МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ

федеральное государственное бюджетное образовательное учреждение высшего образования «Казанский национальный исследовательский технический
университет им. А.Н. Туполева-КАИ»
(КНИТУ-КАИ)
Чистопольский филиал «Восток»

МЕТОДИЧЕСКИЕ УКАЗАНИЯ К ПРАКТИЧЕСКИМ ЗАНЯТИЯМ

по дисциплине

ПРИКЛАДНЫЕ ИНФОРМАЦИОННЫЕ ТЕХНОЛОГИИ

Индекс по учебному плану: **Б1.О.15**
Направление подготовки: **12.03.01 Приборостроение**
Квалификация: **Бакалавр**
Профиль подготовки: **Приборостроение**
Типы задач профессиональной деятельности: **проектно-конструкторская, производственно-технологическая**

Рекомендовано УМК ЧФ КНИТУ-КАИ

Чистополь
2023 г.

Практическое занятие 1

Программирование простейших циклов

1. Вывести 10 раз текст с новой строки

Чистополь

printf (“\n Чистополь ”);

Операторы цикла:

Цикл с предусловием

```
i=0;
while (i<10)                пока условие истинно
{
printf (“\n Чистополь”);    вывод строки
i++;                        оператор
}
```

Цикл с постусловием

```
i=0;
do
{ printf (“\n Чистополь”);  вывод строки
i++;                        делать оператор
} while (i<10);             пока i <10
```

Цикл с параметром

```
for( i=0; i<10; i++)        пока параметр <10
 printf (“\n Чистополь”);   вывод строки
```

2. Вычислить 1+2+3+ …+ n

3. 1 2 *3 *… *N

4. x^n

5. $S= 1+x+x^2/2! +… +x^n/n!$

6. Дано натуральное число n. Определить сумму цифр этого числа и количество цифр. Определить есть ли заданная цифра.

7. Вычислить t +2t +3t+…+nt

Практическое занятие 2

Условный оператор

1. Вычислить y=1/x. Учесть случай x=0. Оператор ветвления (оба вар). 3 варианта if : if (x!=0) {y=1/x; printf();}else printf(); if(x==0); if(!x); (c else) или просто

if (x!=0) printf (“\ny=%f”,1/x);

2. Записать следующие фрагменты программы без помощи оператора for. Дополнить их определением типов переменных.

```
for(r=-1,n=500; n>0; n--)
{ scanf("%f",&z);
  if (z<r) r=z;
}
```

3. Записать следующие фрагменты программы с помощью оператора for. Дополнить их определением типов переменных.

```
y=0; m=100;
while (m >= 1)
{  scanf("%f",&x);
   if(x<0) y=y-x;
        else   y=y+x;
    m=m-1;
}
```

Задача 1. Составить программу построения (вывода) арифметической прогрессии для заданных начального члена, разности прогрессии, требующегося количества членов. Последовательность вида

t t+r t+2r t+3r ...

называют арифметической прогрессией, число t - начальным членом, а число r - разностью прогрессии.

Решение

```
/*      Программа 2.1. Арифметическая прогрессия                */
```

```
#include <stdio.h>
main()
{
  int t;                    /* начальный член          */
  int r;                    /* разность прогрессии     */
  int kol;                  /* количество членов       */
  scanf ("%d%d%d", &t, &r, &kol);
  printf ("\n%d", t);       /* вывод первого члена     */
  for (kol--; kol>0; kol--)
  {  t += r;
     printf ("  %d", t);    /* вывод очередного члена  */
  }
}
```

Задания

1. Дано n – натуральное. Определить совпадают ли 1 и последняя цифры.

2. Дано n – натуральное. Определить четная ли разность 1 и последней цифры.

3. $t \;\; t*r \;\; t*r^2 \;\; t*r^3 \ldots t*r^n$

Выполнить трассировку, составить схемы.

4. Записать следующие фрагменты программы без помощи оператора for. Дополнить их определением типов переменных.

а)

```
for(j=0;j<10;j++)
 m[j] = 0;
```

б)

```
for(r=-1, n=500; n>0; n--)
{ scanf ("%f",&z);
  if (z<r) r=z;}
```

5. Записать следующие фрагменты программы с помощью оператора for. Дополнить их определением типов переменных.

а) б) в)

```
j=0;                    y=0; m=100;                j=0; k=0;
while (j < 80)              while (m >= 1)             while (t[k]!='.')
{  s[j] = '  ';         {  scanf("%f", &x);        {  if (t[k]!='*')
   j++;                        if(x<0)  y = y - x;          {   r[j]=t[k];  j++; }
}                          else      y = y+x;           k++;
   m = m-1;                    }
}
```

6. Составить программу вычисления произведения двух натуральных чисел A и B, не используя операцию умножения.

Практическое занятие 3

Последовательная обработка символов

Задача 1. Дана последовательность символов, продолжающаяся до конца файла (при вводе с клавиатуры она заканчивается комбинацией клавиш Ctrl+Z). Выяснить, верно ли, что среди символов имеются все буквы слова "КНИТУ".

Решение.

```
/*   Программа 1. Буквы слова " КНИТУ "                    */
#include <stdio.h>
main()
{  int i;                  /* счетчик                      */
   char c;                 /*текущий символ                */
   int prov;               /* результат проверки (1 или 0) */
   char s[]="КНИТУ ";
   int priz[5]={0,0,0,0,0};/* признаки проверяемых букв     */
   printf ("\nВведите последовательность символов\n");
   printf ("Завершение последовательности CTRL+Z\n");
   while ((c=getchar())!=EOF)
   {  i=0; prov=1;
      while(i<5 && prov)
      {   if (c==s[i])  { priz[i]=1; prov=0;}
          i++;
      }
   }
   prov = 1;
   for (i=0; i<5; i++)
   if (priz[i]==0) prov=0;
     if (prov) printf ("\n Есть все символы слова КНИТУ \n");
   else
   {  printf("\n Нет символов ");
```

```
      for (i=0; i<5; i++)
        if (!priz[i]) {  putchar(s[i]);  putchar(' ');  }
    }
}
```

Задачи

1. Дан текст произвольной длины, продолжающийся до конца файла. Составить программы для решения следующих задач:

а) Найти порядковый номер первой запятой.

б) Найти порядковый номер последней запятой.

в) длину каждой последовательности пробелов.

г) длину максимальной последовательности пробелов.

д) количество сочетаний AB

е) количество слов с буквой ‘a’

2. Определить длину каждого слова

3. Определить максимальное слово, состоящее из цифр

Практическое занятие 4

Тема: Обработка массивов

Задача 1. Дано натуральное число N>0 и последовательность из N действительных чисел. Напечатать сначала все отрицательные, а затем все остальные числа.

Решение

```
/*Программа 6.1. Печать отрицательных, затем остальных чисел */
#include <stdio.h>
#define N 100
main()
{  float a;                /* переменная для вводимых чисел        */
   float mas[N];           /* массив для неотрицательных чисел     */
   int i;                  /* счетчик чисел                        */
   int n;                  /* количество чисел                     */
   int k=0;                /* количество неотрицательных чисел     */
   int pr=1;               /* 1 - выводить заголовок, 0 - нет      */
   printf ("\nВведите количество чисел (не более 100)\n");
   scanf ("%d", &n);
   printf ("Введите последовательность из %d чисел\n",n);
   for (i=0; i<n; i++)
   {  scanf("%f",&a);
      if (a<0)
      {  if (pr)  {  printf ("\nОтрицательные числа:\n"); pr=0;  }
         printf("%5.2f ",a);
      } else   {  mas[k]=a; k++;  }
   }
   if (k)
   {  printf ("\nНеотрицательные числа:\n");
      for ( i=0; i<k; i++)  printf ("%5.2f ", mas[i]);
      putchar('\n');
```

```
    } else    printf("\nНеотрицательных чисел нет.\n");
}
```

Результаты работы программы

Тест 1

```
Введите количество чисел
10
Введите последовательность из 10 чисел
-1 4 3.2 7 -5.1 6 0 8.3 -9 11
Отрицательные числа:
-1.00 -5.10 -9.00
Неотрицательные числа:
 4.00  3.20  7.00  6.00  0.00  8.30 11.00
```

Тест 2

```
Введите количество чисел
6
Введите последовательность из 6 чисел
-2.0 -8.1 -4.5 -10.9 -1.5 -5.1
Отрицательные числа:
-2.00 -8.10 -4.50 -10.90 -1.50 -5.10
Неотрицательных чисел нет.
```

Задача 2. Дан массив A из M действительных чисел. Известно, что A[1]>0 и что среди остальных элементов есть хотя бы одно отрицательное число. Найти сумму элементов, предшествующих первому отрицательному элементу.

