

Calculadora Posfija

Manual Técnico

Melissa Navarro Villalobos

2018086497

Samuel Piedra Araica

2019007537

ÍNDICE

Declaración de Estructuras	2
Principales Funciones	2
Funciones de Inventario	2
Funciones de Pila	3
Funciones Adicionales	4
Funciones de Control de Entrada	5
Funciones de Análisis y Resolución de String Infijo	8
Argumentos por Línea de Comando	11
Revisión	13
Confesiones, Pulgas, Errores	15

Declaración de Estructuras

Para realizar este programa fue necesario la creación de diversas estructuras, estas son de Operador, Expresión y Operacion como se muestra en la imagen

```
12 // Declaración de estructuras
13 typedef struct Operador
14 {
15     char Caracter;
16     Operador *Anterior;
17     Operador *Siguiente;
18 }*PtrOperador;
19 typedef struct Expresion
20 {
21     char Caracter;
22     float Valor;
23     Expresion *Anterior;
24     Expresion *Siguiente;
25 }*PtrExpresion;
26 typedef struct Operacion
27 {
28     string OperacionInfija;
29     string OperacionPosfija;
30     float Resultado;
31     Operacion *Anterior;
32     Operacion *Siguiente;
33 }*PtrOperacion;
```

struct Operador: Elementos de una lista que almacenan en una pila cada operador de la operación.

struct Expresion: Elementos de una lista de caracteres idéntica al string de la operación posfija. Aquí se realizan las operaciones de la calculadora. El float valor almacena resultados parciales de toda la operación.

struct Operación: Almacena el string infijo, el posfijo así como el resultado de la operación.

Principales Funciones

Funciones de Inventario

```
34
35 // Funciones de inventario de operadores
36 void Inicializar_Inventario_Operador(PtrOperador &Lista)
37 {
38     Lista = NULL;
39 }
40 PtrOperador Crear_Nuevo_Operador(char CaracterIngresando)
41 {
42     PtrOperador Nuevo_Operador = new(Operador);
43     Nuevo_Operador->Caracter = CaracterIngresando;
44     Nuevo_Operador->Siguiente = NULL;
45     Nuevo_Operador->Anterior = NULL;
46     return Nuevo_Operador;
47 }
48
49 void Agregar_Inicio_Inventario_Operador(PtrOperador &Lista, PtrOperador &Nuevo)
50 {
51     if (Lista != NULL)
52     {
53         Nuevo->Siguiente = Lista;
54         Lista->Anterior = Nuevo;
55     }
```

```

56     Lista = Nuevo;
57 }
58 else
59 {
60     Lista = Nuevo;
61 }
62 }
63 }
64 void Agregar_Final_Inventario_Operador(PtrOperador &Lista, PtrOperador &Nuevo)
65 {
66     if (Lista != NULL)
67     {
68         PtrOperador Aux = Lista;
69
70         while (Aux->Siguiente != NULL)
71         {
72             Aux = Aux->Siguiente;
73         }
74         Aux->Siguiente = Nuevo;
75         Nuevo->Anterior = Aux;
76     }
77     else

```

```

78     {
79         Lista = Nuevo;
80     }
81 }
82 void Listar_Inventario_Operador(PtrOperador Lista)
83 {
84     if (Lista != NULL)
85     {
86         PtrOperador Aux = Lista;
87         while (Aux != NULL)
88         {
89             cout << Aux->Caracter;
90             Aux = Aux->Siguiente;
91         }
92         cout << endl << endl;
93     }
94     else
95     {
96         cout << "No hay elementos para listar." << endl << endl;
97     }
98 }
99

```

Para el inventario de Operadores se utilizaron las siguientes funciones:

Inicializar_Inventario_Operador(): Crea una lista vacía

Crear_Nuevo_Operador(): Crea elementos dentro del struct

Agregar_Inicio_Inventario_Operador(): Los elementos se agregan al inicio de la lista

Agregar_Final_Inventario_Operador(): Los elementos se agregan al final de la lista

Listar_Inventario_Operador(): Se listan los elementos

Funciones de Pila

```

290 void Push_Operador(PtrOperador &Lista, PtrOperador &Nuevo)
291 {
292     Agregar_Inicio_Inventario_Operador(Lista, Nuevo);
293 }
294 PtrOperador Top_Operador(PtrOperador Lista)
295 {
296     return Lista;
297 }
298 PtrOperador Pop_Operador(PtrOperador &Lista)
299 {
300     if (Lista != NULL) // Verifica la existencia de elementos dentro de la lista
301     {
302         PtrOperador Aux = Lista; // Crea un puntero auxiliar
303         Lista = Aux->Siguiente; // Cambia la dirección de Lista hacia el siguiente elemento
304         Aux->Siguiente = NULL; // Redirecciona el puntero del elemento hacia NULL
305         return Aux; // Regresa la dirección de un elemento aislado
306     }
307     else
308     {
309         cout << "No hay elementos para hacer POP." << endl << endl;
310         return Lista;
311     }

```

Para funciones de pila de Operadores se utilizaron las siguiente funciones:

Push_Operador(): Agrega los elementos al inicio de la pila

Top_Operador(): Regresa la dirección del último elemento de la lista

Pop_Operador(): Elimina elementos de la pila

Para cada Struct (Inventario y Operaciones) se repetirán las funciones de inventario así como de pila

Funciones Adicionales

```
331 // Funciones adicionales convenientes
332 float FuncionFactorial(float Valor) { //resuelve el factorial de un numero
333     if (Valor == 0 || Valor == 1)
334         return 1;
335     else
336         return(Valor*FuncionFactorial(Valor - 1));
337 }
338 int validarOpcionMenu(int valorMinimo, int ValorMaximo)
339 {
340     int opcion;
341     while (true)
342     {
343         cout << "Ingresar una opcion valida: "; // Itera la instrucción hasta que sea un valor dentro de un rango establecido
344         cin >> opcion;
345         if ((opcion >= valorMinimo) && (opcion <= ValorMaximo))
346         {
347             break;
348         }
349     }
350     return opcion; // Regresa un valor permitido
351 }
352
```

FuncionFactorial(): En esta función se resuelven todas las operaciones factoriales

ValidarOpcionMenu(): Verifica que las opciones que se escojan en el menú sean válidas.

