Desk Calculator

Alumno: Nicolás Figueroa Gómez
Código: 201810294
Profesor: Rubén Rivas Medina

Introducción

En la actualidad estamos en una constante interacción con mecanismos que tienen implementados software que nos proporciona una mayor facilidad al resolver problemas de la vida cotidiana. Pero, para un buen crecimiento intelectual es necesario saber el funcionamiento básico de estos, como por ejemplo una calculadora de escritorio.

Para ello, se determinó el desarrollo e implementación de un software que simule e interprete el funcionamiento de una calculadora de escritorio. Por ello, nos permitirá entender un poco más cómo es posible el funcionamiento y determinación de cuál es la respuesta correcta y como la calculadora lo identifica como tal, el símbolo que nosotros enviamos.

Desarrollo

Para la implementacion del codigo tendremos los siguientes archivos: el main.cpp y calculadora.h.

Calculadora.h

Se utilizó las siguiente librerías ->

```cpp
#include <iostream>
#include <map>
#include <sstream>
#include <string>
#include <cctype>
#include <cmath>
```

```cpp
enum Token_value
{
    NAME,       NUMBER,     END,        FUNC = '$',
    ADD = '+',  SUB = '-',  MUL = '*',  DIV = '/',
    PRINT = ';', ASSIGN = '=', LP = '(', RP = ')',
    EXPO = '^', LCB = '{',  RCB = '}',  COMMA = ','
};
```

Tipo enumeración que controla los TOKEN o entradas utilizadas por el analizador, tener definidas las operaciones u opciones en este tipo, facilita la programación de la calculadora.

Calculadora.h

```
Token_value curr_tok = PRINT;
```

Se crea la variable de enumeración y se inicializa en PRINT, garantizando la ejecución.

```
Token_value get_token();
```

Función de entradas (input), su encabezado, la definición de la misma es posterior.

```
double number_value;
string string_value;
```

Variables globales que se utilizan para obtener el valor de un token o entrada.

```
int no_of_errors;
double error(const string& s);
```

Variable que contiene el error y función que maneja ese error (Error handling (manejo))

Calculadora.h

```
map<string, double> globals;
```

Tabla de símbolos utilizada para almacenar variables y constantes predefinidas

```
struct Function
{
    Function() : arg_count(0) {}
    Function(const string& nm) : name(nm), arg_count(0) {}

    string name;                    // Nombre de la funcion
    size_t arg_count;               // Numero de argumentos que acepta la funcion
    map<string, double> arg_table; // Diccionario de argumentos de pareja nombre/valor
    string body;                    // Definicion
};
```

Estructura para Funciones definidas por el usuario y el mapa para asociar el nombre de la función a la función real. Para definir una función, debe escribir, por ejemplo: "$ func (x) {x = x + 2;}" y la llamada sería: "$ func (20)" e indicará el resultado. Puede guardar el valor de la función en una variable o combinar una llamada de función con otras expresiones.

Calculadora.h

```
map<string, Function> func_table;
```

Mapa o Tabla de símbolos para funciones pre definidas por el usuario.

```
bool funcExists(const string& name);
```

Devuelve si existe una función con el nombre dado en func_table.

Calculadora.h

```
double error(const string& s)
{
    ++no_of_errors;
    cerr << "Error: \"" << s << "\"" << endl;
    return 1;
}
```

Función de manejo (handling) de errores.

Calculadora.h

Función de las entradas a la calculadora.

```
Token_value get_token()
{
    char ch = 0;

    // Se lee la entrada
    do
    {   // Se omiten los espacios en blanco, excepto '\n'
        if(!input->get( & ch))
            return curr_tok = END;
    } while(ch != '\n' && isspace(ch));

    // Se compara la entrada
    switch(ch)
    {
        case 0:
            return curr_tok = END; // Orden de finalizar
        case ';':
        case '\n':
            return curr_tok = PRINT; // Orden de continuar
        case '+':
        case '-':
        case '*':
        case '/':
        case '(':
        case ')':
        case '=':
        case ',':
        case '^':
        case '}':
            return curr_tok = Token_value(ch); // Se retorna el valor

            // Elementos numericos de la calculadora
        case '0': case '1': case '2': case '3': case '4':
        case '5': case '6': case '7': case '8': case '9':
        case '.':
            input->putback(ch);       // Se agrega el caracter al string que sera objeto calculo
            *input >> number_value;   // Su contenido se asigna a la variable global: number_value
            return curr_tok = NUMBER; // Indicamos que es un numero
```