Решение

```
/*   Программа 2. Сумма элементов до отрицательного числа       */
#include <stdio.h>
#define  M 100
main()
{  float a[M];        /* массив действительных чисел                */
```

```
   float s=0;          /* сумма элементов до 1-го отрицательного      */
   int i;              /* счетчик                                     */
   int n;              /* количество элементов в массиве              */
   printf ("\nВведите количество элементов в массиве\n");
   scanf ("%d",&n);
   printf ("Введите массив из %d чисел\n",n);
   for (i=0; i<n; i++)     scanf("%f",&a[i]);
   if (a[0]<0)  printf ("\nМассив начинается с отрицательного числа\n");
   else
   /* цикл суммирования положительных элементов до                */
   /* отрицательного или до конца массива                         */
   {  i=0;
      do
        s += a[i++];              /*  соответствует s = s + a[i];  i = i + 1;  */
      while (a[i] > 0 &&  i < n);
      if (i==n)
      {  printf ("\nВ массиве только положительные числа");
         printf ("\nСумма чисел = %5.2f\n",s);
      }
      else   printf ("\nСумма до отрицательного  = %5.2f\n",s);
   }
}
```

Результаты работы программы

Тест 1

```
Введите количество элементов в массиве
6
Введите массив из 6 чисел
1 2 3 -4 5 6
Сумма чисел до отрицательного  =  6.00
```

Тест 2

Введите количество элементов в массиве

6

Введите массив из 6 чисел

1 2 3 4 5 6

В массиве только положительные числа

Сумма чисел = 21.00

Тест 3

Введите количество элементов в массиве

6

Введите массив из 6 чисел

-1 2 3 4 5 6

Массив начинается с отрицательного числа

Задача 3. Целочисленная квадратная матрица размера n*n (n<=50) вводится по строкам. Составить программу определения номера столбца, имеющего максимальную сумму элементов, расположенных выше главной диагонали.

Решение.

Входная матрица:

(точками отмечены элементы, не участвующие в задаче)

n=5

	0	1	2	3	4
0	.	6	1	3	-2
1	.	.	4	-1	3
2	.	.	.	5	1
3	.	.	.	.	2
4	.	.	.	.	.

Тест

Результат:

Номер столбца с максимальной суммой равен 3

/* Программа 3. Обработка матрицы */

```
/*Поиск столбца с наибольшей суммой выше главной диагонали */
#include <stdio.h>
#define  NMAX  50          /* Максимальный размер матрицы       */
void main (void)
{  int m[NMAX][NMAX];  /* Матрица                                 */
   int  n;                      /* Количество строк и столбцов          */
   int  i;                       /* Индекс текущей строки                */
   int  j;                       /* Индекс текущего столбца              */
   int  jmax;            /* Индекс столбца с максим. суммой    */
   int s, smax;                  /* Сумма текущего столбца и максим.   */

   /* 1. Ввод матрицы m по строкам                                      */
   scanf ("%d", &n);
   for (i=0; i<n; ++i)
       /* Ввод i-й строки                                              */
       for (j=0; j<n; ++j)    scanf("%d", &m[i][j]);
   /* 2.Поиск столбца с максим.суммой выше главной диагонали */
   jmax = 1;   smax = m[0][1];
   for (j=2; j<n; j++)               /* Перебор столбцов                */
   {  /* Вычисление суммы j-го столбца                                  */
      for (s=0, i=0; i<j; i++)
        s = s + m[i][j];
      if (s > smax)   { smax=s;   jmax=j; }
   }
   printf ("\nНомер столбца с максим-й суммой = %d", jmax);
}
```

Задача 4. Составить программу для решения следующей задачи. Заполнить квадратную матрицу A = $\{a_{i,j}\}$ (i, j = 1, ..., n) значениями, вычисленными по формуле $a_{i,,j}$ = i * j % 5 + i - j. Из матрицы получить вектор X= $\{x_i\}$ (i = 1, ..., n). Элемент x_i вычислять как скалярное произведение i-ой

строки матрицы на столбец, содержащий первый по порядку максимальный элемент i-ой строки. Полученный вектор упорядочить по возрастанию методом последовательного нахождения минимума.

Решение

```
/*   Программа 4. Обработка матрицы и вектора                    */
#include <stdio.h>
#define N 10        /*максимальный размер матрицы                 */
main()
{  int n;           /* размер  матрицы                        */
   float a[N][N];   /* матрица                                  */
   float x[N];      /* вектор                                 */
   int i,j,k;       /* текущие индексы строк и столбцов        */
   float m;
   /*          Ввод матрицы                                    */
   printf ("\nВведите размер массива\n");
   scanf("%d",&n);
   for (i=0; i< n; i++)
     for (j=0; j< n; j++)
        a[i][j] = i * j % 5 + i - j;
   /*          Печать  матрицы                                 */
   printf ("\nПолученная матрица\n");
   for (i=0; i< n; i++)
   {  putchar('\n');
      for (j=0; j< n; j++)   printf("%3.0f ",a[i][j]);
   }
   putchar('\n');
   /*          Получение вектора из матрицы                    */
   /* Определение номера столбца, содержащего                  */
   /* первый по порядку максимальный элемент                   */
```

```
   for (i=0; i< n; i++)
   {  k=0;
      for (j=1; j< n; j++)
         if (a[i][j] > a[i][k])  k = j;
      /* вычисление скалярного  произведения                    */
      /*   i-й строки на k-й столбец                            */
      x[i] = 0;
      for (j=0; j<n; j++)
         x[i] = x[i] + a[i][j] * a[j][k];
   }
   /*   Печать  вектора                                         */
   printf ("\nПолученный из матрицы вектор\n");
   for (i=0; i< n; i++)
      printf ("%3.0f ", x[i]);
   putchar('\n');
   /*   Упорядочивание вектора                                  */
  for (i=0; i<n; i++)
  {  j = i;   m = x[i];
     for (k=i+1; k<n; k++)
        if (x[k] < m) { j = k;   m = x[k]; }
     x[j]=x[i]; x[i]=m;
  }
  /* Печать  упорядоченного вектора                            */
  printf ("\nУпорядоченный вектор\n");
  for (i=0; i< n; i++)
    printf("%3.0f ",x[i]);
  putchar('\n');
}
```

Результаты работы программы

Введите размер массива

6

Полученная матрица

0 -1 -2 -3 -4 -5

1 1 1 1 1 -4

2 3 4 0 1 -3

3 5 2 4 1 -2

4 7 5 3 1 -1

5 4 3 2 1 0

Полученный из матрицы вектор

-55 -10 11 27 36 20

Упорядоченный вектор

-55 -10 11 20 27 36

Задачи 5: Дана матрица действительных чисел A размера n*n (n<=40). Составить фрагменты программ для решения следующих задач.

Из матрицы A получить компоненты вектора X[n] равные:

а) суммам элементов строк;

б) значениям средних арифметических элементов строк;

в) числу отрицательных элементов в строке;

г) сумме положительных элементов столбцов;

д) количеству отрицательных элементов строк;

е) наименьшим элементам строк;

ж) наибольшим элементам столбцов;

з) разностям наибольших и наименьших элементов строк;

В векторе X выбрать некоторое значение Z, равное соответственно:

а) наибольшему элементу;

б) номеру наименьшего элемента;

в) сумме наибольшего и наименьшего элемента;

г) наименьшему элементу;

д) номеру наибольшего элемента;

е) сумме элементов вектора;

ж) среднему арифметическому элементов вектора;

з) произведению элементов.

Задачи 6: Дана матрица действительных чисел размера n*m (n<=50, m<=70). Составить фрагменты программ для решения следующих задач.

а) Вычислить сумму положительных элементов каждой строки и найти номер строки, в которой эта сумма наибольшая.

б) Найти минимальный элемент в столбце матрицы (из не равных нулю) и разделить на него все элементы этого столбца.

в) Найти наибольший положительный и наименьший отрицательный элементы.

г) В каждой строке переставить местами наибольший и наименьший элементы.

Задачи 7: Дана квадратная действительная матрица B размера n*n (n<=70).

Составить фрагменты программ для решения следующих задач:

а) Вычислить сумму элементов, расположенных над главной диагональю.