Funciones de Control de Entrada

```
353 // Funciones de control de entrada
354 char SwitchLetras(char Letra)
355 {
356     char Caracter;
357
358     switch (Letra)
359     {
360     case '+':
361     {
362         Caracter = '+';
363         break;
364     }
365     case '-':
366     {
367         Caracter = '-';
368         break;
369     }
370     case '*':
371     {
372         Caracter = '*';
373         break;
374     }
375     case '/':
376     {
377         Caracter = '/';
378         break;
379     }
380     case '^':
381     {
382         Caracter = '^';
383         break;
384     }
385     case '!':
386     {
387         Caracter = '!';
388         break;
389     }
390     default:
391     {
392         Caracter = '?';
393     }
394     }
395
396     return Caracter;
```

Switch_Letras(): Analiza el símbolo ingresado por el usuario y determina si este es un operador válido (+, -, *, /, !, ^)

```

398 bool ContarParentesis(string Operacion)
399 {
400     int NumParentesisApertura = 0;
401     int NumParentesisCierre = 0;
402     int Largo = Operacion.length();
403     bool resultado = false;
404
405     for (int i = 0; i < Largo; i++)
406     {
407         char Letra = Operacion[i];
408
409         switch (Letra)
410         {
411             case '(':
412             {
413                 NumParentesisApertura++;
414                 break;
415             }
416             case ')':
417             {
418                 NumParentesisCierre++;
419                 break;
420             }
421         }
422     }
423
424     if ((NumParentesisApertura == NumParentesisCierre) && (NumParentesisApertura != 0) && (NumParentesisCierre != 0))
425     {
426         resultado = true;
427     }
428
429     return resultado;
430 }

```

```

407     char Letra = Operacion[i];
408
409     switch (Letra)
410     {
411         case '(':
412         {
413             NumParentesisApertura++;
414             break;
415         }
416         case ')':
417         {
418             NumParentesisCierre++;
419             break;
420         }
421     }
422
423     if ((NumParentesisApertura == NumParentesisCierre) && (NumParentesisApertura != 0) && (NumParentesisCierre != 0))
424     {
425         resultado = true;
426     }
427
428     return resultado;
429 }

```

ContarParentesis(): Una función booleana donde se lleva el control de cuantos paréntesis ingresó el usuario para realizar la operación, en esta función se verifica que el número “(“ sea el mismo al número de “)”

```

429 bool ContarOperadores(string Operacion)
430 {
431     int Operandos = 0;
432     int Operadores = 0;
433     int Largo = Operacion.length();
434
435     bool Condicion = false;
436
437     for (int i = 0; i < Largo; i++)
438     {
439         char Letra = Operacion[i];
440
441         if (isalpha(Letra) != 0)
442         {
443             Operandos++;
444         }
445         else
446         {
447             char Evaluando = SwitchLetras(Letra);
448
449             if ((Evaluando != '?') && (Evaluando != '(') && (Evaluando != ')'))
450             {
451                 Operadores++;
452             }
453         }
454     }
455
456     if (Operandos == Operadores)
457     {
458         Condicion = true;
459     }
460
461     return Condicion;
462 }

```

```

451         Operadores++;
452     }
453     if (Evaluando == '!')
454     {
455         Operandos++;
456     }
457 }
458 }
459
460 if (Operandos - Operadores == 1)
461 {
462     Condicion = true;
463 }
464
465 return Condicion;
466 }
467 bool VerificarOperadores(string Operacion) { ... }
510 bool VerificarEntrada(string Operacion)
511 {
512     bool Resultado = false;
513
514     bool Condicion1 = ContarParentesis(Operacion);

```

Contar Operadores(): Se asegura que haya una cantidad menor de operadores que de variables (la diferencia de esta va a ser de 1)

```

467 bool VerificarOperadores(string Operacion)
468 {
469     bool Condicion = true;
470     int Largo = Operacion.length();
471
472     for (int i = 0; i < Largo; i++)
473     {
474         char Letra = Operacion[i];
475         char LetraSwitch = SwitchLetras(Letra);
476
477         if (i < Largo - 1)
478         {
479             char Siguiente = Operacion[i + 1];
480             char SiguienteSwitch = SwitchLetras(Siguiente);
481
482             if ((Letra == '(') && (Siguiente == '('))
483             {
484                 Condicion = false;
485                 cout << "No se puede agregar parentesis que no tengan contenido." << endl;
486             }
487             if ((Letra == ')') && (Siguiente == '('))
488             {

```

```

489                 Condicion = false;
490                 cout << "Los grupos de parentesis deben separarse por un operando." << endl;
491             }
492             if ((LetraSwitch != '?') && (SiguienteSwitch != '?'))
493             {
494                 Condicion = false;
495                 cout << "No se puede agregar dos operandos juntos." << endl;
496             }
497         }
498     }
499     else
500     {
501         if (Letra != ')')
502         {
503             Condicion = false;
504             cout << "El ultimo caracter no puede ser diferente de un parentesis de cierre." << endl;
505         }
506     }
507
508     return Condicion;
509 }
510 bool VerificarEntrada(string Operacion)

```

VerificarOperadores(): Son reglas adicionales para el uso de operadores y paréntesis. No puede haber un número mayor de operadores que de variables, no pueden existir parentesis sin contenido y el último carácter ingresado no puede ser diferente a “)”


```

510 bool VerificarEntrada(string Operacion)
511 {
512     bool Resultado = false;
513
514     bool Condicion1 = ContarParentesis(Operacion);
515     bool Condicion2 = ContarOperadores(Operacion);
516     bool Condicion3 = VerificarOperadores(Operacion);
517
518     if ((Condicion1 == true) && (Condicion2 == true) && (Condicion3 == true))
519     {
520         Resultado = true;
521     }
522     if (Condicion1 == false)
523     {
524         cout << "El numero de parentesis de cierre debe ser igual al de apertura, ambos distintos de cero." << endl;
525     }
526     if (Condicion2 == false)
527     {
528         cout << "La suma de operadores nunca iguala o supera la de operandos." << endl;
529     }
530
531     return Resultado;

```

VerificarEntrada(): Verifica que de las funciones anteriores el resultado sea verdadero para poder continuar con la operación

