Polynomial and Galois Field Library (PGFL)

Manual Version: 0.1

3rd July 2019

Erkan AFACAN
Gazi University
Faculty of Engineering
Department of Electrical-Electronics Engineering
Maltepe/Ankara, Turkey
e-mail: afacane@gmail.com

The aim of these package is to maintain a compact standalone C++ library for polynomial operations over Galois Fields. There are a couple of packages available which can be used for polynomial operations, but they are parts of bigger libraries and it is not easy to use them.

The package is developed as a part of a Tubitak project and is maintained as is.

This package is confirmed under Windows using CodeBlocks.

During the compilation of C++ programs C++11 option must be used.

Introduction

In this section, the class "Polynomial" and polynomial operations are introduced.

Polynomial Representation

First of all, we define a polynomial a as follows:

```
Polynomial a;
```

This means that a is of class "Polynomial".

For example, we can represent the polynomial $3x^2 + 2x + 1$ as follows:

```
a=Polynomial({1,2,3}); // $3 x^2 + 2 x + 1$
std::cout << "a: " << a << std::endl;
```

Please note that the elements of a polynomial are represented beginning from the smallest power, i.e. 0.

We can represent the polynomial x^2 as follows:

```
a=Polynomial({0,0,1}); // $x^2$
std::cout << "a: " << a << std::endl;
```

An alternative representation is

```
a=(Term(1, 2)); // $x^2$
std::cout << "a: " << a << std::endl;
```

Now we can define "0" polynomial as follows:

```
a=Polynomial({0}); // 0
std::cout << "a:␣" << a << std::endl;
```

Note that although a is “0”, it is polynomial “0”, not the number “0”.

In a similar way we can define polynomial “1” as

```
a=Polynomial({1}); // 1
std::cout << "a:␣" << a << std::endl;
```

Unfortunately, the following structure does not work due to the implementation of the class Polynomial.

```
// Does not work
a=Polynomial({0,1}); // $x$
std::cout << "a:␣" << a << std::endl;
```

So for the polynomial x we use:

```
a=(Term(1, 1)); // $x$
std::cout << "a:␣" << a << std::endl;
```

Also, the following structure for $x+1$ does not work

```
// Does not work
a=Polynomial({1,1}); // $x+1$
std::cout << "a:␣" << a << std::endl;
```

In a similar way, for $x+1$ we use:

```
a=Polynomial({1})+Term(1,1); // $x+1$
std::cout << "a:␣" << a << std::endl;
```

But for x^2+x+1 we can write:

```
a=Polynomial({1,1,1}); // $x^2+x+1$
std::cout << "a:␣" << a << std::endl;
```

A somewhat more complex polynomial:

```
a=Polynomial({1,0,0,2,0,3}); // $3 x^5 + 2 x^3 + 1$
std::cout << "a:␣" << a << std::endl;
```

a is the polynomial $3x^5+2x^3+1$.

We can represent the polynomial $5x^3$ as follows:

```
a=(Term(5, 3)); // 5 x^3
std::cout << "a:␣" << a << std::endl;
```

or

```
a=Polynomial({0,0,0,5}); // 5 x^3
std::cout << "a:␣" << a << std::endl;
```

The first representation is more compact for a single term.

Polynomial Addition and Subtraction

We define three Polynomials a, b, c:

```
Polynomial a, b, c;
```

Let's assume that a is the polynomial $x + 1$ and b is the polynomial $x^2 + 1$. Thus:

```
a=Polynomial({1})+Term(1,1); // $x+1$
std::cout << "a:␣" << a << std::endl;

b=Polynomial({1,0,1}); // $x^2+1$
std::cout << "b:␣" << b << std::endl;
```

Now let's add a to b and obtain the Polynomial c.

```
c=a+b;
std::cout << "(" << a << ")" << "␣+␣"
<< "(" << b << ")" << "␣=␣" << c << std::endl;
```

c is equal to $x^2 + x + 2$.

Now subtract b from a and obtain the Polynomial c.

```
c=a-b;
std::cout << "(" << a << ")" << "␣-␣"
<< "(" << b << ")" << "␣=␣" << c << std::endl;
```

c is equal to $-x^2 + x$.

Too simple, isn't it? But what about multiplication?

Polynomial Multiplication

Define a and b as before.

```
a=Polynomial({1})+Term(1,1); // $x+1$
std::cout << "a:␣" << a << std::endl;

b=Polynomial({1,0,1}); // $x^2+1$
std::cout << "b:␣" << b << std::endl;
```

Now multiply them as follows:

```
c=a*b;
std::cout << "(" << a << ")" << "␣*␣"
<< "(" << b << ")" << "␣=␣" << c << std::endl;
```

c is equal to $x^3 + x^2 + x + 1$. Not so easy!