б) Найти наибольший элемент главной диагонали и вывести всю строку, в которой он находится.

в) В каждом столбце матрицы найти максимальный элемент и обменять его с элементом, расположенным на главной диагонали.

г) Найти строку с наибольшим количеством отрицательных элементов и разделить элементы этой строки на минимальный элемент главной диагонали.

Задачи 8: Элемент матрицы назовем седловой точкой, если он одновременно является наименьшим в своем столбце и наибольшим в своей строке или наоборот. Найти индексы всех седловых точек заданной матрицы.

Задачи 9: Определить, является ли заданная целочисленная квадратная матрица магическим квадратом, т.е. такой, что суммы элементов во всех строках и столбцах одинаковы.

Задачи 10: Даны целые числа M и N и вещественная матрица из M строк и N столбцов, вводимая по строкам (строка за строкой). Составить программы решения следующих задач.

а) Определить средние арифметические значения каждой строки.

б) Определить средние арифметические значения каждого столбца.

в) Индексы и значение максимального из элементов матрицы (одного из таких элементов).

Практическое занятие 5

Функции

Задача 1. Составить определение функции F(X) = √X с погрешностью не более 0.00001.

Решение. Используем итерационную формулу Ньютона (см. задачу 2.3 6). Для вычисления квадратного корня k = 2, поэтому

$Y_0=1, Y_{n+1} = (Y_n + X / Y_n) / 2.$

```
/*   Программа 8.1. Функция F(x) = квадратный корень (x) */
#include <math.h>
float F (float x)
{  float y;                  /*последнее приближение функции      */
   float ypr;                /*предыдущее приближение функции     */
   const float e=0.00001;/* допустимая погрешность                 */
   y = 1;
   do
   {  ypr = y;
      y = (ypr + x / ypr) / 2;
   } while (fabs(y-ypr) >= e);
   return y;
}
```

2. Составить определения следующих функций.

а) Расстояние между двумя точками, заданными своими координатами на плоскости.

б) max (x, y).

в) min (x, y).

г) max (a, b, c).

д) min (a, b, c).

е) 1, если заданный символ является латинской буквой, и 0 в противном случае.

ж) 0, если нельзя построить треугольник из трех отрезков заданной длины (каждая сторона треугольника должна быть меньше суммы двух других сторон); 1 - треугольник равносторонний, 2 - равнобедренный, 3 - любой другой.

з) Число Фибоначчи f(n), где f(0) = 0, f(1) = 1, f(j) = f(j-1) + f(j-2) для целого j>1.

и) Наибольший общий делитель натуральных чисел A и B.

к) Количество десятичных цифр заданного целого числа.

л) Значение числа, полученного выписыванием в обратном порядке десятичных цифр заданного натурального числа.

м) Количество единичных битов в двоичной записи заданного числа типа unsigned long (см. задачу 2.44).

н) Ввод и вычисление значения двоичного целого числа.

3. Дана последовательность из 150 действительных чисел. Вычислить сумму и среднее арифметическое значение первых 100 чисел и последующих 50 чисел.

Решение. Составим подпрограмму ввода n чисел и вычисления их суммы s и среднего арифметического значения sa. Входным параметром подпрограммы является n, выходными параметрами - s и sa. Для решения задачи вызовем эту подпрограмму для n=100 и n=50.

```
/*      Программа 8.3. Сумма и среднее арифметическое          */
#include <math.h>
/* Подпрограмма: s = сумма, sa = среднее арифметич. n чисел    */
void sum_sr (int n, float *s, float *sa)
{  float x;                    /* текущее число                */
   int j;
   *s=0;
   for (j=0; j<n; j++)
```

```
    {  scanf("%f", &x);
       *s = *s + x;
    }
    *sa = *s / n;
}
void main()
{  float sum1, sum2, sr1, sr2;
   sum_sr (100, &sum1, &sr1);
   sum_sr (50, &sum2, &sr2);
   printf ("%f %f %f %f", sum1, sr1, sum2, sr2);
}
```

4. Составить подпрограммы для решения следующих задач.

а) Поменять местами значения переменных x, y.

б) Значения переменных x, y, z поменять местами так, чтобы оказалось $x \geq y \geq z$.

в) Вычислить длину окружности, площадь круга и объем шара одного и того же заданного радиуса.

г) Вычислить площадь и периметр прямоугольного треугольника по длинам двух катетов.

д) Звездочками '*' нарисовать на экране прямоугольник с заданной шириной и высотой, например, шириной 7 и высотой 4 символа:

```
*******
*     *
*     *
*******
```

е) Для квадрата ABCD на плоскости даны координаты XA, YA, XC, YC диагонально расположенных вершин A и C. Вычислить координаты вершин B и D.

ж) Вывод заданного целого числа в двоичной системе счисления.

з) Вывод заданного целого числа в шестнадцатеричной системе счисления.

5. Пусть p, q, r - целочисленные беззнаковые переменные. Составить подпрограммы для решения следующих задач.

а) Упаковка. Поместить младшие 3 бита переменной p в младшие 3 бита переменной r, а младшие 6 битов переменной q - в следующие 6 битов переменной r без изменения остальных битов.

б) Распаковка. Присвоить младшим битам переменной p младшие 3 бита переменной r, а младшим битам переменной q - следующие 6 битов переменной r. Остальные биты p и q обнулить.

Практическое занятие 6

Рекурсивные функции

Задача: Дано ***n*** различных натуральных чисел. Напечатать все перестановки этих чисел

```
#include <conio.h>
#include <iostream.h>
#include <stdlib.h>
#include <stdio.h>
#include <string.h>
int n,ks;
char in[30];
char pr;
void recurs(int n){
    if (n!=ks/2){
         pr=in[n];
         in[n]=in[(ks-1)-n];
         in[(ks-1)-n]=pr;
                  recurs(n+1);
                 }
}
void main(){
 clrscr();
  cout << "Введите слово":";
   cin >> in;
    ks=0;
  ks=strlen(in);
  n=0;
 recurs(0);
 cout << "\n" << in;
 getch();
```

}

1. Найдите сумму цифр заданного натурального числа, используя рекурсивную подпрограмму.

2. Подсчитать количество цифр в заданном натуральном числе, используя рекурсивную подпрограмму.

3. Описать функцию C(m,n), где 0≤ m≤ n, для вычисления биномиального коэффициента C_n^m по следующей формуле:

$C_n^0 = C_n^n = 1; C_n^m = C_{n-1}^m + C_{m-1}^{n-1}$ при 0< ***m***< ***n***

используя рекурсивную подпрограмму

4. Описать рекурсивную функцию Root(***f***, ***b***, ***ε***), которая методом деления отрезка пополам находит с точностью ε корень уравнения f(x) = 0 на отрезке [***a***, ***b***] (считать, что ***ε*** > 0, ***a*** < ***b***, ***f***(***a***} • ***f***(***b***) < 0 и ***f***(***x***)- непрерывная и монотонная на отрезке [***a***, ***b***]).

5. Описать функцию ***min***(***x***) для определения минимального элемента линейного массива x, введя вспомогательную рекурсивную функцию ***min***1(***k***), находящую минимум среди последних элементов массива ***x***, начиная с ***k***-го.

6. Описать рекурсивную логическую функцию Simm(***S***, i, j), проверяющую, является ли симметричной часть строки ***S***, начинающаяся ***i***-м и кончающаяся ***j***-м ее элементами.

7. Составить программу вычисления наибольшего общего делителя двух натуральных чисел, используя рекурсивную подпрограмму.

8. Составить программу нахождения числа, которое образуется из данного натурального числа при записи его цифр в обратном порядке, используя рекурсивную подпрограмму. Например, для числа 1234 получаем ответ 4321.

9. Составить программу перевода данного натурального числа в ***p***-ичную систему счисления (2 ≤ p≤ 9) ,используя рекурсивную подпрограмму.

10. Дана символьная строка, представляющая собой запись

натурального числа в ***p***-ичной системе счисления ($2 \le p \le 9$). Составить программу перевода этого числа в десятичную систему счисления , используя рекурсивную подпрограмму.

11. Составить программу вычисления суммы:

$1! + 2! + 3! + \ldots + n!$ ($n \le 20$).

Примечание. Тип результата значения функции - long.

12. Составить программу вычисления суммы:

$2! + 4! + \ldots + n!$ ($n \le 20$, n - четное).

Примечание: Тип результата значения функции — long.

