Lab4

Sheharyar Alam Khan

March 25, 2019

Contents

1 cli.cpp

```
#include "main.hpp"

using namespace std;
```

Since I failed to make a functioning GUI using fltk cairo. I made a CLI in hopes of getting some pointson the lab. I just wanted to demonstrate that everything except for the data transfer between the guiand the callbacks works.

```
debian@debian:~/Labs/Lab4/Lab4$ ./cli
How many terms in Polynomial 1?
2
coefficient: 5
exponent: 2

coefficient: 12
exponent: 2

How many terms in Polynomial 2?
3
coefficient: 1
exponent: 2

coefficient: 2
exponent: 1

coefficient: 4
exponent: 0


p1 = +17x^2
p2 = +1x^2+2x+4
p1 + p2 = +35x^2+2x+4
p1 * p2 = +17x^4+34x^3+68x^2
debian@debian:~/Labs/Lab4/Lab4$
```

```
int main(void){
 Polynomial *p1 = new Polynomial;
 Polynomial *p2 = new Polynomial;
```

```
 int n1, n2, c, e;
 cout << "How many terms in Polynomial 1?" <<endl;
 cin >> n1;
 for(int i = 0; i< n1; i++)
 {
         cout << "coefficient: ";
         cin >> c;
         cout << "exponent: ";
         cin >> e;
         p1->addTerm(c,e);
         cout << endl;
 }
 cout << "How many terms in Polynomial 2?" <<endl;
 cin >> n2;
 for(int i = 0; i< n2; i++)
 {
         cout << "coefficient: ";
         cin >> c;
         cout << "exponent: ";
         cin >> e;
         p2->addTerm(c,e);
         cout << endl;
 }
  cout << endl;
  cout << "p1 = " << p1->print() << endl;
  cout << "p2 = " << p2->print() << endl;
  Polynomial result = *p1 + p2;
  cout << "p1 + p2 = " << result.print() << endl;
  result = *p1 * p2;
  cout << "p1 * p2 = " << result.print() << endl;
  return 0;
}
```

2 gui.h

```
// generated by Fast Light User Interface Designer (fluid) version 1.0304

#ifndef gui_h
#define gui_h
#include <FL/Fl.H>
#include <iostream>
#include <FL/Fl_Window.H>
#include <FL/Fl_Window.H>
#include <FL/Fl_Box.H>
#include <FL/Fl_Input.H>
#include <FL/Fl_Button.H>
extern Fl_Button *addButton;
#include <FL/Fl_Text_Display.H>
extern Fl_Text_Display *resultTextDisplay;
Fl_Window* make_window();
static void cb_addButton(Fl_Button*, void*);
static void cb_Multiply(Fl_Button*, void*);
#endif
```

