ПРИЛОЖЕНИЕ 1
к рабочей программе дисциплины

МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ

ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ
«РЯЗАНСКИЙ ГОСУДАРСТВЕННЫЙ РАДИОТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ ИМЕНИ В.Ф. УТКИНА»

Факультет вычислительной техники

Кафедра «Информационная безопасность»

ОЦЕНОЧНЫЕ МАТЕРИАЛЫ

по дисциплине

Б1.О.19 «Языки программирования»

Специальность: 10.05.01 Компьютерная безопасность

Специализация: № 5 Разработка систем защиты информации компьютерных систем объектов информатизации" (по отрасли или в сфере профессиональной деятельности)

Квалификация выпускника: - специалист по защите информации

Форма обучения - очная

Срок обучения — 5,5 лет

Рязань 2021 г.

1 ОБЩИЕ ПОЛОЖЕНИЯ

Оценочные материалы – это совокупность учебно-методических материалов (практических заданий, описаний форм и процедур проверки), предназначенных для оценки качества освоения обучающимися данной дисциплины как части основной образовательной программы (ОПОП).

Цель – оценить соответствие знаний, умений и уровня приобретенных компетенций обучающихся целям и требованиям ОПОП в ходе проведения промежуточной аттестации.

Основная задача – обеспечить оценку уровня сформированности общекультурных и профессиональных компетенций и индикаторов их достижения, приобретаемых обучающимся в соответствии с требованиями ОПОП.

Контроль знаний обучающихся проводится в форме текущего контроля и промежуточной аттестации.

Текущий контроль успеваемости и промежуточная аттестация проводятся с целью определения степени усвоения учебного материала, своевременного выявления и устранения недостатков в подготовке обучающихся, организации работы обучающихся в ходе учебных занятий и оказания им индивидуальной помощи.

К контролю текущей успеваемости относятся проверка знаний, умений и навыков обучающихся на практических занятиях по результатам выполнения и защиты обучающимися индивидуальных заданий, по результатам выполнения контрольных работ и тестов, по результатам проверки качества конспектов лекций и иных материалов.

В качестве оценочных средств на протяжении семестра используется устные и письменные ответы студентов на индивидуальные вопросы, письменное тестирование по теоретическим разделам курса. Дополнительным средством оценки знаний и умений студентов является отчет о выполнении практических заданий их защита.

По итогам курса обучающиеся сдают зачет с оценкой, экзамен и выполняют курсовую работу. Форма проведения зачета и экзамена – устный ответ с письменным подкреплением по утвержденным билетам, сформулированным с учетом содержания дисциплины. В билет для зачета и экзамена включается два теоретических вопроса и задача. В процессе подготовки к устному ответу студент должен составить в письменном виде план ответа.

2 ПАСПОРТ ОЦЕНОЧНЫХ МАТЕРИАЛОВ ПО ДИСЦИПЛИНЕ

№ п/п	Контролируемые разделы (темы) дисциплины	Код контролируемой компетенции (или её части)	Вид, метод, форма оценочного мероприятия
1	2	3	4
1.	Языки базового уровня программирования - Pascal и Object Pascal	ОПК-7 (ОПК-7.1,ОПК-7.2, ОПК-7.3)	Зачет с оценкой
2.	Языки профессионального программирования С и С++	ОПК-13 (ОПК-13.1, ОПК-13.2)	Экзамен
3.	Язык Ассемблера	ОПК-13 (ОПК-13.1, ОПК-13.2)	Экзамен
4.	Скриптовые языки	ОПК-13 (ОПК-13.1, ОПК-13.2)	Экзамен

2 ПЕРЕЧЕНЬ КОМПЕТЕНЦИЙ С УКАЗАНИЕМ ЭТАПОВ ИХ ФОРМИРОВАНИЯ

При освоении дисциплины формируются следующие компетенции: ОПК-7 (ОПК-7.1,ОПК-7.2, ОПК-7.3), ОПК-13 (ОПК-13.1, ОПК-13.2).

Указанные компетенции формируются в соответствии со следующими этапами:

– формирование и развитие теоретических знаний, предусмотренных указанными компетенциями (лекционные занятия, самостоятельная работа студентов);

– приобретение и развитие практических умений предусмотренных компетенциями (практические занятия, самостоятельная работа студентов);

– закрепление теоретических знаний, умений и практических навыков, предусмотренных компетенциями, в ходе решения конкретных задач на занятиях, выполнения индивидуальных заданий на практических занятиях и их защиты, а так же в процессе сдачи зачета и экзамена.

3 ПОКАЗАТЕЛИ И КРИТЕРИИ ОЦЕНИВАНИЯ КОМПЕТЕНЦИЙ (РЕЗУЛЬТАТОВ) НА РАЗЛИЧНЫХ ЭТАПАХ ИХ ФОРМИРОВАНИЯ, ОПИСАНИЕ ШКАЛ ОЦЕНИВАНИЯ

Сформированность каждой компетенции (или ее части) в рамках освоения данной дисциплины оценивается по трехуровневой шкале:

1) пороговый уровень является обязательным для всех обучающихся по завершении освоения дисциплины;

2) продвинутый уровень характеризуется превышением минимальных характеристик сформированности компетенций по завершении освоения дисциплины;

3) эталонный уровень характеризуется максимально возможной выраженностью компетенций и является важным качественным ориентиром для самосовершенствования.

Принимается во внимание наличие и степень сформированности у обучающихся знаний, умений и обладание навыками, которые должны были формироваться в процессе изучения дисциплины.

Уровень освоения компетенций, формируемых дисциплиной:

Описание критериев и шкалы оценивания тестирования:

Шкала оценивания	Критерий
3 балла (эталонный уровень)	уровень усвоения материала, предусмотренного программой: процент верных ответов на тестовые вопросы от 85 до 100%
2 балла (продвинутый уровень)	уровень усвоения материала, предусмотренного программой: процент верных ответов на тестовые вопросы от 70 до 84%
1 балл (пороговый уровень)	уровень усвоения материала, предусмотренного программой: процент верных ответов на тестовые вопросы от 50 до 69%
0 баллов	уровень усвоения материала, предусмотренного программой: процент верных ответов на тестовые вопросы от 0 до 49%

Описание критериев и шкалы оценивания теоретического вопроса:

Шкала оценивания	Критерий
3 балла (эталонный уровень)	выставляется студенту, который дал полный ответ на вопрос, показал глубокие систематизированные знания, смог привести примеры, ответил на дополнительные вопросы преподавателя
2 балла (продвинутый уровень)	выставляется студенту, который дал полный ответ на вопрос, но на некоторые дополнительные вопросы преподавателя ответил только с помощью наводящих вопросов
1 балл (пороговый уровень)	выставляется студенту, который дал неполный ответ на вопрос в билете и смог ответить на дополнительные вопросы только с помощью преподавателя
0 баллов	выставляется студенту, который не смог ответить на вопрос

На промежуточную аттестацию (зачет, экзамен) выносится тест и два теоретических вопроса. Максимально обучающийся может набрать 6 баллов. Итоговый суммарный балл студента, полученный при прохождении промежуточной аттестации, переводится в традиционную форму по системе «отлично», «хорошо», «удовлетворительно» и «неудовлетворительно».

Оценка «отлично» выставляется студенту, который набрал в сумме 6 баллов (выполнил все задания на эталонном уровне). Обязательным условием является выполнение всех предусмотренных в течение семестра лабораторных работ.

Оценка «хорошо» выставляется студенту, который набрал в сумме от 4 до 5 баллов при условии выполнения всех заданий на уровне не ниже продвинутого. Обязательным условием является выполнение всех предусмотренных в течение семестра лабораторных работ.

Оценка «удовлетворительно» выставляется студенту, который набрал в сумме 3 балла. Обязательным условием является выполнение всех предусмотренных в течение семестра лабораторных работ.

Оценка «неудовлетворительно» выставляется студенту, который набрал в сумме менее 3 баллов или не выполнил всех предусмотренных в течение семестра лабораторных работ.

4 ТИПОВЫЕ КОНТРОЛЬНЫЕ ЗАДАНИЯ ИЛИ ИНЫЕ МАТЕРИАЛЫ

4.1. Промежуточная аттестация в форме зачета с оценкой

Код компетенции	Результаты освоения ОПОП Содержание компетенций
ОПК-7 (ОПК-7.1, ОПК-7.2, ОПК-7.3)	Способен создавать программы на языках высокого и низкого уровня, применять методы и инструментальные средства программирования для решения профессиональных задач, осуществлять обоснованный выбор инструментария программирования и способов организации программ ОПК-7.1 Разрабатывает и реализует на языках программирования высокого уровня алгоритмы решения типовых профессиональных задач ОПК-7.2 Планирует разработку программ на языках общего назначения, осуществляет обоснованный выбор инструментария программирования ОПК-7.3 Применяет методы и инструментальные средства программирования для решения профессиональных задач

Типовые практические задания (тесты):

1. **Результатом выполнения фрагмента программы s:=-5;x:=0;repeat s:=s*(x+2);x:=x+1; until x<2;write(s); будет сообщение**

- ☑ -15.0
- ☒ -30.0
- ☒ 0.0
- ☒ -120.0

2. **В написанном ниже участке кода выберите номера обозначенных строк, которые пройдет программа, в случае если будет иметь место «равнобедренный треугольник» (номера строк программы указывать через пробел).**

```
1 if (треугольник существует)
2   then
3       if (треугольник равносторонний)
4           then
5               writeln('Этот треугольник равносторонний')
6           else
7             if (треугольник равнобедренный)
8                 then
9                     writeln('Этот треугольник равнобедренный')
10                else
11                    writeln('Этот треугольник имеет произвольную форму')
12    else
```

13 writeln(‘Такого треугольника не существует’);

14 Оператор

Ответ:

1 2 3 6 7 8 9 14

3. Какую величину вычисляет программа?

```
var   a : array[1..3,1..4] of integer;
var   i,j,k : integer;
begin
      for i := 1 to 3 do
         for j := 1 to 4 do
          read(a[i,j]);
      k:=0;
      for i := 1 to 4 do
        for j := 1 to 3 do
            if a[j,i]=0 then k:=i
      end;
      write(k)
end.
```

	номер первой из строк массива, содержащих хотя бы один элемент, равный 0
	номер первого из столбцов массива, содержащих хотя бы один элемент, равный 0
	количество положительных элементов массива в каждом столбце
	+номер последнего из столбцов массива, содержащих хотя бы один элемент, равный 0
	номер последней из строк массива, содержащих хотя бы один элемент, равный 0

4. Какое число получится в результате работы программы?

```
const   a : array[1..8] of integer = (3,8,0,-6,0,-1,-
9,3);
var   i,k : integer;
begin
      k := a[1];
      for i := 2 to 8 do
   if a[i] >k then k := a[i];
      write(k)
end.
```

9
+8
7
10

5. **Какое число буден выведено в результате работы программы?**

```
    const   a : array[1..8] of integer = (3,8,0,-6,0,-1,-
9,3);
    var   i,k : integer;
    begin
          k := 1;
          for i := 1 to 8 do
       if a[i] = 0 then k := i;
          write(k)
    end.
```

- +5
- 3
- 2
- 4

6. **Даны описания:**

```
type   tarr = array [1 ..10] of real;
       tzap = record
                c : real;
                a : tarr;
                 end;
var    x : array [1 .. 10] of tzap;
```

Укажите синтаксически правильные варианты обращения к полям записи:

- tzap[1].tarr[1]
- x.a[1]
- +x[1].c
- a.x[1]

7. **Указатель p инициализирован строковой константой: char *p = «тестовая строка». Что в данном случае хранится в указателе p?**

- ☑ адрес первого элемента строки
- ☒ указатель p может хранить только адрес и не может быть инициализирован строковой константой
- ☒ заданная строка
- ☒ адрес заданной строки

8. **Определите результат работы участка кода на JavaScript**

```
<SCRIPT type="text/javascript">
```

```
var y = new Date();
var d =  y.getFullYear();
document.write(y);

</SCRIPT>
```

выводится текущий год

прав.ответ выводятся текущий день, месяц, время и год

допущена ошибка — нельзя вывести переменную y

9. Определите результат работы участка кода на JavaScript

```
<SCRIPT type="text/javascript">
  var result=0;
  var x= 12;
  var y= 5;
  result= x + -y;
  alert(result);
</SCRIPT>
```

выводится -x +-y

прав.ответ выводится 7

выводится 19

10. Определите результат работы участка кода на JavaScript

```
<SCRIPT type="text/javascript">
  var x= 5;
  var y= 2;
  var result=0;
  result= x / y;
  alert(result);
</SCRIPT>
```

выводится «2/5″

выводится 2

прав.ответ выводится «2.5″

11. При каких условиях выполнится оператор2 в конструкции if(условие1){if(условие2){оператор1}else{оператор2}} на JavaScript

если условие1 и условие2 верны

если условие1 неверно, а условие 2 верно

прав.ответ если условие1 верно, а условие2 неверно

12. Определите результаты работы сценария на JavaScript в случае, если окно confirm будет закрыто без нажатия каких-либо кнопок

```
<SCRIPT type="text/javascript">
  if (confirm ("Вы уверены, что хотите посетить INTUIT?"))
```

```
    {
        alert ("В данный момент страница INTUIT не доступна");
    }
    else
    {
        window.defaultStatus = "К сожалению, вы закрыли окно выбора";
    }
</SCRIPT>
```

результатом работы сценария будет отображение диалогового окна с сообщением о том, что страница INTUIT в данный момент недоступна.

прав.ответ результатом работы сценария будет отображение в строке состояния браузера надписи - «К сожалению, вы закрыли окно выбора»

конечным результатом работы сценария будет открытие страницы INTUIT

13. Что будет выведено на экран, когда завершится выполнение следующей программы?

```
#include<iostream>
using namespace std;

double & fun(double &x, double &y)
{
    return x < y ? x : y;
}

void main()
{
    double a = 5.1, b = 4.3;
    fun(a, b) = 20.7;
    cout << a << "\t" << b << endl;
}
```

TRUE

5.1 20.7

FALSE

5.1 4.3

FALSE

20.7 4.3

FALSE

4.3 5.1

14. Что будет выведено на экран, когда завершится выполнение следующего фрагмента программы?

```
long *L = new long;
*L = 4;
delete L;
cout << *L;
```

TRUE

случайное число

FALSE

4

FALSE

5

FALSE

ошибка компиляции

15. Что будет выведено на экран в результате выполнения следующей программы?

```
#include <iostream.h>
void main()
{
      char A[] = "ABC";
      char *U = &A[2];
      cout << "\n" << *U--;
      cout << *U-- ;
      cout << *U;
}
```

TRUE

CBA

FALSE

ABC

FALSE

BC

FALSE

BCA

16. Который из следующих операторов будет ТОЧНО воспроизводить указанный ниже текст?

```
"The stock's value decreased by 10%."
```

TRUE

```
printf("\"The stock's value decreased by %d%%.\"\n",10);
```

FALSE

```
printf("The stock's value decreased by %d'%'.\n");
```

FALSE

```
printf("The stock's value decreased by %d%.\n", 10);
```

FALSE

```
printf("\"The stock\'s value decreased by %d%\.\"\n");
```

17. Что будет выведено на экран в результате выполнения следующего фрагмента программы?

```
int f1(int & x) { return x += 5; }
int f2(int * y) { return *y = *y + 1; }
main()
{
      int a = 7, *k = new int;
      *k = f1(a);
      cout << a << " " << *k << " " << f2(k);
      delete k;
}
```

TRUE

12 13 13

FALSE

```
7 12 13
```
FALSE
```
12 12 13
```
FALSE
```
7 13 13
```

18. Функция вычисления факториала числа n имеет прототип

```
int fact(int n);
```

Как правильно записать определение тела функции, используя рекурсию?

TRUE
```
{   if (n == 1) return 1;
    else return n*fact(n -1); }
```
FALSE
```
{   if (n == 1) return 1;
    else return fact(n);      }
```
FALSE
```
{   if (n == 1) return 1;
    else return fact(n)*fact(n-1);   }
```
FALSE
```
{   if (n == 1) return 1;
    else return fact(n)*(n-1);    }
```

19. Что будет выведено на экран в результате выполнения программы?

```
#include <stdio.h>

void increm(int n, int &k);

int main()
{
    int n = 5;
    int k = 10;
    increm(n, k);
    printf("n = %d &k = %d\n", n, k);
    return 0;
}

void increm ( int n, int &k )
{
    k -= 5;
    --n;
}
```

TRUE

n = 5 &k = 5

FALSE

n = 5 &k = 10

FALSE

Произойдет ошибка во время компиляции

FALSE

n = 5 &k = <адрес переменной в памяти>

20. Что будет выведено на экран, когда завершится выполнение следующей программы?

```
int m = 5;
namespace space1
{
    int x1 = 3;
    namespace space2
    {
        int x1 = 2 + ::m + space1::x1;
    }
}
main()
{
    int x3 = space1::space2::x1 * 2;
    {  int x3 = 10;  }
    cout << x3;
}
```

TRUE

20

FALSE

13

FALSE

10

FALSE

ошибка компиляции

21. Пусть в глобальной области даны определения:

```
int a = 7;
void f() { static int a; a += 5; }
```

и в функции main() выполняется последовательность операторов:

```
f();
cout << a;
```

Что в результате будет выведено на экран?

TRUE

7

FALSE

5

FALSE

12

FALSE

не определено

22. Что будет результатом макроподстановки

```
#define FUNC(a, b, c, d) a(b c d)
```

в случае макровызова:

```
FUNC(cos, x, +, y)
```

TRUE

```
cos(x + y)
```

FALSE

```
cos(x+y)
```

FALSE

```
cos(b c d)
```

FALSE

```
cos(bcd)
```

23. Что будет выведено на экран в результате выполнения следующего фрагмента программы?

```
#define OUTPUT(x) cout<<x
#define F(a) (a)*5
#include <iostream.h>

void main()
{
    int m = 1;
    OUTPUT(F(m+m));
}
```

TRUE

10

FALSE

6

FALSE

5

FALSE

F(m+m)

24. Что будет выведено на экран в результате выполнения следующего фрагмента программы?

```
#define OUTPUT(a) cout<<a
#define Q(x) x*x
#include <iostream.h>

void main()
{
    int p = 2;
    OUTPUT(Q(p+5));
}
```

TRUE

17

FALSE

49

FALSE

9

FALSE

Q(p+5)

Типовые теоретические вопросы:

1. Что такое компиляция программы?
2. Какой вид имеет минимальная программа на Паскале?
3. Что включает в себя понятие типа данных?
4. Что такое порядковые типы данных?
5. Зачем нужно в программе указывать типы данных переменных?

6. Какие типы данных в Паскале называются структурированными и в чем их особенности?

7. Каковы типы констант в Паскале и правила их записи?

8. Чем типизированные константы в Паскале отличаются от обычных констант?

9. Какую структуру имеет простейшая линейная программа на Паскале?

10. Какую структуру имеет на Паскале программа с подпрограммими?

11. Зачем нужны точки останова и режим отладки в программе ?

12. Что такое приведение типа и зачем оно нужно?

13. Что называется выражением и какие виды выражений возможны в Паскале?

14. Каков порядок выполнения оператора присваивания и как определяется тип присваиваемого значения?

15. Как очистить (заполнить нулями) определенные разряды (фрагмент) двоичного кода?

16. Как установить (заполнить единицами) определенные разряды (фрагмент) двоичного кода?

17. Как проинвертировать определенные разряды (фрагмент) двоичного кода? Как проинвертировать все разряды двоичного кода?

18. Как проверить, какое значение (0 или 1) находится в определенном разряде двоичного кода?

19. Как хранятся строки в памяти в программе на Паскале?

20. Каковы особенности интерпретации входного потока для переменных (из списка ввода) разного типа?

21. Что такое бесформатный ввод и вывод?

22. Что такое рекуррентная формула и где и для чего она используется?

23. Что такое подготовка цикла и что в нее включается? Когда при разработке циклов восходящим методом подготовка цикла нужна, а когда - нет?

24. Для чего служат заголовок и тело цикла?

25. Какие задачи приводят к появлению циклов в программе?

26. Что такое зависимая и независимая ситуации в разветвляющихся алгоритмах?

27. Какой вид должно иметь описание ситуации и как идет уточнение ситуаций в разветвляющихся алгоритмах?

28. Чем последовательности отличаются от массивов?

29. Чем отличаются циклы с неизвестным числом повторений от циклов с известным числом повторений?

30. Как хранятся массивы в оперативной памяти?

31. Что такое «свободный массив» («динамический массив»)? Чем он отличается от «открытого массива»?

32. В чем состоит удобство использования множества в программе?

33. Какие известны способы записи повторяющихся действий в программе?

34. Что понимается под блочной структурой программы и в чем состоит особенность использования блоков в языке Паскаль?

35. Что такое область видимости и время жизни переменной программы?

36. Какие существуют механизмы (способы) передачи данных в подпрограммы на языке Паскаль? В чем состоят особенности этих механизмов?

37. В чем состоит побочный эффект при использовании подпрограмм?
38. Что такое опережающее описание подпрограмм и зачем оно нужно?
39. Что такое рекурсия и итерация? В чем заключаются достоинства и недостатки рекурсивных подпрограмм?
40. Зачем и как передавать подпрограммы как параметры в подпрограмму на Паскале?
41. В чем состоят сходство и различие между записями и массивами?
42. Что такое записи с вариантами и каковы способы их использования?
43. Какие типы файлов есть в языке Паскаль и в чем их особенности?
44. Что такое логический и физический файлы, способ организации и метод доступа?

24.1.1.1. 20. Как производится позиционирование на нужную компоненту файла: а) для текстовых файлов; б) для типизированных файлов; в) для нетипизированных файлов?

45. Как в программах на Паскале выполняется обработка ошибок ввода-вывода?
46. Зачем нужны модули в программах на Паскале. Какие есть средства языка Паскаль, аналогичные модулям?
47. Каковы правила видимости для программ, использующих модули? Как можно организовать скрытие данных в модуле?
48. Чем отличаются типизированные указатели от нетипизированных?
49. Как выполняется операция разыменования указателя?
50. Что такое динамические переменные, какие они бывают и в чем их отличия от обычных переменных?
51. Какие возможны ошибки при работе с указателями?

4.2. Промежуточная аттестация в форме экзамена

Код компетенции	Результаты освоения ОПОП Содержание компетенций
ОПК-13 (ОПК-13.1, ОПК-13.2)	Способен разрабатывать компоненты программных и программно-аппаратных средств защиты информации в компьютерных системах и проводить анализ их безопасности я ОПК-13.1 Применяет языки программирования для разработки компонентов программного обеспечения компьютерных систем ОПК-13.2 Разрабатывает компоненты программного обеспечения компьютерных систем в интегрированных средах разработки программ

Типовые практические задания (тесты):

1. Что увидит пользователь, выполнив данный участок кода на JavaScript при a=10?

```
 if (a == 5)
{
 alert("Сообщение")
 } else
 {
 if (a ==10)
```

```
 {
  if (a==8)
  {
   alert("Предупреждение")
  } else
  {
   alert("Вопрос")
  }
 } else
 {
  alert("Вопрос")
 }
}
```

прав.ответ вопрос

ничего

сообщение

предупреждение

2. Что увидит пользователь, выполнив данный участок кода на JavaScript при a=10?

```
if (a == 5) {
 alert("Сообщение")
} else {
 if (a == 10) {
  alert("Предупреждение")
 } else {
  alert("Вопрос")
 }
}
```

прав.ответ предупреждение

вопрос

сообщение

ничего

3. Что будет выведено на экран, когда завершится выполнение следующей программы?

```
#include <stdio.h>
int count = 0;
int main (int argc, char * argv[])
{
    while (1) {if (count > 5) break; ++count;} {
        int count = 0;
        while (1) {if (count > 20) break; ++count;}
    }
    printf("Count = %d\n", count);
    return 0;
}
```

TRUE

```
Count = 6
```

FALSE

```
Count = 0
```

FALSE

```
Count = 5
```

FALSE

```
Count = 21
```

4. Что будет выведено на экран в результате выполнения следующих операторов?

```
int a = 5^3;
float b = 1.5f;
b += --a/2;
cout << b;
```

TRUE

3.50

FALSE

4.00

FALSE

63.00

FALSE

63.50

5. Укажите, какие значения будут иметь переменные m и n после выполнения следующих операторов?

```
int m = 2, n = 5;
while (m <= 3)
{
    while (m <= n) { n = m; break; }
    break;
    ++m;
}
```

TRUE

m = 2; n = 2;

FALSE

m = 2; n = 10;

FALSE

m = 3; n = 2;

FALSE

m = 10; n = 2;

6. Какую величину вычисляет программа?

```
#include <iostream.h>
void main() {
    int matr[5][3];
    int i, j, k;
    for (i = 0; i < 5; ++i)
        for (j = 0; j < 3; ++j)
            cin >> matr[i][j];
    k = -1;
    for (i = 0; i < 3; ++i) {
        for (j = 0; j < 5; ++j)
            if (!matr[j][i]) { k = i; break; }
    }
    cout << k;
}
```

TRUE

номер последнего из столбцов массива, содержащих хотя бы один элемент, равный 0

FALSE

номер первого из столбцов массива, содержащих хотя бы один элемент, равный 0

FALSE

номер первой из строк массива, содержащих хотя бы один элемент, равный 0

FALSE

номер последней из строк массива, содержащих хотя бы один элемент, равный 0

7. Что будет содержать переменная str после выполнения следующих операторов?

```
char str1[] = "Hello";
char str2[] = "World";
char str[20];
strcpy(str, str1);
strcpy(str + 2, str2 + 2);
```

TRUE

`"Herld"`

FALSE

`"llorld"`

FALSE

`"HeWorld"`

FALSE

`"HelloWorld"`

8. Какой из следующих вариантов является корректным способом конкатенации двух строк s1 и s2?

TRUE

```
char * s3;
s3 = (char *)malloc(strlen(s1) + strlen(s2) + 1);
strcpy(s3, s1);
strncat(s3, s2, strlen(s2));
```

FALSE

```
char * s3;
s3 = s1 + s2;
```

FALSE

```
char * s3;
strcpy(s3, s1);
strcat(s3, s2);
```

FALSE

```
String s3 = NULL;
s3 = s1 + s2;
```

9. Задан фрагмент программы с описанием структуры, содержащей битовые поля:

```
#include<iostream>
using namespace std;
struct
{
    int a:5;
    int b:10;
} xx, *px;

void main()
{
    px = &xx;
     . . .
}
```

Выберите вариант неправильного использования оператора присваивания вместо многоточия.

TRUE

```
    &px.b = 48;
```

FALSE

```
    xx.b = 48;
```

FALSE

```
    px->b = 48;
```

FALSE

```
    (*px).b = 48;
```

10. Что будет выведено на экран, когда завершится выполнение следующей программы?

```
#include<iostream>
using namespace std;

union mix
{
    short b[2];
    long c;
};
void fun1(mix &arg)
{
    arg.b[0] *= 2;
}
void fun2(mix arg)
{
    arg.b[0] *= 2;
}
void main()
{
    mix mob;
    mob.c = 10;
    mob.b[0] = 1;
    mob.b[1] = 0;
    fun1(mob);
    fun2(mob);
    cout << mob.c << "\t" << mob.b[0];
    cout << "\t" << mob.b[1] << endl;
}
```

!TRUE

2 2 0

!FALSE

4 4 0

!FALSE

10 4 0

!FALSE

10 2 0

11. Что будет выведено на экран, когда завершится выполнение следующего фрагмента программы?

```
void Output(int *p)
{
    printf("element = %d\n", *(p + 3));
}

int main()
{
    int ar[3][3] = { { 0 }, { 2, 3 }, { 4, 5, 6 } };
    Output((int *) ar);
    return 0;
}
```

TRUE

element = 2

FALSE

element = 0

FALSE

element = 4

FALSE

element = 5

12. Что будет выведено на экран, когда завершится выполнение следующего фрагмента программы?

```
int a[4] = { 1, 2, 3, 4 };
int * p = a;
cout << (*p+2) + *p;
```

TRUE

4

FALSE

6

FALSE

10

FALSE

адрес ячейки памяти

Типовые теоретические вопросы:

1. В чет отличие структуры программы на языках Си и Паскаль?
2. Какие виды и типы констант имеются в языке Си?
3. Что такое модификатор типа?
4. В чем состоит разница в реализации блока в Си и Паскаль?
5. Зачем нужны операции «запятая» и «условие» в Си?
6. В чем разница между форматным и бесформатным вводом и выводом в Си?
7. В чем состоит особенность циклов в Си по сравнению с Паскалем?

8. Какие соглашения о вызовах возможны в программе на Си?
9. Как реализовать процедуру (по аналогии с Паскалем) в Си?
10. Как передать значение по адресу в функцию на Си?
11. Зачем нужны прототипы функций?
12. В чем состоит разница между префиксными и потоковыми функциями для работы с файлами в Си?
13. Что такое текстовый и двоичный режимы доступа к файлу (потоку)?
14. Какие типичные способы представления строк в программе на Си какие типичные ошибки при работе со строками?
15. Как в программе на Си определить тип символа?
16. Зачем нужны классы памяти в программе на Си?
17. Зачем нужны указатели на функции в Си?
18. Какая структура программы на Ассемблере?
19. Как выполняется описание данных в программе на Ассемблере?
20. Какие типы адресации используются в Ассемблере?
21. Как представляются типичные конструкции языков программирования (разветвления, циклы, подпрограммы, ...) в программе на Ассемблере?
22. Как связать программу на Си с подпрограммой на Ассемблере?
23. Как используется стек при вызове подпрограммы на Ассемблере?
24. Какова структура программы на языке JavaScript?
25. Как вставить программу на JavaScript в Web-страницу?
26. В чем отличие php-страницы и html-страницы?
27. Что возвращает web-сервер при запросе php-страницы?
28. Для чего используется в PHP цикл foreach(), какие существуют виды этого цикла?

4.3. Курсовая работа (ОПК-7 (ОПК-7.1, ОПК-7.2, ОПК-7.3), ОПК-13 (ОПК-13.1, ОПК-13.2))

Тематика курсовой работы: «Разработка прикладных программ на языках Pascal и C)». В рамках курсовой работы обучающиеся должны выполнить:

- анализ задания
- постановку задачи
  а) определить функции программы
  б) определить интерфейс программы
  в) выбрать необходимые структуры данных
  г) выбрать (разработать) метод решения
- разработку внутренней (физической) структуры программы
- разработку алгоритмов подпрограмм
- разработку программы (с описанием мер по защите от ошибок)
- тестирование программы
- разработать программную документацию (схемы алгоритмов, руководство программиста, руководство оператора).

Задания условно делятся на следующие группы:

а) обработка текста

1. Разработать программу форматирования текста, читаемого из файла и состоящего из строк разной длины. Форматирование должно осуществляться путем распределения пробелов между словами, чтобы выровнять строки по ширине. Переносы слов не допускаются. Для форматирования необходимо использовать следующие параметры (ключи запуска):

- требуемая ширина строки,
- число строк на странице;
- размер отступа слева и справа.

Полученный текст должен быть сохранен в новом текстовом файле. Кроме того, программа должна подсчитывать количество слов в каждой строке и сохранять эту информацию в конце нового файла, соблюдая правила форматирования.

2. Подготовить текст программы, написанной на языке Си, к выводу на принтер, для чего выполнить следующие действия:

- выровнять текст по ширине;
- заменить символы табуляции на заданное число пробелов;
- добавить в каждую строку знак "конец строки";
- разбить текст на страницы заданного размера с помощью вставки в текст символа "жесткий" разделитель страниц.

Параметры (ключи) запуска:

- размер страницы (число строк на странице);
- число пробелов для знака табуляции;
- размер отступа слева.

б) шифрование

1. Чтобы зашифровать текст, записанный с помощью букв (русских или английских) и знаков препинания (в виде «зпт» или «тчк»), его нужно переписать, заменив каждую букву на другую, следующую (предшествующей) за исходной (заменяемой) по алфавиту на (правее или левее) n букв, где n - заданное натуральное число. Удобно при этом представить, что буквы выписаны по кругу, как цифры на циферблате, так что за буквой я следует буква а, а за буквой z следует буква а. Если буква в слове повторяется подряд несколько раз, то группа одинаковых букв заменяется на одну и за ней число таких букв. Необходимо написать программу, которая умеет зашифровывать данный текст и расшифровывать его.

Параметры (ключи запуска):

- имя входного файла;
- алфавит текста - русский или английский;
- параметр *n* - размер сдвига букв;
- действие - зашифровать или расшифровать;
- имя выходного файла.

2. Шифрование текста с помощью решетки (решетка Кардано) заключается в следующем. Решетка, т.е. квадрат из клетчатой бумаги 10×10 клеток, некоторые клетки в котором вырезаны, совмещается с целым (без прорезей) бумажным квадратом 10×10 клеток и через прорези на бумажный квадрат наносятся первые буквы зашифровываемого текста. Затем решетка поворачивается на 90° и через прорези записываются следующие буквы. Это повторяется еще раз, потом еще раз и т.д.. Таким образом, на бумагу будут нанесены все 100 букв текста (из которых только те, которые записаны в самом начале представляют исходное сообщение). Решетку можно изображать квадратной матрицей порядка 10 из нулей и единиц (нуль изображает прорезь). Матрица aij (i=1..10, j=1..10) может служить ключом шифрования.

Дано:

- текст;
- матрица-ключ.

Необходимо написать программу, которая умеет:

- разбивать заданный текст на последовательности из 100 букв и их зашифровывать;
- расшифровывать заданный текст.

в) кодирование

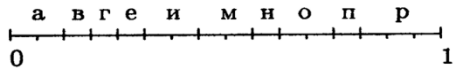
1. При групповом кодировании (RLE) заменяют повторяющиеся группы одинаковых входных символов на пару (количество, символ). