ELEC 278

Tutorial Week 4

2022 Fall

Instructors:
Dr. Jianbing Ni & Dr. Mohammadali Hedayati

Tutorial TAs:
AmirHossein Sojoodi & Shayan Noei

Adapted from slides by
Bryony Schonewille & Shayan Noei

Outline

- Memory types
  - Static memory
    - Global memory (Data Segment)
  - Local memory (Stack Segment)
  - Dynamic memory (Heap)
- Linked Lists
  - Examples

Memory types

Pay attention to the lifetime of each memory types.

```
#define LENGTH 3

int global_array[LENGTH] = {1, 2, 3};

void main() {
  int local_array[LENGTH] = {4, 5, 6};

  int *dynamic_array;

  dynamic_array = (int *)malloc(sizeof(int) * LENGTH);

  free(dynamic_array);
}
```

Memory types

- Static memory
  - Global memory (Data Segment)
    - Allocated at the loading time (they exist even before any instruction of the program is executed)
  - Local memory (Stack Segment)
    - Any variable/array that is allocated within any function is stored on the "Stack".
    - They will be destroyed/cleared at the end of the function.
- Dynamic memory (Heap)
  - Any memory allocated with malloc
  - They will not be freed automatically unless passed to the free() function

Memory types

Let's see some examples.

```
int global_array[LENGTH] = {1, 2, 3};

void print_array(char *name, int *array, int length) {
  printf("%S: {");
  for (int i = 0; i < length; i++) {
    printf("%d", array[i]);
    if (i < length - 1) {
      printf(", ");
    }
  }
  printf("}\n");
}

void main() {
  int local_array[LENGTH] = {4, 5, 6};
  int *dynamic_array;
  dynamic_array = (int *)malloc(sizeof(int) * LENGTH);
  free(dynamic_array);
}
```

Problem – Project Statement

The goal of this project is to extend your knowledge of basic data structures. You will be given a list of functions for implementing a ***polynomial****.*

A polynomial is a sum of terms, where each term is of the form K x^n, where K is a numeric coefficient, n is the exponent and x is the independent variable.

$$f = \sum_{n} K\, x^n$$

For example: $6x^3 - 2x^2 + 4x^1 - 3x^0$

Problem – term structure


```
typedef struct polyterm_st polyterm, *p_polyterm;

struct polyterm_st {
  struct polyterm_st *ptnext;  // link to next item in list
  int exp;                     // Exponent value
  double factor;               // Factor value
};
```

So, the expression $\mathbf{6x^3 - 2x^2 + 4x^1 - 3x^0}$ could be held in 4 nodes:

(6,3) -> (-2,2) -> (4,1) -> (-3, 0)

Problem – *polynomial* structure

Polynomial is linked list of polynomial terms. Polynomial descriptor is pretty minimal.

```
typedef struct poly_st polynomial, *p_polynomial;

struct poly_st {
  polyterm *pt;  // pointer to list of terms
};
```

Problem – *create* polynomial

Write a function to create new polynomial.

```
polynomial *new_polynomial(void) {
  polynomial *ppnm = (polynomial *)malloc(sizeof(polynomial));
  if (ppnm != NULL) {
    ppnm->pt = NULL;
  }
  return ppnm;
}
```

Problem – *create* a polyterm

Write a function to create new polyterm.

```
polyterm *new_polyterm(int expon, double multiplier) {
  polyterm *ptm = (polyterm *)malloc(sizeof(polyterm));
  if (ptm != NULL) {
    ptm->ptnext = NULL;
    ptm->exp = expon;
    ptm->factor = multiplier;
  }
  return ptm;
}
```

Problem – power function

Write a power function for a Polyterm.

