Polynomials - C++ project

About the project

This project is a program that processes a polynomial given by the user and returns a result in a proper form depending on a chosen action.

Used technology

The project was entirely written using C++ language in CodeBlocks 17.12 (with GNU GCC compiler).
Among the technologies that were used were:

- regular expression, used in parser
- classes
- derived methods
- virtual methods
- lists

How to use

Input data should be given in form like the example: x^12+13x^7-12.
Identical powers do not need to be grouped.
In next step the user is asked to choose from the list:

add - adds two polynomials
subtract - subtracts polynomial 1 from polynomial 2
scalar - performs scalar multiplication of chosen polynomial
mult - performs multiplication for given polynomials
m_find - find result of multiplication by substituting x
x_find - find result of chosen polynomial by substituting x
exit - exits program

After evaluation, the user is asked to choose whether they want to perform another operation on given inputs or exit the program.

Examples of running application

Menu of actions:

```
Enter the first polynomial:
x^6+2x^4-3x^5+9
Enter the second polynomial:
-2x^6-8x^2+3-9x
What operation do you choose?
list of possible operations:
        add - adds two polynomials
        subtract - subtracts polynomial 1 from polynomial 2
        scalar - performs scalar multiplication of chosen polynomial
        mult - performs multiplication for given polynomials
        m_find - find result of multiplication by substituting x
        x_find - find result of chosen polynomial by substituting x
        exit - exits program
```

After performing an operation:

```
Enter the first polynomial:
x^6+2x^4-3x^5+9
Enter the second polynomial:
-2x^6-8x^2+3-9x
What operation do you choose?
list of possible operations:
        add - adds two polynomials
        subtract - subtracts polynomial 1 from polynomial 2
        scalar - performs scalar multiplication of chosen polynomial
        mult - performs multiplication for given polynomials
        m_find - find result of multiplication by substituting x
        x_find - find result of chosen polynomial by substituting x
        exit - exits program

subtract
3x^6-3x^5+2x^4+8x^2+9x^1+6
Once again? Y/N
```

Another operation on the same examples:

```
add
-1x^6-3x^5+2x^4-8x^2-9x+12
Once again? Y/N
y
......................
What operation do you choose?
list of possible operations:
        add - adds two polynomials
        subtract - subtracts polynomial 1 from polynomial 2
        scalar - performs scalar multiplication of chosen polynomial
        mult - performs multiplication for given polynomials
        m_find - find result of multiplication by substituting x
        x_find - find result of chosen polynomial by substituting x
        exit - exits program

mult
-2x^12+6x^11-4x^10-8x^8+15x^7-4x^6-27x^5+6x^4-72x^2-81x+27
Once again? Y/N
n
......................
Bye bye!
```

About the code

There is short table explaining a few fragments of the code.

Some elements had been replaced with *(...) //explanation* for better readability.

name	type	functionality, explanation	fragment of code

Polynomial_input	class	class contains all the methods that allow further usage in functions and other classes; one of the most important classes	void Polynomial_input::get_input() {
			int max_power=0;
			string input;
			cin>>input;
			
			list<Polynomial_part> list_unsorted = parse_input(input);
			for (list<Polynomial_part>::iterator i = list_unsorted.begin(); i!=list_unsorted.end();i++){
			if (max_power<(*i).get_power()) max_power = (*i).get_power();
			}
			
			this->polynomial = list_unsorted;
			}
			
			list<Polynomial_part> Polynomial_input::get_polynomial()
			{
			return this->polynomial;
			}
			
			void Polynomial_input::print() {
			for (list<Polynomial_part>::iterator i = this->polynomial.begin(); i!=this->polynomial.end();i++){
			(*i).print();
			}
			
			cout << endl;
			}

parse_input	function (returning list of type Polynomial_part)	function gets string as an input, which gets parsed using regular expressions	list<Polynomial_part> parse_input(string input) {
			regex parts_regex("\\-?(?:\\d+x?|x)(?:\\^\\-?\\d+)?");
			auto expression_begin = sregex_iterator(input.begin(), input.end(), parts_regex);
			auto expression_end = sregex_iterator();
			
			list<Polynomial_part> parts;
			
			for (sregex_iterator i = expression_begin; i != expression_end; ++i) {
			string expression = (*i).str();
			bool is_constant = expression.find("x") == string::npos;
			
			regex inner_parts_regex("(\\-?\\d+|\\-|^)(\\-?\\d*)");
			auto number_begin = sregex_iterator(expression.begin(), expression.end(), inner_parts_regex);
			auto number_end = sregex_iterator();

			for (sregex_iterator i = number_begin; i != number_end; ++i)
			{
			double coeff;
			if ((*i).str() == "-") coeff = -1;
			else if ((*i).str() == "") coeff = 1;
			else coeff = stod((*i).str());
			
			if (!is_constant) ++i;
			int parsed_power = i == number_end
			? 1
			: stoi((*i).str());
			
			int power = is_constant ? 0 : parsed_power;
			
