Контрольная работа по дисциплине «Алгоритмы и структуры данных»

1. Выбрать тип списка по варианту.
2. Выбрать тип хранимых данных.
3. Сформировать тип звена списка для заданного типа списка.
4. Выбрать из имеющегося набора процедуру добавления звена для заданного типа списка.
5. Выбрать из имеющегося набора процедуру удаления звена для заданного типа списка.
6. Выбрать из имеющегося набора процедуру поиска звена для заданного типа списка.
7. Разработать процедуру просмотра списка.
8. Создать ведущее звено для заданного типа списка.
9. Выбрать из имеющегося набора процедуру сортировки массива по варианту.

Вариант выбирается по **двум последним** цифрам номера студенческого билета **AB.**

Хранящиеся в списке данные выбираются по сумме цифр **A** и **B** (см. таблицу), если **A**+**B** = 0, то выбирается вариант **10**:

1	Целые числа (**int**)	10	Знаковые целые числа (**signed int**)
2	Знаковые символы (**signed char**)	11	Беззнаковые целые числа (**unsigned int**)
3	Длинные целые числа (**long**)	12	Длинные беззнаковые целые числа
4	Действительные числа (**float**)	13	Действительные числа (**long float**)
5	Действительные числа (**long double**)	14	Знаковые символы (**signed char**)
6	Короткие целые числа (**short**)	15	Беззнаковые целые числа (**unsigned int**)
7	Действительные числа (**double**)	16	Действительные числа (**float**)
8	Символы (**char**)	17	Короткие целые числа (**short**)
9	Беззнаковые символы (**unsigned char**)	18	Действительные числа (**double**)

Тип списка определяется по **младшей цифре 4-ричного кода**, соответствующего **сумме цифр A и B**.

$(A+B)_4$	Тип списка
0	Односвязный кольцевой
1	Односвязный линейный
2	Двусвязный кольцевой
3	Двусвязный линейный

Соответствие между 10-тичным и 4-ричным кодами приведено в таблице

$(A+B)_{10}$	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18
$(A+B)_4$	**0**	**1**	**2**	**3**	1**0**	1**1**	1**2**	1**3**	2**0**	2**1**	2**2**	2**3**	3**0**	3**1**	3**2**	3**3**	4**0**	4**1**	4**2**

Метод сортировки определяется по **B** (последняя цифра номера), если **B** больше **4**, то по **B** – **5**:

B Метод сортировки

0 Отбор

1 Вставка

2 Пузырьковая

3 Быстрая сортировка

4 Метод Шелла

Показанные процедуры работы со списками и сортировки приведены только в качестве образца и могут не совпадать по именам типов, переменных и типам полей записей с выбранными студентом.

Процедуры добавления звена

```
void i1(Link* Pred, int data)
{
  Link* Loc = new Link;
  Loc->Value = data;
  Loc->next = Pred->next;
  Pred->next = Loc;
}

void i2(Link* Pred, int data)
{
  Link* Loc = new Link;
  Loc->Value = data;
  Loc->next = Pred->next;
  Loc->prev = Pred;
  Pred->next = Loc;
  if (Loc->next)
    Loc->next->prev = Loc;
}
```

```
void i3(Link* Pred, int data)
{
  Link* Loc = new Link2;
  Loc->Value = data;
  Loc->next = Pred->next;
  Loc->prev = Pred;
  Pred->next = Loc;
  Loc->next->prev = Loc;
}
```

Процедуры удаления звена

```
void d1(Link* Pred)
{
  Link* Loc;
  if (Pred->next)
  {
    Loc = Pred->next;
    Pred->next = Loc->next;
    delete Loc;
  }
}

void d2(Link* Pred)
{
  Link* Loc;
  Loc = Pred->next;
  Pred->next = Loc->next;
  delete Loc;
}
```

```
void d3(Link* Del)
{
  Del->prev->next = Del->next;
  if (Del->next)
    Del->next->prev = Del->prev;
  delete Del;
}

void d4(Link* Del)
{
  Del->prev->next = Del->next;
  Del->next->prev = Del->prev;
  delete Del;
}
```

Процедуры поиска звена

```
int p1(Link* Start, Link*& Find, Link*& Pred, int Key)
{
  Link* Cur = Start->next;
  Pred = Start;
  int Success = 0;
  while (Cur && !Success)
  {
    if (Cur->Value == Key)
    {
      Find = Cur;
      Success = 1;
      break;
    }
    Pred = Cur;
    Cur = Cur->next;
  }
  return Success;
}
```

```
int p2(Link* Start, Link*& Find, Link*& Pred, int Key)
{
  Link* Cur = Start->next;
  Pred = Start;
  int Success = 0;
  while (Cur != Start && !Success)
  {
    if (Cur->Value == Key)
    {
      Find = Cur;
      Success = 1;
      break;
    }
    Pred = Cur;
    Cur = Cur->next;
  }
  return Success;
}

int p3(Link* Start, Link*& Find, int Key)
{
  Link* Cur = Start->next;
  int Success = 0;
  while (Cur && !Success)
  {
    if (Cur->Value == Key)
    {
      Find = Cur;
      Success = 1;
      break;
    }
    Cur = Cur->next;
  }
```

```
 return Success;
}

int p4(Link* Start, Link*& Find, int Key)
{
 Link* Cur = Start->next;
 int Success = 0;
 while (Cur != Start && !Success)
 {
  if (Cur->Value == Key)
  {
   Find = Cur;
   Success = 1;
   break;
  }
  Cur = Cur->next;
 }
 return Success;
}
```