Практическое занятие 7

Рекурсивные функции

Ввести строки. Сравнить первые десять символов двух строк. Если они одинаковы, объединить две строки, исключив из второй первые десять символов. В случае отличия первых десяти символов скопировать вторую строку в первую. Посчитать длину исходной и полученной первой строки.

Вариант решения 1.

```
#include <stdio.h>
#include <string.h>
#include <locale.h>
int main()
{setlocale (LC_CTYPE, "Russian");
char a[200], b[200], rest[200];
char c1[11], c2[11];
printf ("\nСтрока 1: ");
fgets (a, 200, stdin);
if (a[strlen(a) - 1] == '\n')
a[strlen(a) - 1] = '\0';
printf ("\nСтрока 2: ");
fgets (b, 200, stdin);
if (b[strlen(a) - 1] == '\n')
b[strlen(a) - 1] = '\0';
strncpy(c1, a, 10);
strncpy(c2, b, 10);
c1[10] = c2[10] = '\0';
printf ("\nДлина первой строки до изменения:\t, %d", strlen(a));
if (strcmp(c1, c2) == 0)
{   printf( "\nПервые десять символов совпадают. Копируем вторую строку в первую\n");
memcpy (rest, b+10, strlen(b)-9);
```

```
        strcat(a, rest);
        }
        else
        {    printf ( "\nПервые десять символов не совпадают. Объединяем строки\n");
        strcat(a,b);
        }
        printf( "\nСтрока  1: %s",a);
        printf( "\nДлина первой строки после изменения:\t %d", strlen(a));
        getchar();
        return 0;
        }
```

Вариант решения 2

```
#include <stdio.h>
#include <string.h>
int main()
{
   char a[200], b[200], rest[200];
   char c1[11], c2[11];
   printf ("\nСтрока 1: ");
   fgets (a, 200, stdin);
   if (a[strlen(a) - 1] == '\n')
      a[strlen(a) - 1] = '\0';
   printf ("\nСтрока 2: ");
   fgets (b, 200, stdin);
   if (b[strlen(a) - 1] == '\n')
```

```
    b[strlen(a) - 1] = '\0';
  strncpy(c1, a, 10);
  strncpy(c2, b, 10);
  c1[10] = c2[10] = '\0';
  printf ("\nДлина первой строки до изменения:\t, %d", strlen(a));
  if (strcmp(c1, c2) == 0)
  {  printf( "\nПервые десять символов совпадают. Копируем вторую строку в первую\n");
     memcpy (rest, b+10, strlen(b)-9);
     strcat(a, rest);
  }
  else
  {  printf ( "\nПервые десять символов не совпадают. Объединяем строки\n");
     strcat(a,b);
  }
  printf( "\nСтрока  1: %s",a);
  printf( "\nДлина первой строки после изменения:\t %d", strlen(a));
  return 0;
}
```

1. Ввести предложение, слова в которых разделены пробелами и запятыми. Распечатать это предложение, удалив из него те слова, которые встретились там более одного раза.
2. Даны две символьные строки, состоящие только из цифр (длина каждой более 10 символов). Считая, что в этих строках находятся очень длинные числа, сформировать третью строку- сумму этих чисел.

3. Дан произвольный текст. Отредактировать текст так, чтобы:
   -- между словами был ровно один пробел;
   -- предложения в тексте разделялись ровно двумя пробелами.
4. Ввести два предложения и распечатать самые длинные слова, общие для этих предложений. Если нужных слов нет - сообщить об этом.

Практическое занятие 8

Структуры

Задача 1: Написать программу поиска сотрудников, имеющих стаж работы в ОВД от 10 до 15 лет, использующая функцию *vvod_dannyh* () для ввода информации о сотрудниках и функцию *poisk* () для организации указанного поиска.

```
/*      Программа 1. Обработка структур                    */
#include "stdafx.h"
#include <iostream>
using namespace std;
struct sotrudnik {
char Fam [20]; int Nomer_udost; int Stag;};
void vvod_dannyh (sotrudnik *v_point);
void poisk (sotrudnik *p_point);
int main()
{
sotrudnik Sev_ROVD[5], *point;
point=&Sev_ROVD[0];
vvod_dannyh (point);
cout <<endl;
cout << "Rezul'taty poiska:"<<endl;
poisk (point);
return 0;
}
void vvod_dannyh (sotrudnik *v_point)
{
for (int i=0; i<5; i++)
{cout << "Vvedite Familiu=";cin >> v_point->Fam;
cout << "Vvedite Nomer slug. udostovereniay=";
cin >> v_point->Nomer_udost;
```

```
cout << "Vvedite stag slugby v OVD=";
cin >> v_point->Stag;
v_point++;}
}
void poisk (sotrudnik *p_point)
{
int k=0;
for (int i=0; i<5; i++)
{
if (p_point->Stag >= 10 && p_point->Stag <= 15)
 {k++;
  cout << k <<". "<< p_point->Fam<<"=";
  cout <<p_point->Stag<<" let";
  cout << endl; }
  p_point++;
}
if (k==0)
cout << "Sotrudnikov s ukazamnnym stagem net!"<< endl;
        }
```

Задача 2:

а) Реализовать массив структур (*Priem*), содержащий сведения об абитуриентах, поступающих в Вуз. Структура *abiturient* имеет следующие поля: фамилию абитуриента (*Fam*), его имя (*Imay*) и отчество (*Otch*), год рождения (*God_rogd*), номер аттестата о среднем образовании (*Nomer_at*), средний балл аттестата (*Sr_ball*), оценка по трем вступительным экзаменам (*Rus, Math, Inf*), контактный телефон (*Tel*). Размер массива не более 25 элементов.

б) Вывести на экран информацию об абитуриентах, набравших не менее 13 баллов по результатам сдачи вступительных экзаменов. Информацию представить в виде таблицы, имеющей следующие столбцы:

фамилия, имя и отчество абитуриента; сумма баллов, набранная на вступительных экзаменах; средний балл аттестата.

Задача 3:

а) Реализовать массив структур (*Biblio*), содержащий сведения о книгах, находящихся в библиотечном фонде Вуза. Структура *kniga* имеет следующие поля: автор книги (*Fio*), название книги (*Name*), год издания (*God_izd*), издательство (*Izdat*), количество экземпляров (*Kol_vo*), название дисциплины, в которой используется книга (*Predmet_name*), шифр книги (*Kod*). Размер массива не более 25 элементов.

б) Вывести на экран информацию о книгах, имеющихся в библиотеке, не позже 2003 года издания. Информацию представить в виде таблицы, имеющей следующие столбцы: автор книги; название, год издания, количество экземпляров.

Практическое занятие 9

Работа с файлами

1. Подсчитать, количество в данном файле f символов «А».

2. Подсчитать, в данном файле f количество сочетаний КАИ.

3. Распечатать все строки данного файла, содержащие заданную строку в качестве подстроки. Имя файла и сама строка задаются пользователем с клавиатуры.

4. Реализовать программу, которая определяет какой символ чаще других встречается в заданном файле. Имя файла задается пользователем

5. Реализовать программу, которая определяет сколько имеется строк, состоящих из одного, двух, трех и т.д. символов, содержится в данном файле. Считать, что длина каждой строки - не более 80 символов. Имя файла задается пользователем

6. Реализовать программу, которая определяет самую длинную строку в данном файле. Если таких строк несколько, то в качестве результата вывести первую из них. Имя файла задается пользователем.

7. Имеются два непустых файла. Вывести номер строки и номер символа в строке, в котором имеется отличие содержимое одного файла от содержимого второго файла . Если их содержимое абсолютно одинаково , то вывести на экран 0 и соответствующее сообщение; если один из этих файлов можно считать началом другого, то вывести соответствующее сообщение и - n+1, 1, где n - количество строк в более коротком файле. Данные файлы задаются своими именами пользователем с командной строки.

8. Имеется непустой файл, в котором содержится последовательность целых чисел . Имя файла задается пользователем в командной строке.

вывести наибольшее из этих чисел;

вывести, количество четных чисел, которые содержатся в файле;

c) вывести, образуется из этих чисел арифметическая прогрессия;

d) вывести, являются ли эти числа возрастающей последовательностью;

е) вывести количество чисел последовательности, которые являются точными квадратами;

9. Написать программу, определяющую, какая из строк чаще других встречается в данном файле.