Calculadora.h

```
    case '$': // Como se explico en la estructura, se define la funcion o la llamada
    {
        if(!input->get( & ch) || !isalpha(ch)) // Caso de la opcion de salir de la calculadora
            return curr_tok = END;
        else
            string_value = ch; // Asignamos el caracter

        // Se diferencia en la entrada entre digitos y letras
        while(input->get( & ch) && (isalnum(ch) || ch == '_'))
        {
            string_value += ch; // Se anexa el elemento
        }
        input->putback(ch); // Asignamos el elemento al final del input
        return curr_tok = FUNC; // Retornamos el toquen de tipo FUNC
    }
    case '{':
    {
        string_value.clear(); // Limpiamos el string de valores
        while(input->get( & ch) && ch != '}') // Mientras no se reciba la llave de cierre, asignamos elementos
        {
            string_value += ch;
        }
        return curr_tok = LCB;
    }
    default: // Nombre o se produjo un error
        if(isalpha(ch)) // Si es alfabetico
        {
            string_value = ch;
            while(input->get( & ch) && (isalnum(ch) || ch == '_'))
                string_value.push_back(ch);  // Asignamos al final del string
            input->putback(ch);
            return curr_tok = NAME; // Retornamos el nombre
        }
        error( s: "bad token"); // Se produjo un error
        return curr_tok = PRINT; // Orden de continuar
}
```

Calculadora.h

Función definida para la suma y la resta.

```
double expr(bool get)
{
    double left = term(get);

    for(;;) // Ciclo infinito, se suponen infinidad de elementos en una expresion de calculo
    {
        switch(curr_tok)
        {
            case ADD: // Opcion de suma
            {
                left += term( get: true);
                break;
            }
            case SUB: // Opcion de resta
            {
                left -= term( get: true);
                break;
            }
            default:
                return left; // Retornaos el elemento
        }
    }
}
```

Calculadora.h

Manejo de la multiplicación, división y exponencial.

```
double term(bool get)
{
    double left = prim(get);

    for(;;) // Ciclo infinito, se suponen infinidad de elementos en una expresion de calculo
    {
        switch(curr_tok)
        {
            case MUL: // Multiplicacion
            {
                left *= prim( get: true);
                break;
            }
            case DIV: // Division
            {
                if(double d = prim( get: true))
                {
                    left /= d;
                    break;
                }
                return error( s: "divide by zero");
            }
            case EXPO: // Exponencial
            {
                left = pow(left, prim( get: true));
                break;
            }
            default:
                return left;
        }
    }
}
```

Calculadora.h

Manejo primarias, como nombre o número.
La función prim () usa para obtener valores para las variables, un puntero a un mapa, y para las expresiones regulares, se establece en la tabla de símbolos principal. Cuando se evalúa una función se señala el mapa de argumentos para esa función y copia las variables globales en ella, para que puedan usarse en la llamada a la función.

```
double prim(bool get)
{
    if(get) get_token(); // Se lee un token

    switch(curr_tok) // CASE de la varible de enumeracion: curr_tok
    {
        case NUMBER: // Notese la recursividad
        {
            double v = number_value;
            get_token();
            return v;
        }
        case NAME:
        {
            double& v = table->operator[](string_value);
            if(get_token() == ASSIGN)
                v = expr( get: true);
            return v;
        }
        case FUNC:
        {
            if(funcExists(string_value))    // Verificamos si la funcion existe
            {
                Function f(func_table[string_value]); // Se llama la funcion

                if(get_token() == LP)
                {
                    map<string, double>::iterator it = f.arg_table.begin();
                    while(get_token() != RP && curr_tok != END)
                    {
                        if(it == f.arg_table.end()) continue;
```

Calculadora.h

```
        switch(curr_tok)
        {
            case NAME:
                if(table->count(string_value) > 0)
                {
                    it->second = table->operator[](string_value);
                    ++it;
                    break;
                }
                else
                    return error( s: "argumento de funcion desconocida");
            case NUMBER:
                it->second = number_value;
                ++it;
                break;
            case COMMA:
                continue;
            default:
                return error( s: "argumento de funcion no valida");
        }
    }

    /*
     * Comprueba si el número de argumentos es el mismo para la llamada a la función
     * y la definición de la misma:
     */
    if(f.arg_table.size() == f.arg_count)
    {
        // Ahora evalúa la función:
        istringstream func(f.body);
        istream* tmp = input;
        input = &func;
        map<string, double>* tmptable = table; /* Asignacion temporal de la tabla,
                                                  para evitar que se pierda */
```