Процедуры сортировки

```
void s1(float* item, int n)
{
 int a,b;
 float buf;
 for (a = 1; a<n; ++a)
  for (b = n-1; b>=a; --b)
   if (item[b-1]>item[b])
   {
    buf = item[b-1];
    item[b-1] = item[b];
    item[b] = buf;
   }
}
```

```
void s2(float* item, int n)
{
  int a,b,c;
  float buf;
  int change;
  for (a = 0; a<n-1; ++a)
  {
    change = 0;
    c = a;
    buf = item[a];
    for (b = a+1; b<n; ++b)
      if (item[b]<buf)
      {
        buf = item[b];
        c = b;
        change = 1;
      }
    if (change)
    {
      item[c] = item[a];
      item[a] = buf;
    }
  }
}

void s3 (float* item, int n)
{
  int a,b;
  float buf;
  for (a = 1; a<n; ++a)
  {
     buf = item[a];
     b = a-1;
     for (b = a-1; b>=0 && buf < item[b]; b--)
```

```
      item[b+1] = item[b];
    item[b+1] = buf;
  }
}

void s4(float* item, int n)
{
  int step[5] = {9,5,3,2,1};
  int i,j,k,h;
  float buf;
  for (k = 0; k<ST; k++)
  {
    h = step[k];
    for (i = h; i<n; i++)
    {
      buf = item[i];
      for(j = i-h; buf<item[j] && j>=0; j-=h)
        item[j+h] = item[j];
      item[j+h] = buf;
    }
  }
}

void s5(float* item, int left, int right)
{
  int i,j;
  float comp,buf;
  i = left; j = right;
  comp = item[(left+right)/2];
  do {
    while (item[i]<comp && i<right)
      i++;
    while (comp<item[j] && j>left)
      j--;
```

```
    if (i<=j) {
      buf = item[i];
      item[i] = item[j];
      itme[j] = buf;
      i++; j--;
    }
  } while(i<=j);
  if (left<j)
    s5(item, left, j);
  if (i<right)
    s5(item, i, right);
}
```

Пример выполнения контрольной работы

Пусть номер студенческого билета 083457.

Тогда две последние цифры: A = 5, B = 7.

Сумма цифр A + B = 5 + 7 = 12.

$(A+B)_4 = 3\mathbf{0}$. Последняя цифра = 0.

1. **Тип списка** – **1-связный кольцевой**.

2. **Тип данных** – длинные беззнаковые целые числа (**unsigned long int**).

3. **Тип звена списка**:

```
struct Zveno {
  unsigned long int Data;
  Zveno *next;
};
```

Процедуры работы со связным списком выбираются по типу списка – для **1-связного кольцевого** списка.

4. **Процедура добавления звена**:

```
void i1(Zveno* Pred, unsigned long int data)
{
  Zveno* Loc = new Link1;
  Loc->Data = data;
  Loc->next = Pred->next;
  Pred->next = Loc;
}
```

5. **Процедура удаления звена**:

```
void d2(Zveno* Pred)
{
  Zveno* Loc;
  Loc = Pred->next;
  Pred->next = Loc->next;
  delete Loc;
}
```

6. **Процедура поиска в списке**:

```
int p2(Zveno* Start, Zveno*& Find, Zveno*& Pred, unsigned long int Key)
{
  Zveno* Cur = Start->next;
  Pred = Start;
  int Success = 0;
  while (Cur!=Start && !Success)
  {
    if(Cur->Data = = Key)
    {
      Find = Cur;
      Success = 1;
      break;
    }
    Pred = Cur;
    Cur = Cur->next;
  }
  return Success;
}
```

7. **Процедура просмотра списка** строится по аналогии с процедурой поиска и является значительно упрощённым вариантом последней:

```
void prosmotr(Zveno* Start)
{
  Zveno* Cur = Start->next;
  while (Cur!=Start)  // Проще условие выполнения цикла
  {
    cout << Cur->Data << ”  ”;  // Вывод данных на экран
    Cur = Cur->next;
  }
}
```

8. Ведущее звено **1-связного кольцевого** списка:

Zveno* L1;

L1 = new **Zveno**;

L1->next = L1;

9. **Процедура сортировки** массива:

B = 7. B > 4 => B – 5 = 7 – 5 = 2

Пузырьковая сортировка.

Процедура сортировки выбирается по известным для каждого метода особенностям процедур.

```
void sort(unsigned long int* item,int n)
{
 int a,b;
 unsigned long int buf;
 for (a=1; a<n; ++a)
   for (b=n-1; b>=a; --b)
     if (item[b-1]>item[b])
     {
       buf = item[b-1];
       item[b-1] = item[b];
       item[b] = buf;
     }
}
```