Funciones de Análisis y Resolución de String Infijo

```

534 // Funciones de análisis y resolución de string infijo
535 string OrdenarElementos(string Operacion, PtrExpresion &PilaExpresiones, PtrOperador &PilaOperadores)
536 {
537     string CadenaPosfija;
538     int Largo = Operacion.length();
539
540     for (int i = 0; i < Largo; i++)
541     {
542         char Letra = Operacion[i];
543         PtrOperador AuxOperador;
544         PtrExpresion Aux = Crear_Nuevo_Expresion(Letra);
545
546         if (isalpha(Letra) != 0)
547         {
548             Agregar_Final_Inventario_Expresion(PilaExpresiones, Aux);
549             CadenaPosfija += Letra;
550         }
551         else if (Letra == ')')
552         {
553             AuxOperador = Pop_Operador(PilaOperadores);
554             Aux->Caracter = AuxOperador->Caracter;
555             Agregar_Final_Inventario_Expresion(PilaExpresiones, Aux);
556
557             Letra = Aux->Caracter;
558             CadenaPosfija += Letra;
559         }
560         else
561         {
562             char LetraSwitch = SwitchLetras(Letra);
563
564             if (Letra == '(')
565             {
566                 AuxOperador = Crear_Nuevo_Operador('(');
567                 Push_Operador(PilaOperadores, AuxOperador);
568             }
569             else if (LetraSwitch != '?')
570             {
571                 AuxOperador = Crear_Nuevo_Operador(Letra);
572                 Push_Operador(PilaOperadores, AuxOperador);
573             }
574         }
575     }
576
577     return CadenaPosfija;

```

Ordenar Elementos(): Convierte la operación de infija a posfija. Posee una pila que almacena operadores y extrae sus elementos cuando encuentra un “)”. Cada variable y operador válido que encuentra los almacena en una lista especial para resultados parciales de la operación (en una lista enlazada del Struct Expresion). El orden de esta lista es el mismo que el del string posfijo.

```

579 void SustituirVariables(PtrExpresion &PilaExpresiones)
580 {
581     PtrExpresion Aux = PilaExpresiones;
582
583     while (Aux != NULL)
584     {
585         char Letra = Aux->Caracter;
586
587         if (isalpha(Letra) != 0)
588         {
589             cout << "Ingrese el valor de la variable " << Letra << " ";
590             cin >> Aux->Valor;
591         }
592         Aux = Aux->Siguiente;
593     }
594 }
595 float ResolverOperacionParcial(PtrExpresion &Referencia, PtrExpresion Operando1, PtrExpresion Operando2) { ... }
606 void ResolverOperacion(PtrExpresion &PilaExpresiones) { ... }
607
608 // Programa principal
609 int main(int argc, char **argv) { ... }

```

SustituirVariables(): Lista enlazada que asigna un número al atributo “valor” de cada elemento que sea una variable. Pide los valores en la pantalla

```

595 float ResolverOperacionParcial(PtrExpresion &Referencia, PtrExpresion Operando1, PtrExpresion Operando2)
596 {
597     char Operador = Referencia->Caracter;
598     float resultado;
599
600     switch (Operador)
601     {
602     case '+':
603     {
604         resultado = Operando1->Valor + Operando2->Valor;
605         break;
606     }
607     case '-':
608     {
609         resultado = Operando2->Valor - Operando1->Valor;
610         break;
611     }
612     case '*':
613     {
614         resultado = Operando1->Valor * Operando2->Valor;
615         break;
616     }
617     case '/':
618     {
619         resultado = Operando2->Valor / Operando1->Valor;
620         break;
621     }
622     case '^':
623     {
624         resultado = pow(Operando2->Valor, Operando1->Valor);
625         break;
626     }
627     case '!':
628     {
629         resultado = FuncionFactorial(Operando1->Valor);
630         break;
631     }
632     }
633
634     return resultado;
635 }
636 void ResolverOperacion(PtrExpresion &PilaExpresiones) { ... }
637

```

ResolverOperacionParcial(): De acuerdo al operador que se encuentra en la función OperacionParcial() realiza una operación u otra. Ingresa el valor de cada elemento necesario para resolver la operación

```

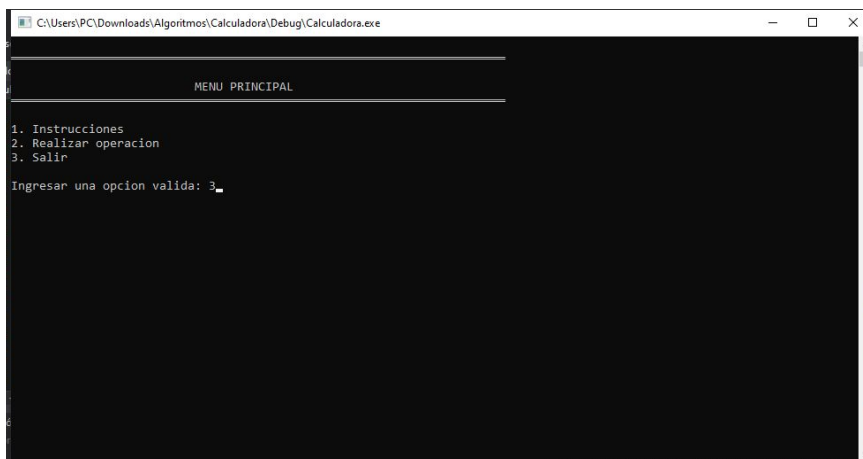
636 void ResolverOperacion(PtrExpresion &PilaExpresiones)
637 {
638     if (PilaExpresiones != NULL)
639     {
640         PtrExpresion Aux = PilaExpresiones;
641         bool Condicion = true;
642
643         while (Condicion == true) // Busca la aparición del primer operador en la cadena posfija
644         {
645             char Letra = Aux->Caracter;
646
647             if (isalpha(Letra) == 0)
648             {
649                 Condicion = false;
650             }
651             else
652             {
653                 Aux = Aux->Siguiete;
654             }
655         }
656
657         if (Aux->Caracter != '!')
658
659             PtrExpresion Operando1 = Sacar_Inventario_Expresion_Anterior(PilaExpresiones, Aux);
660             PtrExpresion Operando2 = Sacar_Inventario_Expresion_Anterior(PilaExpresiones, Aux);
661
662             if ((Operando1 != NULL) && (Operando2 != NULL))
663             {
664                 Aux->Valor = ResolverOperacionParcial(Aux, Operando1, Operando2);
665                 Aux->Caracter = 'A';
666             }
667         }
668         else
669         {
670             PtrExpresion Operando1 = Sacar_Inventario_Expresion_Anterior(PilaExpresiones, Aux);
671
672             if (Operando1 != NULL)
673             {
674                 Aux->Valor = ResolverOperacionParcial(Aux, Operando1, NULL);
675                 Aux->Caracter = 'A';
676             }
677         }
678
679         PilaExpresiones = Aux;
680
681         Aux->Caracter = 'A';
682     }
683     else
684     {
685         PtrExpresion Operando1 = Sacar_Inventario_Expresion_Anterior(PilaExpresiones, Aux);
686
687         if (Operando1 != NULL)
688         {
689             Aux->Valor = ResolverOperacionParcial(Aux, Operando1, NULL);
690             Aux->Caracter = 'A';
691         }
692     }
693
694     PilaExpresiones = Aux;
695
696     if (PilaExpresiones->Siguiete != NULL)
697     {
698         ResolverOperacion(PilaExpresiones);
699     }
700 }