Практическое занятие 10

Функции доступа к файлам

Список функций для работы с файловыми потоками хранится в библиотеке (заголовочном файле) fstream.h. Поэтому во всех рассматриваемых ниже фрагментах программ предполагается, что в начале программы есть соответствующая директива #include:

#include <fstream.h>

Создание потока ввода-вывода

Прежде чем начать работать с потоком необходимо его создать. Поток ввода создается инструкцией

ifstream <имя потока ввода>;

Поток вывода создается инструкцией

ofstream <имя потока вывода>;

Пример:

ifstream input;

ofstream output;

эти инструкции создают поток ввода input и поток вывода output.

Открытие и закрытие файла

После создания потока его можно подключить к файлу (открыть файл) инструкцией

<имя потока>.open (<имя файла>);

Здесь <имя файла> - текстовая константа или переменная.

Например, для подключения потока ввода ifstream с именем input к файлу data.txt надо выполнить инструкцию

input.open (“data.txt”);

Аналогичная инструкция

output.open (“data.txt”);

подключит поток вывода output к файлу data.txt – файл подготовлен к записи данных. Важно отметить, что при выполнении оператора подготовки файла к записи, прежние данные из файла data.txt будут удалены.

Для отключения потока ввода-вывода от файла надо выполнить инструкцию закрытия файла:

<имя потока>.close ();

Так, инструкции

input.close();

output.close ();

отключают потоки ввода input и вывода output от файла data.txt, к которому они были подключены в предыдущих примерах. При закрытии файла вывода в конец файла записывается метка конца end_of_file.

Обработка ошибок

При выполнении операций над файлами, например, открытие и закрытие файлов, достаточно высока вероятность возникновения ошибочных ситуаций.

Один из простейших способов контроля корректности выполнения файловых операций заключается в вызове функии

<имя потока>.fail()

например,

input.fail()

Эта инструкция выполняется как вызов булевской функции, которая возвращает значение false (0), если последняя операция с потоком input завершилась успешно и возвращает значение true (1), если последняя операция с потоком input привела к ошибке (например, была попытка открытия несуществующего файла). В случае возникновения ошибки поток может быть поврежден, поэтому работу с ним продолжать нельзя.

Распознать ошибку можно и с помощью перегруженной операции отрицания. Выражение !<имя потока> также принимает значение false (0), если последняя операция с потоком завершилась успешно и принимает значение true (1), если последняя операция с потоком привела к ошибке.

Пример:

ifstream input;

input.open (“data.txt”);

if(!input) exit(1); // завершение работы программы

функция exit() описана в библиотеке stdlib.h.

Чтение-запись символов в файл

После того, как файл открыт для ввода данных, из него можно считывать отдельные символы. Чтение текущего символа из потока ввода выполняется инструкцией

<имя потока>.get(<имя переменной>);

Например, после выполнения инструкции input.get(ch); произойдет следующее: переменной ch будет присвоено значение текущего символа (шаг 1), и поток input будет подготовлен для чтения следующего символа (шаг 2).

Аналогично после того, как файл открыт для вывода данных, в него можно записывать отдельные символы. Запись символа в поток вывода выполняется инструкцией

<имя потока>.put(<имя переменной>);

Например, после выполнения инструкции output.put(ch); произойдет следующее: в поток output будет помещено значение символьной переменной ch (шаг 1), и поток output будет подготовлен для записи следующего символа (шаг 2). Вместо имени переменной можно указать значение выводимого в поток символа: инструкция output.put(‘t’); выводит в поток output символ ‘t’.

При чтении данных из файла (из потока ввода, связанного с файлом) необходимо уметь определять конец файла. В C++, как и в Паскале, для этой цели используется функция eof(). Логическое выражение <имя потока>. eof() принимает значение true (1), если конец файла достигнут и значение false (0), если можно продолжать чтение.

Пример (посимвольное чтение данных из файла и вывод на экран):

#include <iostream.h>

#include <fstream.h>

#include <stdlib.h>

void main()

```
{
  char ch;
  ifstream input;
  input.open("a:/text.txt");
  if(!input)
    {cout <<"\nmistake of open\n"; exit(1);} //Ошибка открытия файла
  while (!input.eof()){
    input.get(ch); //чтение очередного символа
    cout <<ch; //вывод символа на экран
  }
  input.close();
}
```

Ввод-вывод с преобразованием типов

Для того чтобы программа могла работать с числовыми данными, которые записаны в текстовом файле, необходимо при вводе выполнять преобразование символьной записи чисел (внешнее представление) в их внутреннее представление в памяти компьютера, т.е. выполнять преобразование типов данных при вводе. Аналогично при выводе на экран числа должны преобразовываться из внутреннего (двоичного) во внешнее представление. В Паскале такое преобразование автоматически выполнялось операторами ввода-вывода (**read/write**).

В **C++** преобразование числовых данных из внешнего (символьного) представления во внутреннее (двоичное) выполняет оператор >>. Обратное преобразование выполняет оператор <<. Эти операторы мы уже использовали при работе со стандартными потоками ввода-вывода **cin** и **cout**.

Инструкция ввода данных из потока ввода с преобразованием типов выглядит так:

поток ввода >>переменная >>переменная … >>переменная;

Например,

input >>a >>b >>c;

Аналогично выглядит инструкция вывода данных из потока вывода с преобразованием типов:

поток вывода <<выражение <<выражение ... <<выражение;

Например,

output <<a <<b+c <<2*m[5];

Пример (вывод-ввод в файл числовой последовательности):

```
#include <iostream.h>
#include <fstream.h>
#include <stdlib.h>
void main()
{
  int n, a;
  cout <<"\ninput n: "; cin >>n;
  ifstream input;
  ofstream output;
  output.open (“D:/data.txt”);
  if(!input)
     {cout <<”\nmistake of open\n”; exit(1);} //Ошибка открытия файла
  for (int i=1; i<=n; i++){
    cout <<"\ninput next number: "; cin >>a;
    output <<a;
    if (i!=n) output <<' '; // output.put(' ')
  }
  output.close ();
  input.open ("D:/data.txt");
  input >>a;
  while (!input.eof()){
    cout <<a <<' '; input >>a;
  }
```

input.close ();

}

При выводе в текстовый файл чисел необходимо предусмотреть разделитель. В приведенном примере в качестве разделителя в файл выводится пробел.

Чтение символьных строк из потока ввода

Для ввода строк из потока ввода (например, с клавиатуры) пригоден оператор >>, но его применение ограничено, поскольку этот оператор считает пробелы разделителями. Допустим, в программе содержатся операторы:

char name[20];

cout << "/ninput name: ";

cin >> name;

Если в сеансе работы с программой в ответ на запрос

input name:

ввести текст: **Петя Иванов**, переменной **name** будет присвоено только значение "Петя", т.к. оператор >> считает пробел разделителем, который сигнализирует о завершении ввода значения.

Очевидно, использовать здесь посимвольный ввод с помощью рассмотренной выше функии **get** тоже неудобно.

Для ввода символьных строк часто более удобной оказывается функция **getline(...)**, имеющая 2 параметра:

<имя потока>. getline(<имя строковой переменной>, n);

Здесь **n** - длина строки без учета нуль-символа **'\0'**.

Например, оператор:

input.getline(str, 80);

позволяет считать из потока ввода **input** строку с пробелами длиной до 79 символов (последний, 80-й символ строки – служебный нуль-символ). Аналогичный оператор **cin.getline(str, 80);** позволяет получить такую же строку от пользователя с клавиатуры.

Практическая работа 11

Графы

Граф - это пара (V,E), где V - конечное непустое множество вершин, а E - множество неупорядоченных пар (u,v) вершин из V, называемых ребрами. Ребро s=(u,v) соединяет вершины u и v. Ребро s и вершина u (а также s и v) называются **инцидентными**, а вершины u и v **смежными**. Степень вершины равна числу инцидентных ей ребер.

Ориентированный граф, или **орграф**, (V,E) отличается от обычного графа тем, что E - это множество упорядоченных пар (u,v) вершин,называемых дугами. Дуга (u,v) ведет от вершины u к вершине v. Вершина u называется **предшественником** v, а вершина v - **преемником** u. Графы представляются в программе чаще всего в виде **матрицы смежности** или **матрицы инцидентности**.