$2^3 = 8$

```
int power(int n, int exp) {
  int count;
  int rslt = 1;  // lets us handle 0 exponent
  if (exp < 0)
    rslt = -1;
  else
    for (count = 1; count <= exp; count++) rslt = rslt * n;
  return rslt;
}
```

Problem - polynomial evaluation (A)

Write a function that takes a linked list of Terms and performs the evaluation as follows:

```
set sum = 0;
for all the Terms in the linked list {
    compute value of this term: coeff * (x raised to
    the power exp) add this value to sum
}
return sum;
```

Problem - polynomial evaluation (B)


```
int evaluate(polyterm *pt, double x) {
  int sum = 0;
  while (pt != NULL) {
    sum = sum + (pt->factor) * power(x, pt->exp);
    pt = pt->ptnext;
  }
  return sum;
}
```

Problem - polynomial representation

Write a function to print a representation of the polynomial

```
void print(polyterm *pt) {
  int donefirst = 0;  // you'll see why in a moment
  while (pt != NULL) {
    char sign = (pt->factor < 0) ? '-' : '+';
    double abscoeff = abs(pt->factor);
    // print sign
    if (donefirst || sign == '-') {
      printf(" %c ", sign);
    }
    donefirst = 1;
    printf("%8.2lf X^%d ", abscoeff, pt->exp);
    pt = pt->ptnext;
  }
}
```

Problem - differentiating a polynomial

Write a function to differentiate a polynomial

```
void differentiate(polyterm *pt) {
  while (pt != NULL) {
    pt->factor = pt->factor * pt->exp;
    pt->exp--;
    if (pt->ptnext->exp == 0) {
      polyterm *ptmp = pt->ptnext;
      pt->ptnext = NULL;  // unlink it from polynomial
      free(ptmp);         // get rid of the Term
    }
    pt = pt->ptnext;
  }
}
```

Problem – set a coefficient (A)

Write a function to set a coefficient in a polynomial

```
void setcoefficient (polyterm *ppt, int i, double c) {
  polyterm pll = *ppt
  polyterm *pt = (polyterm) malloc (sizeof (polyterm));
  polyterm *ptmp;    // temporary Term pointer
  if (pt == NULL)  return 0;  // should report
  pt->coeff = c;
  pt->exp = i;
  pt->next = NULL;
  if (pll == NULL)  {
    pll = pt;
    *ppt = pll;
    return;
  }
```

Continued …

Problem – set a coefficient (B)


```
  ptmp = *pll;
  if (pt->exp > ptmp->exp) {
    // new node at front
    pt->next = ptmp;
    pll = pt;
    *ppt = pll;
    return;
  }
  while (ptmp->exp > pt->exp) {
    if (ptmp->next == NULL || pt->exp > ptmp->next->exp){
      pt->next = ptmp->next;
      ptmp->next = pt; // current next becomes new node
      return;
    }
    ptmp = ptmp->next;
  }
}
```

Problem – add polynomials (A)

Write a function to add two polynomials.

Example: $(3x^2+2) + (2x^2+4x+1)$

```
polynomial *addPolynomials(polynomial *pa, polynomial *pb) {
  polynomial *pnew;  // point to new polynomial

  if (pa == NULL || pa->pt == NULL || pb == NULL || pb->pt == NULL) {
    pnew = NULL;
  } else {
    polyterm *pta, *ptb;
    pnew = new_polynomial();
    pta = pa->pt;
    ptb = pb->pt;
```

Problem – add polynomials (B)


```
while (pta != NULL || ptb != NULL) {
      if ((ptb == NULL) || (pta->exp > ptb->exp)) {
        polyterm *p = new_polyterm(pta->exp, pta->factor);
        InsertPolyTerm(pnew, p);
        pta = pta->ptnext;
      }
      else if ((pta == NULL) || (ptb->exp > pta->exp)) {
        polyterm *p = new_polyterm(ptb->exp, ptb->factor);
        InsertPolyTerm(pnew, p);
        ptb = ptb->ptnext;
      }
      else if (pta->exp == ptb->exp) {
        polyterm *p = new_polyterm(pta->exp, pta->factor + ptb->factor);
        InsertPolyTerm(pnew, p);
        pta = pta->ptnext;
        ptb = ptb->ptnext;
      }
}
```