```

ResolverOperacion(): Recorre una lista enlazada, por cada vez que encuentra un operador, busca los dos elementos anteriores, es decir, la variables. Toma los valores de estos y los ingresa en la función ResolverOperacionParcial(). Para este proceso se trabajan tres elementos de la lista enlazada (uno por el operador y 2 por las variables). Al obtener el resultado parcial se sustituyen los tres elementos en uso por 1 que almacenará el resultado parcial, al realizar esto se disminuirá el tamaño de la lista hasta que solamente quede un elementos

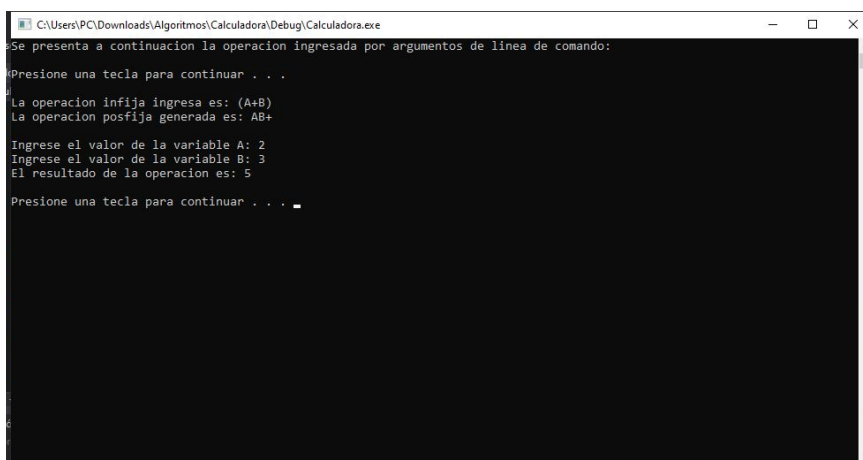
Argumentos por Línea de Comando

La calculadora puede resolver operaciones que se ingresan como parámetros desde la línea de comando. El único requisito es ingresar el string de la operación como argumento. El programa funcionará de exactamente la misma forma que lo hace para resolver operaciones que se ingresan por medio del menú; solamente hay que cerrar el menú principal para que se ejecute un apartado del código que trabaja con argumentos por línea de comando.



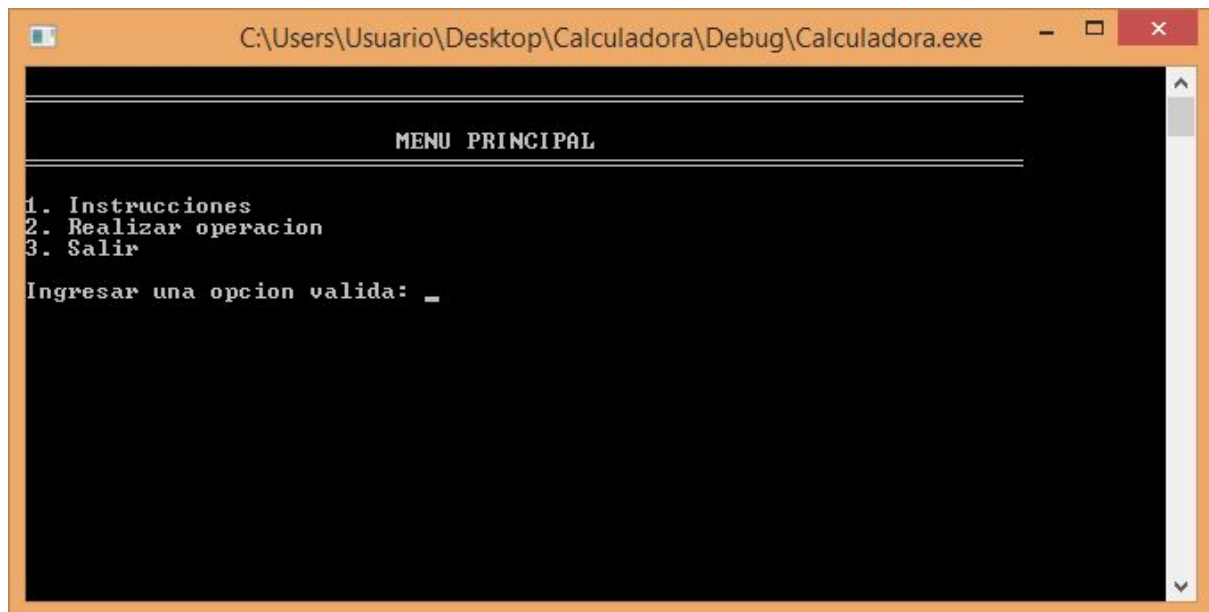
```
C:\Users\PC\Downloads\Algoritmos\Calculadora\Debug\Calculadora.exe
MENU PRINCIPAL
1. Instrucciones
2. Realizar operacion
3. Salir
Ingresar una opcion valida: 3_
```

Aparecerá un conjunto de líneas que indican la operación ingresada y la forma de proceder en adelante. Al finalizar se cierra el código.