We can obtain the degree of c:

```
n=c.degree();
std::cout << "n:␣" << n << std::endl;
```

The degree of c is $n = 3$. There is also a size() function but it is not always equal to degree(). For example let $a = x^5 + x^3 + x + 1$ and $b = x^5 - 2$. $c = a - b$ is $x^3 + x + 3$. The degree of c is 3, but the size of c is 6. So always try to use degree().

```
a=Polynomial({1,1,0,1,0,1}); // $x^5 + x^3 + x + 1$
std::cout << "a:␣" << a << std::endl;

b=Polynomial({-2,0,0,0,0,1}); // $x^5-2$
std::cout << "b:␣" << b << std::endl;

c=a-b;
std::cout << "(" << a << ")" << "␣-␣"
<< "(" << b << ")" << "␣=␣" << c << std::endl;

n=c.degree();
std::cout << "n:␣" << n << std::endl;

m=c.size();
std::cout << "m:␣" << m << std::endl;
```

Polynomial Division

Assume that we have two Polynomials $g = x^4 - 7x^3 + 5x^2 + 4$ and $f = x^2 + 3$. Now we want to divide g to f and to find the quotient q and remainder r, such that $g = q * f + r$.

Below we give the complete C++ code. Note that “ToF” is the file name and all the output is written into this file instead of console.

```
#include <iostream>
#include <vector>
#include <string>
#include <sstream>
#include <fstream>
#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#include "poly.h"

std::ofstream ToF;

int main ()
{
    Polynomial g,f,r,q;

    ToF.open("poly_sample.txt");

    g=Polynomial({4,0,5,-7,1});
    // $g=x^4 + -7x^3 + 5x^2 + 4$ // g = q*f + r
    f=Polynomial({3,0,1}); // $f= x^2 + 3$
    r=deconv(g,f);
    q=deconv_q(g,f);

    ToF << "g:␣" << g << std::endl;
    ToF << "q:␣" << q << std::endl;
    ToF << "f:␣" << f << std::endl;
    ToF << "r:␣" << r << std::endl;
    ToF << "g-(q*f+r):␣" << g-(q*f+r) << std::endl;

    ToF.close();
    return(0);
} // of main()
```

The code gives:

```
g: x^4 + -7x^3 + 5x^2 + 4
q: x^2 + -7x + 2
f: x^2 + 3
r: 21x + -2
g-(q*f+r): 0
```

By pencil and paper you can verify that:

$$x^4 - 7x^3 + 5x^2 + 4 = (x^2 - 7x + 2) * (x^2 + 3) + 21x - 2$$

From the code listing

```
r=deconv(g,f);
q=deconv_q(g,f);
```

we understand that the function `deconv(g,f)` gives the quotient polynomial and the function `deconv_q(g,f)` gives the remainder polynomial.

Polynomial Operations over Galois Field $GF(p)$

A Galois field is a finite field having p elements, where p is prime. Galois fields are generally shown as $GF(p)$. Note that $GF(p) = \mathbb{Z}_p$, where $\mathbb{Z}_p$ is the integers $\{0, 1, \ldots, p-1\}$.

Now we can, as well, add, subtract, multiply and divide (with remainder) polynomials over $GF(p)$. For addition we use the function `poly_add`, for subraction `poly_sub`, for multiplication `poly_mul`. To find the remainder we use the function `gfdeconv`, to find the quotient we use the `gfdeconv_q`, similar to usual polynomial operations. The prefix "gf" shows that these operations are over $GF(p)$.

Similar to given in Section "Polynomial Division", assume that we have two Polynomials $g = 3x^4 + 5x + 2$ and $f = 2x^3 + x^2 + 5$. Now we want to divide g to f and to find the quotient q and remainder r, such that $g = q * f + r$ in Galois field $GF(11)$. The complete C++ code is below.

```
#include <iostream>
#include <vector>
#include <string>
#include <sstream>
#include <fstream>
```

```
#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#include "poly.h"

int main ()
{
    std::ofstream ToF;

    int p;
    Polynomial g,f,r,q;

    ToF.open("gfpoly1.txt");

    p=11; /// Galois field GF(11)

    g=Polynomial({2,5,0,0,3});
/// g = q*f + r /// g = 3 x^4 + 5x + 2
    f=Polynomial({5,0,1,2}); /// f = 2x^3+ x^2 + 5
    r=gfdeconv(g,f,p);
    q=gfdeconv_q(g,f,p);

    ToF << "g:␣" << g << std::endl;
    ToF << "q:␣" << q << std::endl;
    ToF << "f:␣" << f << std::endl;
    ToF << "r:␣" << r << std::endl;
    ToF << "g-(q*f+r):␣"
<< poly_sub(g, poly_add( poly_mul(q,f,p),r,p ), p)
<< std::endl;

    ToF.close();
    return(0);
} /// of main()
```

This code gives:

```
g: 3x^4 + 5x + 2
q: 7x + 2
f: 2x^3 + x^2 + 5
r: 9x^2 + 3x + 3
g-(q*f+r): 0
```

Creation of the Galois Field $GF(p)$

To be continued...

Creation of the Galois Field $GF(p^n)$

To be continued...