	0	1	2	3	4	5	6
0	0	1	1	0	0	0	0
1	1	0	1	0	0	0	0
2	1	1	0	1	0	0	0
3	0	0	1	0	0	0	0
4	0	0	0	0	0	0	0
5	0	0	0	0	0	0	1
6	0	0	0	0	0	1	0

а) матрица смежности

	a	b	c	d	e
0	1	1	0	0	0
1	1	0	1	0	0
2	0	1	1	1	0
3	0	0	0	1	0
4	0	0	0	0	0
5	0	0	0	0	1
6	0	0	0	0	1

б) матрица инцидентности

В матрице смежности 1 на пересечении i-й строки и j-го столбца означает, что вершины i и j смежны, а в матрице инцидентности – что вершина i и ребро j инцидентны. Для орграфа элемент матрицы смежности g[i][j]=1, если есть дуга i->j.

Описание на Си графа, представленного в виде матрицы смежности:

```
int  g[NMAX][NMAX];
```

где NMAX - это константа, задающая максимальное число вершин в графе.

Граф, представленный в виде матрицы инцидентности, можно описать так:

int g[NMAX][RMAX];

где RMAX - константа, задающая максимальное число ребер в графе (зависит от максимального числа вершин NMAX : RMAX=NMAX*(NMAX-1)/2, если граф без петель).

Здесь ребра пронумерованы, начиная с 0.

Внешнее представление графа может отличаться от внутреннего.

Например, граф можно задать в виде количества вершин и последовательности ребер, где каждое ребро - пара смежных вершин:

Задача

Задан граф без петель в виде количества вершин n<=7 и матрицы смежности. Сформировать для этого графа матрицу инцидентности.

Программа

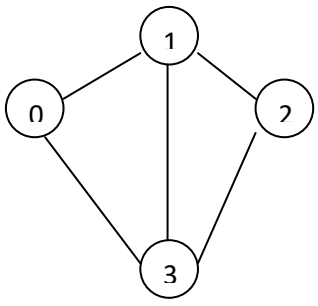
```
#include <stdio.h>
#include <conio.h>
   /* Глобальные данные */
#define  NMAX  7   /* максимальное число вершин графа */
#define  RMAX 21   /* максимальное число ребер        */
int  g1 [NMAX][NMAX] , /* матрица смежности */
    g2 [NMAX][RMAX] , /* м-ца инцидентности */
    n ,             /* количество вершин */
    k ;             /* количество ребер  */
 /*        функция ввода матрицы смежности     */
 void  VVOD_MATR_SM ()
     /*  Глобальные данные:  g1,n */
  { int  i,j ;  /* параметры циклов */
    printf ("Введите матрицу смежности:\n\n");
```

```
   printf (" ¦ ");
   for (j=0; j<n; j++)  printf ("%d ",j);
   putchar ('\n');
   for (i=0; i<2*n+2; i++)  putchar ('-');
   for (i=0; i<n; i++)
     { printf ("\n%d¦ ",i);
       for (j=0; j<n; j++)  scanf ("%d",&g1[i][j]);
     }
   putchar ('\n');
  }
/*     функция вывода матрицы инцидентности    */
void  VIVOD_MATR_IN  ()
    /*  Глобальные данные: g2,n,k */
  { int  i,l ;  /* параметры циклов */
   printf ("Матрица инцидентности\n\n");
   printf (" ¦ ");
   for (l=0; l<k; l++)  printf ("%d ",l);
   putchar ('\n');
   for (i=0; i<2*k+2; i++)  putchar ('-');
   for (i=0; i<n; i++)
     { printf ("\n%d¦ ",i);
       for (l=0; l<k; l++)
         if (l<10) printf ("%d ",g2[i][l]);
            else  printf ("%d  ",g2[i][l]);
     }
   putchar ('\n');
  }
   /*     главная функция     */
void main()
  { int  i,j,l; /* индексы элементов матриц g1,g2 */
```

```
  printf ("\nВведите количество вершин:");
  scanf ("%d",&n);
  VVOD_MATR_SM();   /* ввод матрицы смежности g1 */
     /* Формировавание матрицы инц-ти g2 */
  /* Обнулять м-цу g2 не нужно,  т.к. глобальные и статические
    массивы автоматически инициализирутся нулем ! */
  k=0;
  for (i=0; i<n; i++)
    for (j=i; j<n; j++)
      if (g1[i][j])
        { g2[i][k]=1;
          g2[j][k]=1;
          k++;
        }
  VIVOD_MATR_IN();   /* вывод G2 */
  getch();
}
```

Тесты

1. Исходные данные: n=4 матрица смежности



	0	1	2	3
0	0	1	0	1
1	1	0	1	1
2	0	1	0	1
3	1	1	1	0

Ожидаемый результат: м-ца инцидентности

	0	1	2	3	4
0	1	1	0	0	0
1	1	0	1	1	0
2	0	0	1	0	1
3	0	1	0	1	1

2. Граф, изображенный на рис.1а. n=NMAX=7. Вид матрицы смежности см. на рис.2а. Ожидаемый результат: матрица инцидентности, приведенная на рис. 2б (только индексы столбцов цифровые).

3. Полный граф (все вершины смежны между собой), число вершин максимальное.

Исходные данные:

n=7, матрица смежности:

```
 ¦ 0 1 2 3 4 5 6
 ----------------
0¦ 0 1 1 1 1 1 1
1¦ 1 0 1 1 1 1 1
2¦ 1 1 0 1 1 1 1
3¦ 1 1 1 0 1 1 1
4¦ 1 1 1 1 0 1 1
5¦ 1 1 1 1 1 0 1
6¦ 1 1 1 1 1 1 0
```

Ожидаемый результат:

матрица инцидентности:

```
 ¦ 0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20
-------------------------------------------
0¦ 1 1 1 1 1 1 0 0 0 0 0  0  0  0  0  0  0  0  0  0  0
1¦ 1 0 0 0 0 0 1 1 1 1 1  0  0  0  0  0  0  0  0  0  0
2¦ 0 1 0 0 0 0 1 0 0 0 0  1  1  1  1  0  0  0  0  0  0
3¦ 0 0 1 0 0 0 0 1 0 0 0  1  0  0  0  1  1  1  0  0  0
4¦ 0 0 0 1 0 0 0 0 1 0 0  0  1  0  0  1  0  0  1  1  0
5¦ 0 0 0 0 1 0 0 0 0 1 0  0  0  1  0  0  1  0  1  0  1
6¦ 0 0 0 0 0 1 0 0 0 0 1  0  0  0  1  0  0  1  0  1  1
```

4. В графе нет ребер.

Исходные данные: Ожидаемый результат:

n=3 м-ца смежности м-ца инцидентности

```
 ¦ 0 1 2          ¦
 --------         --
```

0	0 0 0	0
1	0 0 0	1
2	0 0 0	2

Задания

1. Задан граф в виде количества вершин n<=10 и последовательности ребер (каждое ребро задается парой смежных вершин). Получить матрицу смежности.

а) Напечатать матрицу смежности. Проверить, есть ли в графе петли.

б) Напечатать матрицу смежности. Проверить, есть ли в графе вершины, не смежные с другими.

в) Напечатать для каждой вершины номера смежных вершин.

г) Проверить, есть ли в графе вершина, смежная со всеми другими вершинами.

д) Определить степень каждой вершины графа.

е) Напечатать номера вершин со степенью 1.

2. Задан орграф в виде количества вершин n<=10 и последовательности дуг (дуга задается парой “предшественник преемник”).

а) Напечатать номера вершин, имеющих более двух преемников.

б) Напечатать номера вершин, не имеющих предшественников.

в) Для каждой вершины напечатать номера всех предшественников.

г) Проверить, есть ли в графе вершины, имеющие только одного преемника.

3. Задан орграф в виде количества вершин n<=10 и матрицы смежности.

а) Напечатать номера вершин, имеющих и предшественников и преемников.

б) Напечатать список дуг орграфа.

в) Напечатать номер вершины, имеющей наибольшее число преемников.

4. Задан граф без петель в виде количества вершин n<=7, количества ребер k<=21 и матрицы инцидентности.

а) Для каждой вершины напечатать список инцидентных ей ребер.

б) Определить наибольшее число смежных между собой ребер, инцидентных одной и той же вершине.

в) Проверить, есть ли вершины со степенью 0.

г) Напечатать номера вершин, инцидентных только одному ребру.