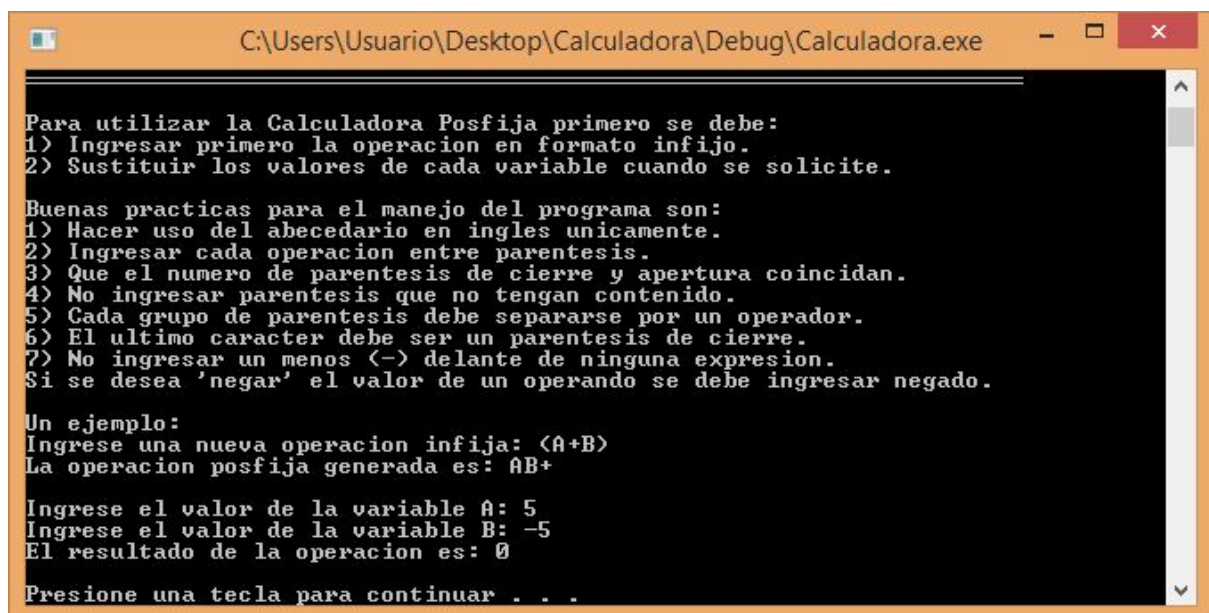


```
C:\Users\PC\Downloads\Algoritmos\Calculadora\Debug\Calculadora.exe
Se presenta a continuacion la operacion ingresada por argumentos de linea de comando:
Presione una tecla para continuar . . .
La operacion infija ingresa es: (A+B)
La operacion posfija generada es: AB+
Ingrese el valor de la variable A: 2
Ingrese el valor de la variable B: 3
El resultado de la operacion es: 5
Presione una tecla para continuar . . .
```

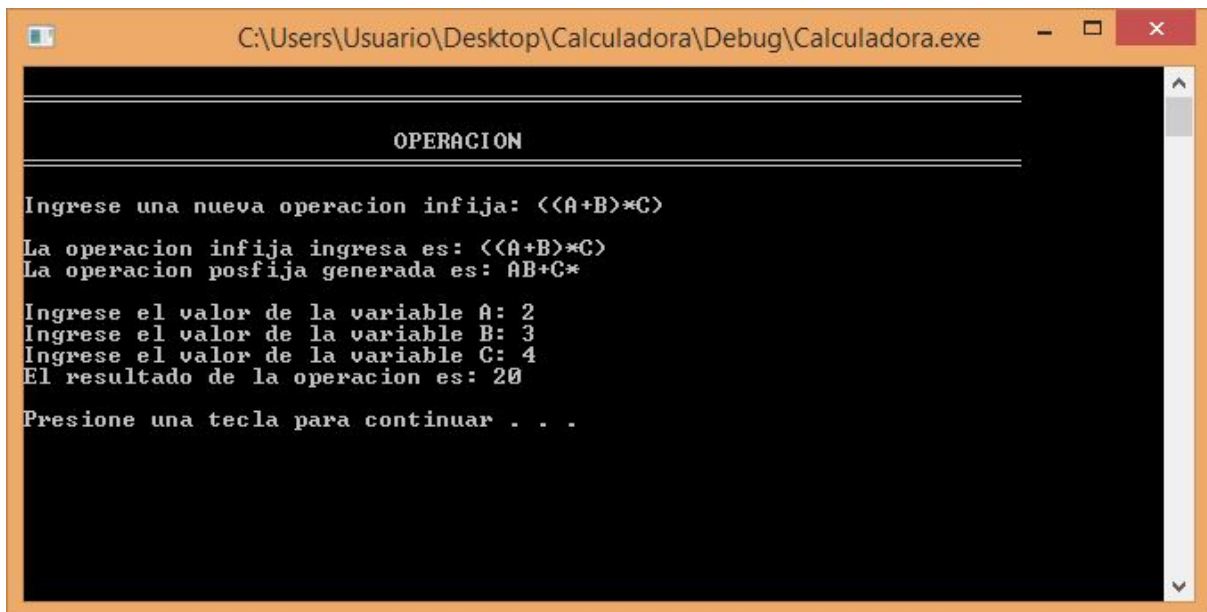

Revisión



Al iniciar el programa se muestra el menú principal



En la primera opción se mostrarán las instrucciones para utilizar la calculadora de la mejor manera



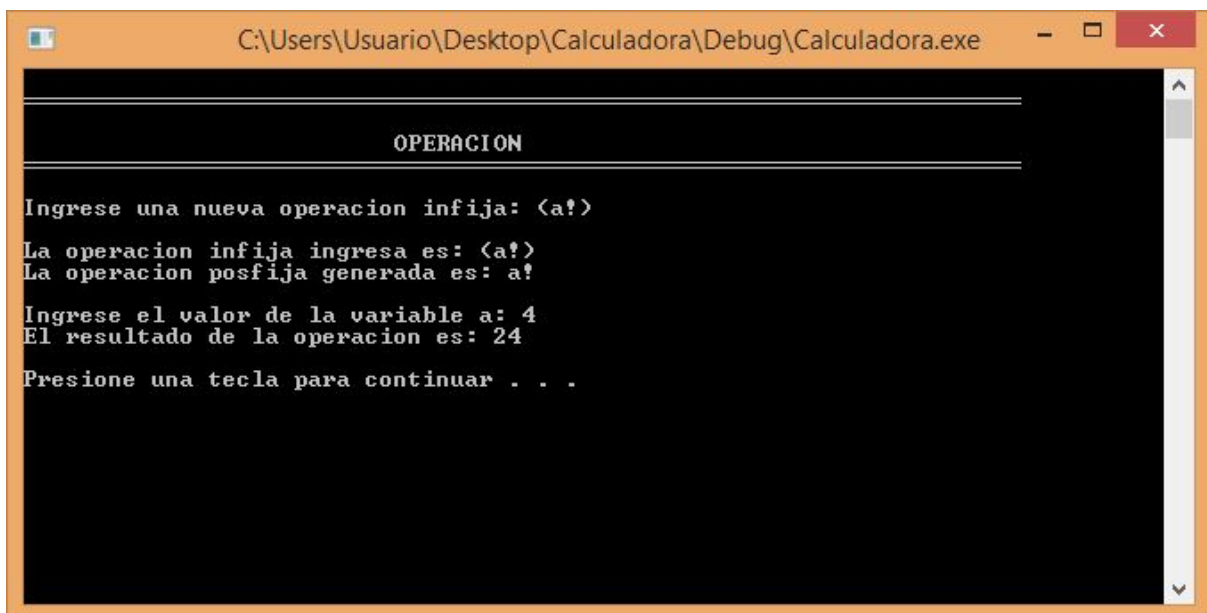
```
OPERACION

Ingrese una nueva operacion infija: <<(A+B)*C>
La operacion infija ingresa es: <<(A+B)*C>
La operacion posfija generada es: AB+C*

Ingrese el valor de la variable A: 2
Ingrese el valor de la variable B: 3
Ingrese el valor de la variable C: 4
El resultado de la operacion es: 20

Presione una tecla para continuar . . .
```

En la segunda opción se pedirá primero la operación infija, seguidamente se mostrará la operación posfija que se generará y se pedirán los valores de las variables. También mostrará el resultado en pantalla



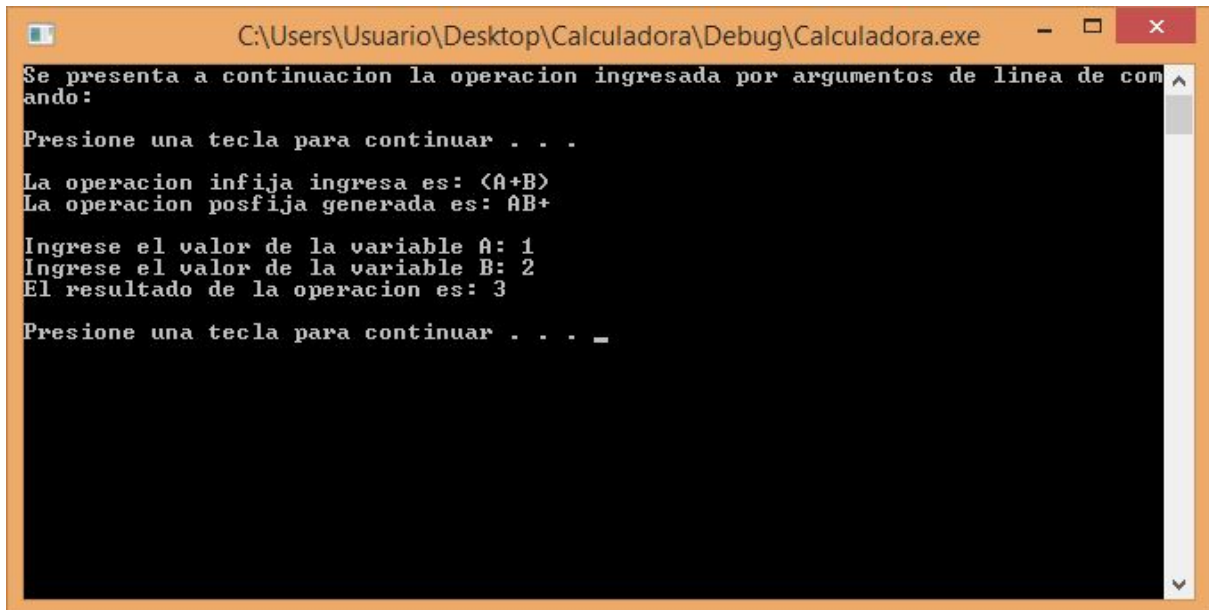
```
OPERACION

Ingrese una nueva operacion infija: <a!>
La operacion infija ingresa es: <a!>
La operacion posfija generada es: a!

Ingrese el valor de la variable a: 4
El resultado de la operacion es: 24

Presione una tecla para continuar . . .
```

Ejemplo de operación con factorial




```
Se presenta a continuacion la operacion ingresada por argumentos de linea de comando:
Presione una tecla para continuar . . .
La operacion infija ingresa es: <A+B>
La operacion posfija generada es: AB+
Ingrese el valor de la variable A: 1
Ingrese el valor de la variable B: 2
El resultado de la operacion es: 3
Presione una tecla para continuar . . . _
```

En la opción 3, se desglosará el valor de la operación ingresada por argumentos de línea de comando

Confesiones, Pulgas, Errores

1. Es probable que para una determinada forma de escribir la operación se produzcan errores. Como es en el siguiente caso:



```
OPERACION
Ingrese una nueva operacion infija: (A+(B*C))
La operacion infija ingresa es: (A+(B*C))
La operacion posfija generada es: ABC*(
Ingrese el valor de la variable A: 2
Ingrese el valor de la variable B: 3
Ingrese el valor de la variable C: 4
El resultado de la operacion es: -4.31602e+08
Presione una tecla para continuar . . . _
```

Que puede resolverse al escribir la operación de otra forma:


```
C:\Users\PC\Downloads\Algoritmos\Calculadora\Debug\Calculadora.exe

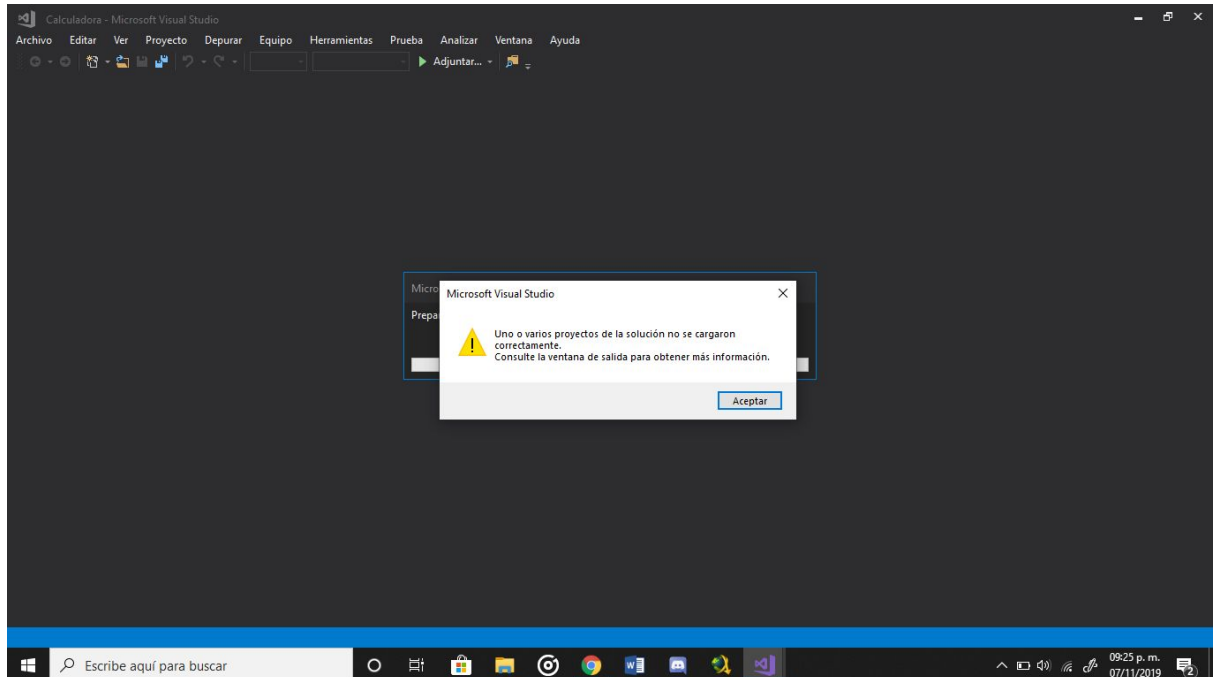
OPERACION

Ingrese una nueva operacion infija: ((B*C)+A)
La operacion infija ingresa es: ((B*C)+A)
La operacion posfija generada es: BC*A+

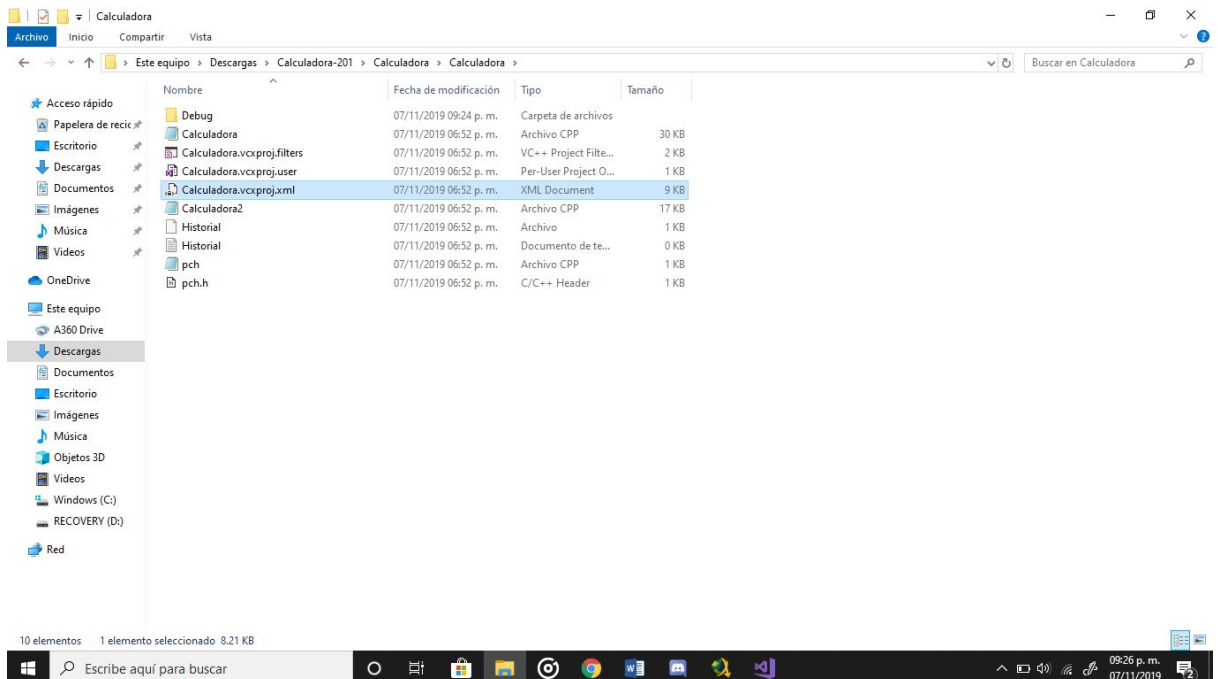
Ingrese el valor de la variable B: 3
Ingrese el valor de la variable C: 4
Ingrese el valor de la variable A: 2
El resultado de la operacion es: 14

Presione una tecla para continuar . . .
```

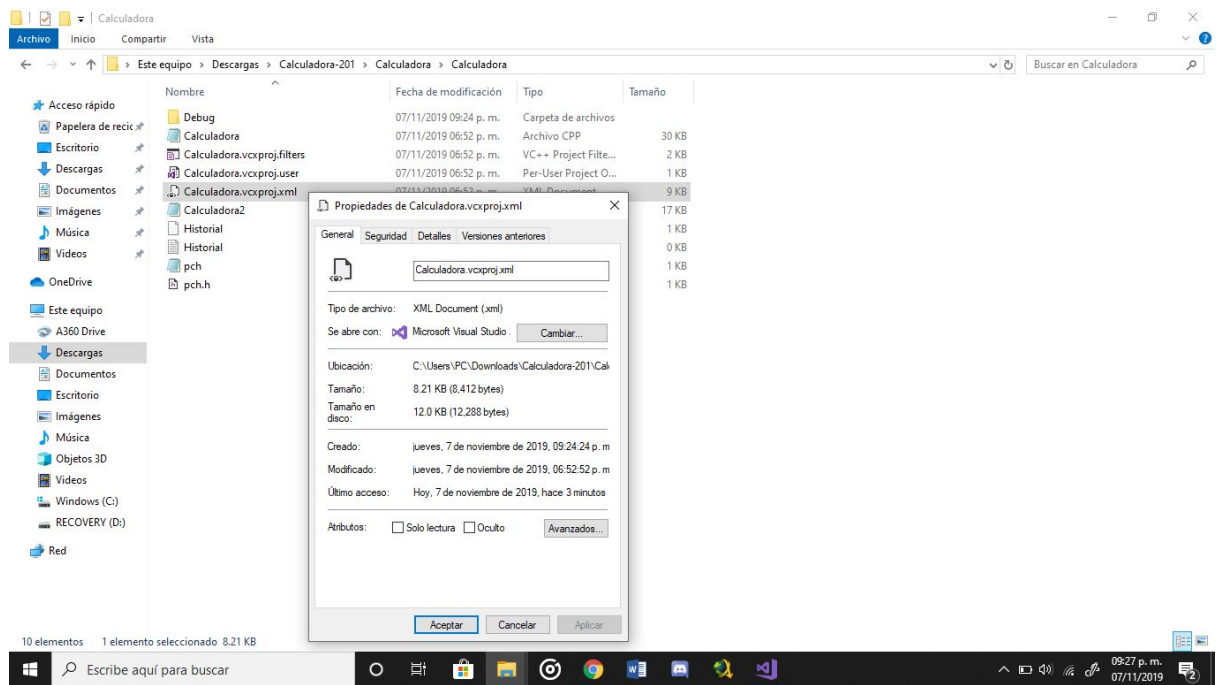
2. Es probable que después de haber descargado el archivo del Google Drive e intentar acceder al código aparezca lo siguiente:



