Обработка на полиноми

$$p(x) = p_{n-1}x^{n-1} + \ldots + p_1 x + p_0$$

Събиране на полиноми

$$r(x) = p(x) + q(x)$$

1. Представяне чрез масив от коефициентите –

- **предимство – директен достъп до всеки коефициент;**
- **недостатък – голямо количество памет (особено при много нулеви коефициенти).**

```
// Представяне чрез масив от коефициенти
int p[4] = {1, 2, 0, -3}, q[4] = {2, -1, 0, 0}, r[4];
int n = 4;
for (int i = n-1; i >= 0; i--)
  r[i] = p[i] + q[i];
```

2. Представяне чрез свързан списък от коефициентите – възлите се пазят в намаляващ ред на степента на полинома.

```
struct node                          // възел  cx^i
{
  int c;                             // коефициент
  int i;                             // степен
  struct node *next;
};
typedef struct node NODE;
typedef NODE *LINK;
struct list                          // списък
{
  LINK head;
};
typedef struct list LIST;
```

Операции

1. **Създаване на празен свързан списък**
2. **Включване на възел в подреден свързан списък**
3. **Събиране на полиноми**
4. **Печат на полином**

```
#define FAILURE 0                    // неуспешна операция
#define SUCCESS 1                    // успешна операция
```

1. Създаване на празен свързан списък

Алгоритъм
начало ← ПРАЗНО

head

```
void create (LIST *l)
{
        l->head = NULL;              // празен списък
}
```

2. Включване на възел в подреден свързан списък

Алгоритъм
отдели памет за нов възел
ако отделянето на памет не е успешно
** неуспешна операция**
в противен случай
** запази данните за коефициента и степента във възела**
** ако списъкът е празен**
**   маркирай новия възел като край на списъка**
**   насочи началото на списъка към новия възел**
** в противен случай**
**   ако степен(начало) < степен(новия възел) – новият възел се включва преди началото**
**   насочи следващата компонента на дадения възел към начало**
**   насочи начало към новия възел**
**   в противен случай**

```
    насочи временна връзка към начало
    докато следващата компонента(временната връзка) ≠
    ПРАЗНО и
    степен(временната връзка) > степен(новия възел)
    повтаряй
      придвижи временната връзка към следващия възел
    насочи следващата компонента(новия възел) към
    следващата компонента(временната връзка)
    насочи следващата компонента(временната връзка)
    към новия възел
върни начало
```

```
LIST insert (LIST l, int c, int i)
{
  LINK p, save;
  p = (LINK) malloc (sizeof(NODE));              // нов възел
  if (p == NULL)
    printf ("Няма достатъчно памет!\n");
  else
  {
    p->c = c;                                    // запазване на данните
    p->i = i;                                    // в новия възел
    if (l.head == NULL)                          // ако списъкът е празен
    {
        p->next = NULL;                          // новият възел става
        l.head = p;                              // първи в списъка
    }
    else
```

```
    {   if (l.head->i < p->i)        // ако мястото на новия възел
        {  p->next = l.head;         // е преди началото на списъка
           l.head = p;
        }
        else
        {   // инициализация на временна връзка
            save = l.head;
            // търсене на мястото на включване
            while (save->next != NULL && (save->next->i > p->i))
               save = save->next;
            // връзка между новия възел и предшественика му
            p->next = save->next;
            save->next = p;
        }
    }
  }
  return l;
}
```

3. Събиране на полиноми

Алгоритъм

```
създай празен полином r
насочи p към p1
насочи q към p2
докато p1 ≠ ПРАЗНО и p2 ≠ ПРАЗНО повтаряй
  ако степен(p) = степен(q)
    ако коефициент(p) + коефициент(q) ≠ 0
      включи в r възел(коефициент(p)+коефициент(q),
                       степен(p))
      придвижи p към следващия възел
      придвижи q към следващия възел
    в противен случай
      придвижи p към следващия възел
      придвижи q към следващия възел
  в противен случай
    ако степен(p) >степен(q)
      включи в r възел(коефициент(p), степен(p))
      придвижи p към следващия възел
```

```
    в противен случай
      включи в r възел(коефициент(q), степен(q))
      придвижи q към следващия възел
докато p ≠ ПРАЗНО повтаряй
  включи в r възел(коефициент(p), степен(p))
  придвижи p към следващия възел
докато q ≠ ПРАЗНО повтаряй
  включи в r възел(коефициент(q), степен(q))
  придвижи q към следващия възел
върни r
```

```
LIST add (LIST p1, LIST p2)
{
  LIST r;
  LINK p, q;
  r.head = NULL;
  p = p1.head;
  q = p2.head;
  while (p != NULL && q != NULL)
  {
    if (p->i == q->i)
    {
        if ((p->c + q->c) != 0)
        {
          r = insert (r, p->c+q->c, p->i);
          p = p->next;
          q = q->next;
        }
```

```
        else
        {
          p = p->next;
          q = q->next;
        }
    }
    else
        if (p->i > q->i)
        {
          r = insert (r, p->c, p->i);
          p = p->next;
        }
        else if (q->i > p->i)
        {
          r = insert (r, q->c, q->i);
          q = q->next;
        }
  }
```

```
  while (p != NULL)
  {
    r = insert (r, p->c, p->i);
    p = p->next;
  }
  while (q != NULL)
  {
    r = insert (r, q->c, q->i);
    q = q->next;
  }
  return r;
}
```

4. Печат на полином

Алгоритъм

насочи p **към** начало
ако p **= ПРАЗНО**
  отпечатай: Празен полином
в противен случай
  отпечатай: коефициент(p**)**степен(**p**)**
  придвижи p **към следващия възел**
  докато p ≠ **ПРАЗНО повтаряй**
    отпечатай: +|-|коефициент(p**)|** степен(**p**)**
    придвижи p **към следващия възел**
  премини на нов ред

```
void print (LIST l)
{
  LINK p;
  p = l.head;
  if (p == NULL)
    printf ("Празен полином.\n");
  else
  {
    printf ("%dx**%d", p->c, p->i);
    p = p->next;
    while (p != NULL)
    {
        printf ("%c%dx**%d", (p->c>0)?'+':'-', abs(p->c), p->i);
        p = p->next;
    }
    printf ("\n");
  }
}
```