			Polynomial_part polynom_into_list(coeff,power);
			parts.push_back(polynom_into_list);
			
			if (i == number_end) break;
			}
			}
			
			return parts;
			}

Polynomial_adder	class	one of few classes performing mathematical operations; inherits from virtual classes Operations and Ordering	list<Polynomial_part> Polynomial_adder::internal_add (list<Polynomial_part> first_polynomial, list<Polynomial_part> second_polynomial) {
			list<Polynomial_part> result_list;
			
			(...)// part where program goes through two lists created previously from two input polynomials, adds/subtracts those with identical powers etc.
			
			return result_list;
			}
			
			Polynomial_adder() {};
			list<Polynomial_part> operate(list<Polynomial_part> list1, list<Polynomial_part> list2) {
			list<Polynomial_part> res = this->internal_add(list1, list2);
			return order(res);
			};
			};

switcharoo	function	function contains several components that are directly related to user's interaction with a program; collects user's choices and passes them to specific functions	int switcharoo(){
			double a;
			Polynomial_input input;
			Polynomial_input input2;
			
			cout<<"Enter the first polynomial: "<<endl;
			input.get_input();
			
			cout<<"Enter the second polynomial: "<<endl;
			input2.get_input();
			
			Polynomial_adder adder;
			Polynomial_subtractor subtractor;
			Polynomial_scalar_multiplication scalar_mult;
			Polynomial_multiplication mult;
			
			(...) // map
			
			once_again:
			
			string c;

			cout<<"What operation do you choose?"<<endl;
			
			(...) // here's cout with all the menu, followed by switch
			
			char co;
			cout<<endl<<"Once again? Y/N"<<endl;
			cin>>co;
			cout<<"....."<<endl;
			if (co=='Y' || co == 'y') goto once_again;
			if (co=='N' || co == 'n') {
			cout<<"Bye bye!"<<endl;
			Sleep(5000);
			return 0;
			}
			}

options	map	map contained in function switcharoo; takes "normal" commands and "links" them to their number equivalent, everything for better user experience as words feel more natural as commands than numbers	map <string, int> options;
			options["add"] = 1;
			options["subtract"] = 2;
			options["scalar"] = 3;
			options["mult"] = 4;
			options["m_find"] = 5;
			options["x_find"] = 6;
			options["exit"] = 7;

	switch	user chooses which operation should be performed, after proper command (and optionally some additional questions) they get an answer; every case leads to different function or method of class	switch (options[c]){
			case 1: {
			list<Polynomial_part>zmienna = adder.operate(input.get_polynomial(), input2.get_polynomial());
			print(zmienna);
			}break;
			
			case 2: {
			list<Polynomial_part>zmienna = subtractor.operate(input.get_polynomial(), input2.get_polynomial());
			print(zmienna);
			}break;
			
			case 3: {
			int a;
			double scalar;
			cout<<"What scalar?"<<endl;
			cin>>scalar;
			cout<<"which should we check? 1 or 2"<<endl;
			do{
			cout<<"Write number: "<<endl;
			cin>>a;
			}while(!(a==1 || a==2));
			if (a == 1) {
			list<Polynomial_part>zmienna = scalar_mult.operate(input.get_polynomial(), scalar);
			print(zmienna);
			}

```
            if (a == 2) {
            list<Polynomial_part>zmienna =
scalar_mult.operate(input2.get_polynomial(), scalar);
            print(zmienna);
            }

            }break;

            case 4: {
            list<Polynomial_part>zmienna =
mult.operate(input.get_polynomial(), input2.get_polynomial());
            print(zmienna);
            }break;

            case 5: {
            list<Polynomial_part>zmienna =
mult.operate(input.get_polynomial(), input2.get_polynomial());
            check_for_given_x(zmienna);
            }break;

            case 6: {
            int a;
            cout<<"which should we check? 1 or 2"<<endl;
            do{
            cout<<"Write number: "<<endl;
            cin>>a;
            }while(!(a==1 || a==2));
            if (a == 1) check_for_given_x(input.get_polynomial());
            if (a == 2) check_for_given_x(input2.get_polynomial());
            }break;

            case 7: {
            cout<<"Bye bye!"<<endl;
            return 0;
            }break;

}
```

Summary

The most complicated part of whole process was setting a proper regexes to “catch” important pieces of information from input, so they could be used in further operations.