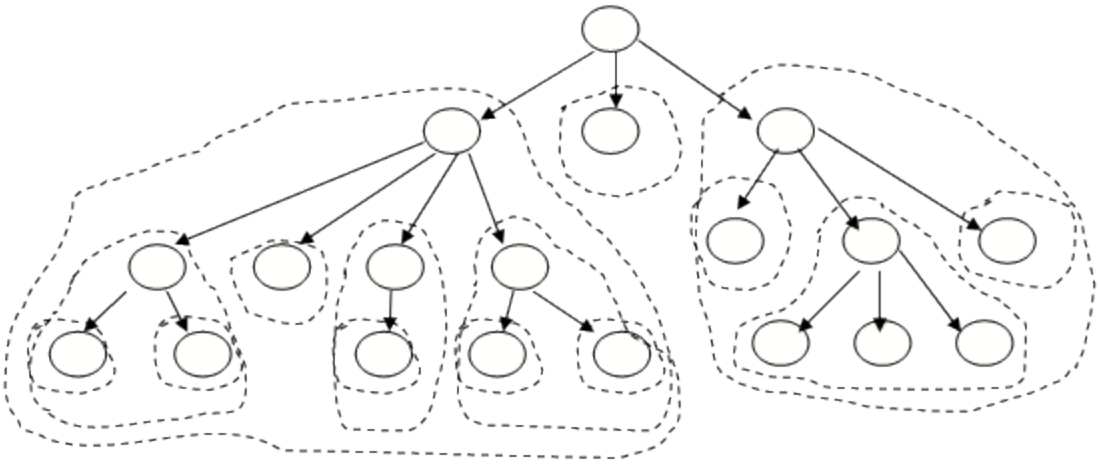
Деревья

Структуры данных типа “дерево” широко используются в программной индустрии. В отличие от списковых структур деревья можно назвать **нелинейными** структурам. Любое дерево состоит из элементов – узлов или вершин, которые по определенным правилам связаны друг с другом. В списковых структурах за текущей вершиной (если она не последняя) всегда следует **только одна** вершина, тогда как в древовидных структурах таких вершин может быть **несколько**. Для дерева определена **единственная** начальная вершина (**корень** дерева) и **множество** конечных (терминальных) вершин (**листьев**). От корня к листьям существует **множество** путей, тогда как в списках (линейных структурах!) путь всегда один.

Математически дерево рассматривается как частный случай графа, в котором отсутствуют замкнутые пути (циклы). Исторически сложилось так, что при описании деревьев используется «генеалогическая» терминология, основанная на понятиях «**родительская**» вершина (parent node) и «**дочерние**» вершины или вершины-«**потомки**» (childnode).

Разветвляющаяся структура дерева затрудняет его формальное определение и здесь на выручку приходит понятие **рекурсии**. Дерево является типичным примером **рекурсивно определённой структуры** данных, поскольку оно определяется в терминах самого себя. Как известно, рекурсия позволяет сводить **сложный** объект к набору более **простых** объектов той же природы. Дерево с N вершинами сводится к **набору** деревьев с **меньшим** числом вершин, каждое из них точно так же сводится к **своему** набору поддеревьев с еще меньшим числом вершин, до тех пор, пока не будут получены **простейшие** деревья с 1 или 0 вершин.

На рисунке ниже показано сведение дерева с 19 вершинами к трем поддеревьям с 10, 1 и 7 вершинами, каждое из которых сводится к своему набору своих поддеревьев. Например, самое большое поддерево с 10 вершинами сводится к четырем поддеревьям с 3, 1, 2 и 3 вершинами.



Формальное рекурсивное определение дерева с базовым типом T задается двумя **правилами**:

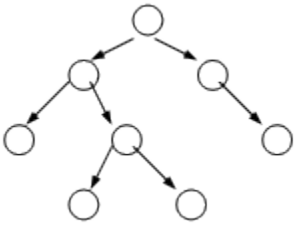
-это либо **пустое** дерево (не содержащее ни одного узла)

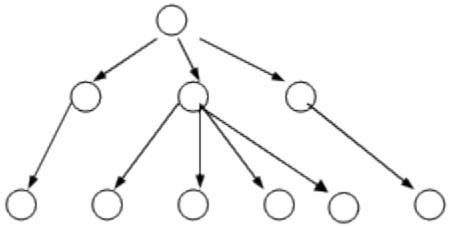
-либо некоторая **вершина** типа T, с которой связано **конечное** число деревьев с тем же базовым типом T, называемых **поддеревьями**

Существует много **разновидностей** деревьев, но, пожалуй, две самые **важные** разновидности возникают в зависимости от **числа** возможных **потомков** у каждой вершины:

Если у **КАЖДОЙ** вершины может быть **не более двух** потомков (т.е. два, один или ноль), то такие деревья называют **двоичными** (бинарными).

Если число потомков у вершины может быть **любым** (естественно – не бесконечным), то дерево называется **недвоичным** или **сильноветвящимся**.

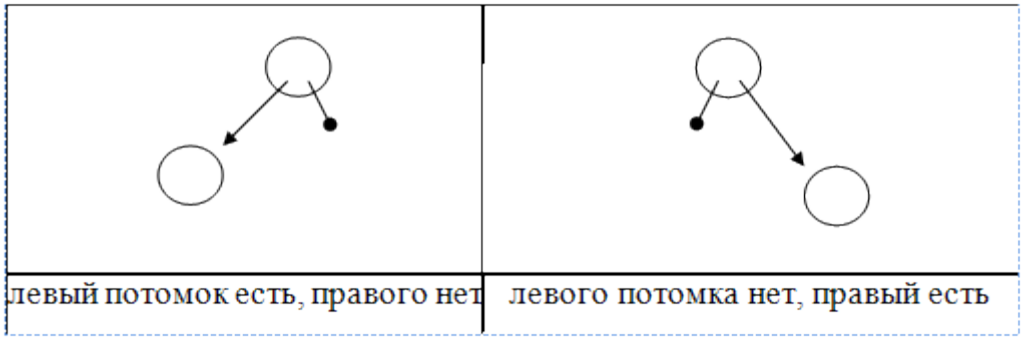




Двоичные деревья имеют большое самостоятельное значение, например – при эффективной организации поисковых операций. Кроме того, двоичные деревья являются одним из способов представления недвоичных деревьев. Поэтому в дальнейшем сначала будут рассмотрены именно двоичные деревья, а затем – недвоичные.

Еще одной разновидностью деревьев являются **упорядоченные** деревья: для них важен **порядок** следования потомков. Для таких деревьев вводится понятие **левый** и **правый** потомок (для двоичных деревьев) или более левый/правый (для недвоичных деревьев).

Пример двух **разных** упорядоченных двоичных деревьев:



При использовании деревьев часто встречаются такие понятия как **путь** между начальной и конечной вершиной (последовательность проходимых ребер или вершин), **высота** дерева (наиболее длинный путь от корневой вершины к терминальным).

Далее рассмотрим общие вопросы, связанные с реализацией **двоичных упорядоченных** деревьев. Каждая вершина такого дерева должна иметь следующие поля:

-одно или несколько **информационных** полей, содержащих обрабатываемые данные (если эти данные достаточно большие по объему, можно рассмотреть вопрос об их **отдельном** от вершины хранении с адресацией соответствующими указателями)

-связующее поле для указания возможного **левого** потомка

-связующее поле для указания возможного **правого** потомка

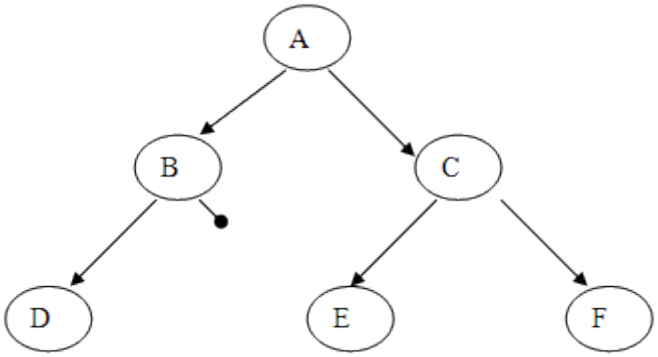
Отсюда следует, что вершина дерева описывается как **запись/структура** с набором необходимых полей. Все вершины одного дерева имеют одну и ту же структуру (однотипны). Двоичное дерево, аналогично линейным структурам, можно реализовать **двумя** способами:

-на основе **массива записей** (статическая или непрерывная реализация)

-на базе механизма динамического распределения памяти и указательных переменных (динамическая реализация)

В первом случае записи-вершины являются элементами массива, **индексируются** порядковыми номерами и связующие поля у каждой вершины должны содержать **номера** ячеек, где находятся левый и правый потомок этой вершины. Если потомка нет, соответствующее связующее поле должно содержать некоторый **фиктивный** индекс (например 0 при индексации элементов в массиве с 1, или -1 при индексации с 0). Для использования такой структуры надо знать индекс размещения в массиве корневой вершины. По умолчанию самое простое – разместить корень в первой ячейке массива.

