compiler.

This else still belongs to second if.

else

cout << “You’re speeding.”;

Do not use {} if not needed.

You can think if and else together as a 1 single statement here. It is compound statement by itself because it has 2 parts in it but it is a single statement. Compound statements are treated as a single statement.

You can use switch statement same as in C.

You can prefer switch statement in MENUs.

* Provides clearer “big-picture” view
* Shows menu structure effectively
* Each branch is one menu choice

**Conditional Operator (Ternary Operator)**

if (n1 > n2)

max = n1;

else IS SAME AS THIS 🡪 max = (n1>n2) ? n1 : n2;

max = n2; “?” and “:” form this “ternary” operator

n1>n2 will always be executed first.

So ternary operator guarantees the order of

execution

BOTH OF THESE PRODUCE SAME ASSEMBLY CODE

WHILE / DO-WHILE / FOR LOOPS ARE SAME AS IN C.

**Comma Operator**

* Evaluate list of expressions, returning value of the last expression.
* Most often used in a for-loop.
* Example:

first = (first=2, second=first+1);

Comma will evaluate “first=2” first.

Then it will evaluate “second=first+1.

Since comma is an operator, it will return something.

Comma returns the result of the second part which is assigned value which is 3.

* first gets assigned the value 3
* second gets assigned the value 3
* Guarantees the order of expression evaluations 🡪 left to right

**Loop Pitfalls: Misplaced ;**

while (response != 0); X

If this is your intention do this:

while (response != 0)

;

This would make your intention clearer to the reader.

This will make sense when you write multithreaded or multiprocess programs (in embedded systems) (more than 1 program running in the same time), they have both access to variable named response. One of them will be waiting for response to be 0, other one is working on the other stuff then it makes response 0 and waiting program will continue.

while (true) 🡪 infinite loop

**The break and continue Statements**

* These statements violate natural flow.
* Only used when absolutely necessary!

**Nested Loops**

* Any valid C++ statements can be inside body of loop.
* This includes additional loop statements!
  + Called “nested loops”.
* Requires careful indenting:

for (outer = 0; outer < 5; outer++)

for (inner = 7; inner > 2; inner--)

cout << outer << inner;

* Notice no { } since each body is 1 statement
* **Do not use { } if not needed!!!**

When you do like this:

if (a<2) {

b = 2;

}

it is thought that you forgot the 2nd or maybe the 3rd statements inside this block.