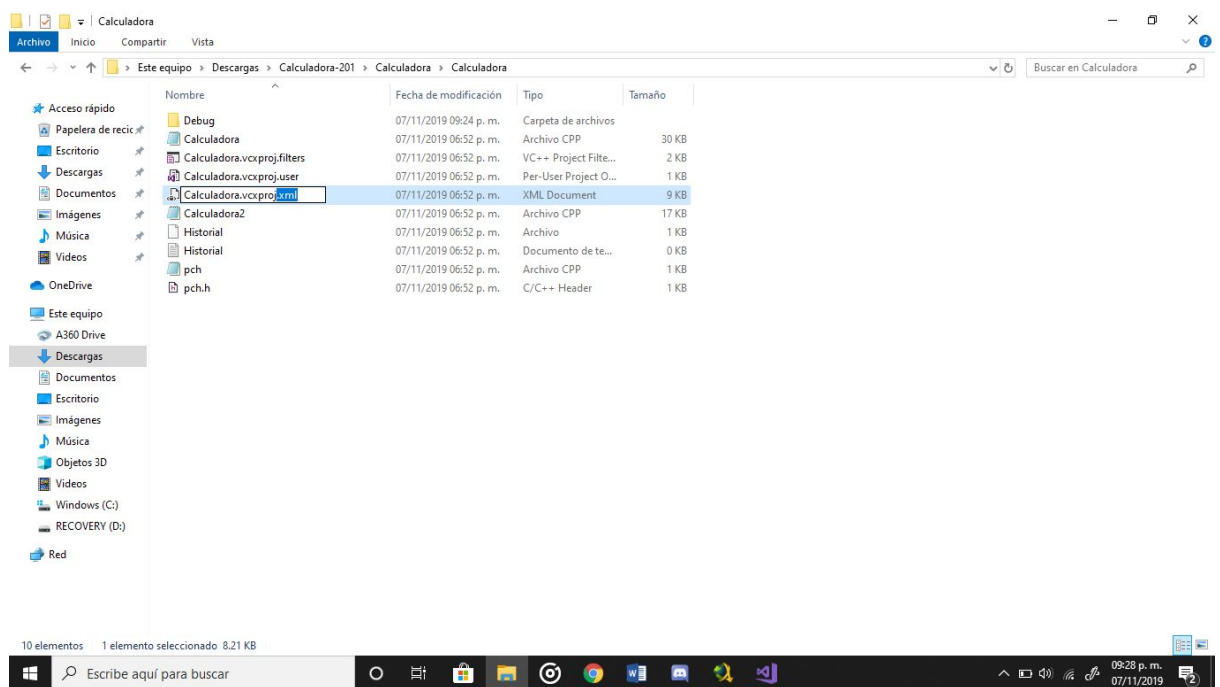
Para resolver esta situación planteamos lo siguiente:



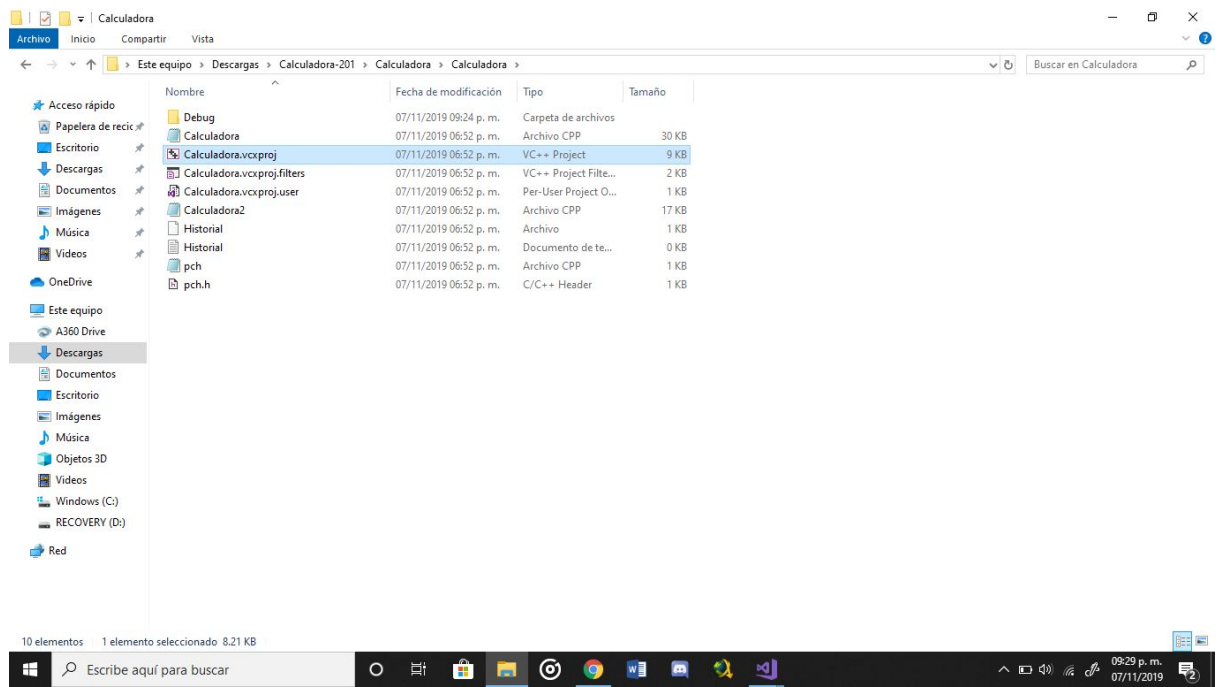
Revisar si en la carpeta del código aparece un archivo de tipo “.xml”. Creemos que el método que usa Google Drive para descomprimir archivos de algún modo altera la extensión de ese archivo. La solución está en cambiarla.



Primero revisando que el archivo se abra con Microsoft Visual Studio.



Luego borrando la extensión “.xml”



Para conseguir este resultado. Bajo estas condiciones ya funciona correctamente el código.