3 gui.cpp

```
// generated by Fast Light User Interface Designer (fluid) version 1.0304
#include "main.hpp"
#include "gui.h"

Here I'm declaring global variables so i can transfer data throughout the program without
    returninganything.

int coeff; int exp; string res = ""; int showResult = 0;
Polynomial *p1 = new Polynomial;
Polynomial *p2 = new Polynomial;
Polynomial result;


    This is the callback function for the coefficiant input box
static void cb_coefficiant(Fl_Input*, long v) {
        cout << "in coefficient input box"<<endl;
  coeff = (int)v;
 cout << coeff <<endl;
}

    This is the callback function for the exponent input box
static void cb_exponent(Fl_Input*, long v) {
        cout << "in exponent input box"<<endl;
  exp = (int)v;
 cout << exp <<endl;
}

    This is the callback function for the Add to Polynomial 1 button.
static void cb_Add(Fl_Button*, void*) {
  p1->addTerm(coeff,exp);
```

```
        cout << "in Add to Polynomial 1" <<endl;
        cout << p1->print()<<endl;
}

    This is the callback function for the Add to Polynomial 2 button.
static void cb_Add1(Fl_Button*, void*) {
  p2->addTerm(coeff,exp);
        cout << "in Add to Polynomial 2" <<endl;
        cout << p1->print()<<endl;
}
makes a button pointer to null.
Fl_Button *addButton=(Fl_Button *)0;


makes a null text display pointer
Fl_Text_Display *resultTextDisplay=(Fl_Text_Display *)0;

This function makes a window, makes widgets and addsit to the window, and then returns a
    pointer to the window.  the window contains all the widgets and buttons.
Fl_Window* make_window() {
  Fl_Window* w;
  { Fl_Window* o = new Fl_Window(305, 605);
    w = o; if (w) { empty }
    o->label("Polynmial Addition and Multiplication");
    o->color(FL_BACKGROUND2_COLOR);
    { Fl_Box* o = new Fl_Box(130, 198, 45, 61, "x");
      o->color(FL_BACKGROUND2_COLOR);
      o->labeltype(FL_EMBOSSED_LABEL);
      o->labelfont(10);
      o->labelsize(65);
      o->align(Fl_Align(FL_ALIGN_TEXT_OVER_IMAGE));
    } // Fl_Box* o
```

```
{ resultTextDisplay = new Fl_Text_Display(15, 65, 270, 45, "result: ");
  resultTextDisplay->textfont(10);
  resultTextDisplay->when(FL_WHEN_NEVER);
} // Fl_Text_Display* resultTextDisplay
{ Fl_Input* o = new Fl_Input(25, 181, 110, 83, "coefficiant");
  o->box(FL_BORDER_BOX);
  o->labeltype(FL_EMBOSSED_LABEL);
  o->labelfont(9);
  o->labelsize(72);
  o->textsize(72);
  o->callback((Fl_Callback*)cb_coefficiant, (void*)0);
  o->align(Fl_Align(36|FL_ALIGN_INSIDE));
  o->when(FL_WHEN_CHANGED);
} // Fl_Input* o
{ Fl_Input* o = new Fl_Input(170, 173, 50, 42, "exponent");
  o->box(FL_BORDER_BOX);
  o->labelsize(32);
  o->textsize(32);
  o->callback((Fl_Callback*)cb_exponent, (void*)(0));
  o->align(Fl_Align(FL_ALIGN_LEFT|FL_ALIGN_INSIDE));
  o->when(FL_WHEN_CHANGED);
} // Fl_Input* o
{ Fl_Button* o = new Fl_Button(65, 335, 180, 55, "Add to Polynomial 1");
  o->down_box(FL_DOWN_FRAME);
  o->color(FL_BLUE);
  o->selection_color((Fl_Color)5);
  o->labelfont(1);
  o->labelcolor(FL_BACKGROUND2_COLOR);
  o->callback((Fl_Callback*)cb_Add);
} // Fl_Button* o
{ Fl_Button* o = new Fl_Button(65, 395, 180, 55, "Add to Polynomial 2");
  o->color(FL_BLUE);
```

```
      o->selection_color((Fl_Color)5);
      o->labelfont(1);
      o->labelcolor(FL_BACKGROUND2_COLOR);
      o->callback((Fl_Callback*)cb_Add1);
    } // Fl_Button* o
    { addButton = new Fl_Button(65, 455, 90, 60, "Add");
      addButton->color((Fl_Color)208);
      addButton->labelfont(1);
      addButton->labelsize(20);
      addButton->labelcolor(FL_BACKGROUND2_COLOR);
      addButton->callback((Fl_Callback*)cb_addButton);
    } // Fl_Button* addButton
    { Fl_Button* o = new Fl_Button(160, 455, 85, 60, "Multiply");
      o->color((Fl_Color)71);
      o->selection_color(FL_BACKGROUND2_COLOR);
      o->labelfont(1);
      o->labelsize(16);
      o->labelcolor(FL_BACKGROUND2_COLOR);
      o->callback((Fl_Callback*)cb_Multiply);
    } // Fl_Button* o
    o->end();
  } // Fl_Window* o
  return w;
}

This is the callback function to the add button
static void cb_addButton(Fl_Button*, void*) {
        cout << "in add button" <<endl;
        result = *p1 + p2;
        res = result.print() + "\n";

}
```

```
This is the callback function to the add button
static void cb_Multiply(Fl_Button*, void*) {
        cout << "in multiply" <<endl;
        result = *p1 * p2;
        showResult = 1;
}
```

4 main.cpp

```
#include "main.hpp"
#include "gui.h"

using namespace std;
```

This program was supposed use the cairo 2d graphics library within an FLTK window to display apolynomial:
e.g.

$$2x^2 + 3x + 1$$

.
However,I couldn't get the gui to transfer any data to the program. whatever I type in the input feildsis not being transfered over to the variables. the coefficient and exponent input values appear totake input but that input isn't stored anywhere. the value for the input continue to be 0 on thebackend and I couldn't quite figure out why.

```
debian@debian:~/Labs/Lab4$
debian@debian:~/Labs/Lab4/
in coefficient input box
0
in coefficient input box
0
in exponent input box
0
in Add to Polynomial 1
+0
in Add to Polynomial 2
+0
in add button
in multiply
```

col: 4 sel: 0 INS TAB

Polynmial Addition and Multiplication ×

result:

$24x^{2}$

Add to Polynomial 1

Add to Polynomial 2

Add Multiply

```
int main()
{
        Fl_Window *w = make_window();
        w->show();
        return (Fl::run());
}
```

5 poly.cpp

```
#include "main.hpp"
This file contains all the function definitions for the Polynomial and Term classes.

    This function is an overloaded constructor.  it sets the Term object's exponent and
    coefficient to the ones provided in the parameters respectedly.
Term::Term(int coeff, int exp){
    this->exp = exp;
    this->coeff = coeff;
}

    The addTerm function creates a new term during runtime using the overloaded operator
    explained above.
    1.  It then checks if the list is empty.  if its it adds the term to the start of the
    list.
    2.  if it isn't it checks if the exponent of the new term is greater than or equal to
    the exponent of the first term.  if they are equal it simply adds the coefficients.  if
    its greater it adds the term to the start of the list.
    3.  If that isn't true either then it checks if the exponent of the new term is less
    than or equal to the exponent of the last term.  if they are equal it simply adds the
    coefficients.  if its greater it adds the term to the end of the list.
    4.  if that isn't true either then it has no choise but to traverse through the list
    and add the term in its place.
void Polynomial::addTerm(int coeff, int exp){
    Term *newterm = new Term(coeff, exp);
    if (this->h == nullptr)
        addToStart(newterm);
    else if (exp >= this->h->exp)    {
        if (exp > this->h->exp)
            addToStart(newterm);
        else
```

```
            this->h->coeff += coeff;
    }
    else if (exp <= this->t->exp){
        if (exp < this->t->exp)
            addToEnd(newterm);
        else
            this->t->coeff += coeff;
    }
    else
        addInPlace(newterm);
}
```