Calculadora.h

```
                    table = &f.arg_table;
                    it = tmptable->begin();
                    // Copiar variables globales a la tabla de símbolos:
                    while(it != tmptable->end())
                    {
                        table->operator[](it->first) = it->second;
                        ++it;
                    }
                    double value = 0;
                    while(*input) // Mientras existan entrada
                    {
                        get_token();                    // Leemos
                        if(curr_tok == END) break;      // Verificamos si salimos
                        if(curr_tok == PRINT) continue; // o continuamos
                        value = expr( get: false);
                    }
                    input = tmp;      // Reasignamos la entrada
                    table = tmptable; // Reasignamos la tabla o mapeo de funciones
                    return value;
                }
                else
                {
                    return error( s: "numero invalido de argumentos");
                }
            }
            else
            {
                return error( s: "'(' esperado en llamada de funcion");
            }
        }
```

Calculadora.h

```
else // Definicion de la funcion
{
    Function f(string_value);
    if(get_token() == LP)   // Se Obtienen los argumentos
    {
        while(get_token() != RP && curr_tok != END)
        {
            switch(curr_tok)
            {
                case NAME:  // El nombre
                    f.arg_table[string_value];
                    break;
                case COMMA: // El separador
                    continue;
                    break;
                default:    // Si se llega aca, es porque se produjo un error
                    return error( s: "parametro de funcion invalido");
            }
        }
        f.arg_count = f.arg_table.size();
        // Se lee el cuerpo de la funcion
        if(get_token() == LCB)
        {
            f.body = string_value;
        }
        else
        {
            return error( s: "funcion indefinida; definicion faltante");
        }

        // Leida la funcion, se guarda en la tabla de símbolos:
        func_table[f.name] = f;
        return 0;
    }
    else // Error
    {
        return error( s: "'(' esperado en la definicion de la funcion");
    }
```

Calculadora.h

```
                return 0;
            }
        }
        case SUB: // Resta
            return -prim( get: true);
        case LP: // Caso de ausencia de elementos de cierre, como parentesis
        {
            double e = expr( get: true);
            if(curr_tok != RP)
                return error( s: "')' esperado, pero no encontrado");
            get_token();
            return e;
        }
        default: // En el caso de que no existe el elemento de apertura,
            // estaria por ejemplo: el parentesis de cierre, pero no el de apertura
            return error( s: "primario esperado");
    }
}
```

Calculadora.h

```
bool funcExists(const string& name)
{
    return func_table.count(name) ? true : false;
}
```

Verifica si una función existe

Main.cpp

Función principal, la cual puede recibir argumentos

Casos en los argumentos pasados a la función principal, por línea de comandos. Recuérdese que por línea de comandos, el nombre del ejecutable se considera el primer argumento.

```
int main(int argc, char* argv[])
{
    switch(argc)
    {
        case 1:
            input = &cin;
            break;
        case 2:
        {
            istringstream ss(argv[1]);
            input = &ss;
            break;
        }
        default:
            error( s: "Demasiados argumentos");
            return 1;
    }
```

Este primer caso, refiere a que no se recibieron argumentos en la línea de comandos, por lo que deben ser leídos luego de la ejecución del ejecutable del programa, esta es la opción más recomendada.

Se controla el caso en el que se exceda la cantidad de parámetros pasados por líneas de comandos.

Main.cpp

```
table = &globals;
```

Configura la tabla de símbolos para que apunte a los globales. Ambos definidos en el archivo de cabecera .h

```
table->operator[]( k: "pi") = 3.14159265358979323385;
table->operator[]( k: "e") = 2.7182818284590452354;
```

Se establecen constantes predefinidas, en el caso de requerirse, considerando que es una calculadora.

Main.cpp

Se realiza la lectura de entradas, el proceso de la calculadora.

```
    while(*input)
    {
        get_token();                    // Leemos un TOKEN, es decir entradas asociadas a la enumeracion
        if(curr_tok == END) break;      // El enum END, termina la ejecucion
        if(curr_tok == PRINT) continue; // El enum PRINT, continua la ejecucion

        cout << expr( get: false) << endl;
    }

    cout << endl; // Mostramos una linea o nueva linea

    return no_of_errors; // Retornamos el error asociado a la ejecucion del programa, en caso de existir
}
```

Conclusión

- Al finalizar el reconocimiento del código y su importancia en la implementación, pude ampliar mis conocimientos y relacionar el funcionamiento de una calculadora con temas vistos en otros cursos como por ejemplo: teoría de la computación.

  En el curso mencionado se vio la definición de un autómata, el cual es un modelo matemático para una máquina de estado finito, la cual lo comparé con la calculadora. Por lo que, estas dos recibirán una serie de símbolos y esta va a tener que seguir todas las ramas posibles hasta encontrar la respuesta correcta, es decir un estado de aceptación. Por ende, devolver un valor correspondiente.


¡¡GRACIAS!!