Пример. Дано небольшое двоичное дерево с символьными элементами:



Тогда его размещение в массиве может выглядеть так:

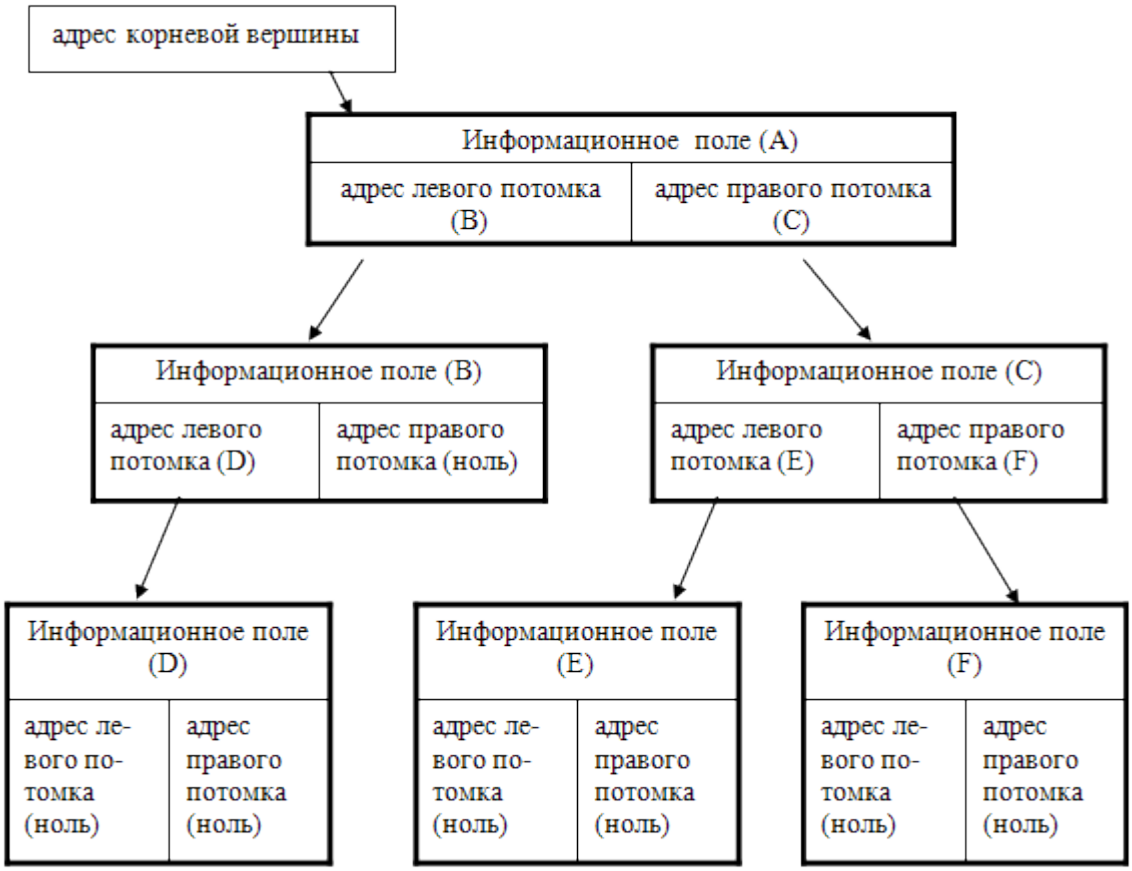
Индекс	1	2	3	4	5	6	7	8	9
Информационное поле	A		C	B	F	E		D	
Индекс левого потомка	4		6	8	0	0		0	
Индекс правого потомка	3		5	0	0	0		0	

Однако чаще двоичные деревья реализуются **динамически**. В этом случае его вершины размещаются в динамически распределяемой памяти и связующие поля содержат **адреса** потомков. Если потомка нет – адресное поле содержит **пустое** (нулевое) значение. Отсюда следует, что у терминальных вершин оба адресных поля нулевые. Это, кстати, один из недостатков динамической реализации: примерно половина вершин в

двоичном дереве являются терминальными и у всех них адресные поля нулевые.

Для доступа к дереву достаточно знать адрес размещения в памяти корневой вершины (аналогично адресу первого элемента списка).

Например, рассмотренное выше дерево с шестью вершинами в динамической реализации можно условно показать так:



Для динамической реализации необходимы **стандартные** объявления:

-описание **структуры** вершин дерева с двумя **обязательными** указательными полями (в Паскале для этого необходим указательный тип)

-описание одной **основной** указательной переменной для адресации **корневой** вершины

Паскаль	Си
type pTreeNode = ^TTreeNode; TTreeNode = **record** info : **char**; left : pTreeNode;	**struct** TreeNode { **char** info; **struct** TreeNode *left; **struct** TreeNode *right;

right : pTreeNode; **end**; **var** pRoot : pTreeNode;	}; **struct** TreeNode *pRoot;

Тогда пустое дерево просто определяется установкой переменной pRoot в нулевое значение (например – **nil**).

Основные операции с двоичными деревьями:

- обход всех вершин дерева в некотором порядке (общая операция)
- добавление новой вершины как потомка заданной вершины
- удаление заданной вершины
- поиск заданной вершины

Последние три операции имеют разную реализацию в зависимости от типа дерева (обычное двоичное дерево или специальное поисковое дерево).

Список литературы

Основная литература

1. Немцова, Т. И. Программирование на языке высокого уровня. Програм-мирование на языке С++ : учебное пособие / Т.И. Немцова, С.Ю. Голова, А.И. Те-рентьев ; под ред. Л.Г. Гагариной. — Москва : ФОРУМ : ИНФРА-М, 2023. — 512 с. + Доп. материалы [Электронный ресурс]. — (Среднее профессиональное обра-зование). - ISBN 978-5-8199-0699-6. - Текст : электронный. - URL: https://znanium.com/catalog/product/1916204 (дата обращения: 28.06.2023). – Режим доступа: по подписке.

2. Рейзлин, В. И. Язык С++ и программирование на нём : учебное пособие / В. И. Рейзлин. — 3-е изд., перераб. — Томск : ТПУ, 2021. — 206 с. — ISBN 978-5-4387-0975-6. — Текст : электронный // Лань : электронно-библиотечная система. — URL: https://e.lanbook.com/book/246239 (дата обращения: 28.06.2023). — Режим доступа: для авториз. пользователей.

3. Ефимова Ю.В. Программирование на языке высокого уровня: Практикум. - Казань: Изд-во Казан. гос. техн. ун-та, 2012. - 32 с.

4. Чукич, И. Функциональное программирование на С++ : учебное пособие / И. Чукич ; перевод с английского В. Ю. Винника, А. Н. Киселева. — Москва : ДМК Пресс, 2020. — 360 с. — ISBN 978-5-97060-781-7. — Текст : электронный // Лань : электронно-библиотечная система. — URL: https://e.lanbook.com/book/140597 (дата обращения: 28.06.2023). — Режим досту-па: для авториз. пользователей.

5. Программирование на языке Си/А.В.Кузин, Е.В.Чумакова - М.: Форум, НИЦ ИНФРА-М, 2015. - 144 с. - Режим доступа: http://znanium.com/bookread2.php?book=505194

Дополнительная литература:

1. Язык Си: кратко и ясно: Учебное пособие / Д.В. Парфенов. - М.: Альфа-М: НИЦ ИНФРА-М, 2014. - 320 с. - Режим доступа: https://znanium.com/read?id=356040

2. Программирование графики на C++. Теория и примеры : учеб. пособие / В.И. Корнеев, Л.Г. Гагарина, М.В. Корнеева. — М. : ИД «ФОРУМ» : ИНФРА-М, 2017. — 517 с. - Режим доступа: http://znanium.com/bookread2.php?book=562914

3. Программирование на языке C++: Учебное пособие / Т.И. Немцова, С.Ю. Голова, А.И. Терентьев; Под ред. Л.Г. Гагариной. - М.: ИД ФОРУМ: ИНФРА-М, 2012. - 512 с. - Режим доступа: http://znanium.com/bookread2.php?book=244875