This function adds the term to the start of the Polynomial.

```
void Polynomial::addToStart(Term *newterm){
        if (this->h == nullptr){
                this->h = newterm;
                this->t = newterm;
        }else{
                newterm->nxt = this->h;
                this->h = newterm;
        }
}
```

This function adds the term to the end of the list. It uses advantage of the fact that the Polynomial also tracks the tail pointer and uses that to avoid having to traverse through the entire list. it is very efficient in this way.

```
void Polynomial::addToEnd(Term *newterm){
    this->t->nxt = newterm;
    this->t = newterm;
}
```

The add in place function traverses through the list and add the new term in place. it

keeps traveling the list until it reaches the Node whose expont is less than or equal to the exponent of the newterm. if they are equal it adds the coefficiants. if they aren't it adds the new node after it.

```
void Polynomial::addInPlace(Term * newterm){
    Term *trav = this->h;
    while (trav != nullptr && trav->exp >= newterm->exp){
        trav = trav->nxt;
    }
    if(trav != nullptr){
      if (trav->exp == newterm->exp)
          trav->coeff += newterm->coeff;
      else{
          Term *tmp = trav->nxt;
          trav->nxt = newterm;
          newterm->nxt = tmp;
      }
    }
}
```

This function overloads the + operator so that it may add polynomials. It takes in a pointer to a polynomial. (It could take in a normal polynomial but enjoy using the -> operator so I just used a pointer). It also returns a Polynomial.1. First it creates a Polynomial result and two Term pointer to traverse through the two Polynomials to be added.2. In a While loop it checks if t1 and t2 have the same exponents. if they do it uses the addTerm fuction create a Term and add it to result. if one of the lists reaches the end while the other doesn't it goes through the entire second list and adds the remaining terms to the new polynomial.

```
Polynomial Polynomial::operator+(Polynomial *right){
    Polynomial result;
    Term *t1 = this->h;
    Term *t2 = right->h;
    while (t1 != nullptr && t2 != nullptr){
```

```
        if (t1->exp == t2->exp)
            result.addTerm(t1->coeff + t2->coeff,t1->exp);
        else if (t2->exp > t1->exp){
            result.addTerm(t2->coeff,t2->exp);
            t2 = t2->nxt;}
        else{
            result.addTerm(t1->coeff,t1->exp);
            t1 = t1->nxt;}
        if (t1 == nullptr)
          while(t2 != nullptr){
            result.addTerm(t2->coeff,t2->exp);
            t2 = t2->nxt;}
        else if (t2 == nullptr)
          while(t1 != nullptr){
            result.addTerm(t1->coeff,t1->exp);
            t1 = t1->nxt;}
        else{
          if (t1->nxt == nullptr && t2->nxt != nullptr)
            t2 = t2->nxt;
          else if (t1->nxt != nullptr && t2->nxt == nullptr)
            t1 = t1->nxt;
          else{
            t1 = t1->nxt;
            t2 = t2->nxt;
        }}
    }
    return (result);
}
```

This function overloads the *operator so that it may add polynomials. It takes in a pointer to apolynomial.It also returns a Polynomial. it uses a nested while loop to multiply each of the terms inthe first list with each of therms in the second list and

```
    add the new term to the result Polynomial.
Polynomial Polynomial::operator*(Polynomial *right){
    Term *t1 = this->h;
    Term *t2 = right->h;
    Polynomial result;
    while (t1 != nullptr)
    {
        t2 = right->h;
        while (t2 != nullptr)
        {
            result.addTerm(t1->coeff * t2->coeff, t1->exp + t2->exp);
            t2 = t2->nxt;
        }
        t1 = t1->nxt;
    }
    return (result);
}
```

```
    This function makes an empty string and adds characters to it depending on the values
    of its exponent andcoefficent.  In the end it returns the string.
string Term::print(){
    string term = "";
    if(coeff < 0)
        term += "-";
    else
        term += "+";
    term += to_string(coeff);
    if(exp > 0)
        term += "x";
    if(exp > 1)
        term += "^" + to_string(exp);
    return term;
```

```
}


    This function prints out all the terms in the polynomial.  it makes an empty string and
    adds the strings returned by the Term::Print function above.  it also returns a string
    in the end.
string Polynomial::print(){
    string poly = "";
    if(this->h != nullptr){
        Term *trav = this->h;
        while(trav != nullptr){
          poly += trav->print();
          trav = trav->nxt;
        }
    }else{
        cout << "empty poly" << endl;
        }
    return poly;
}
```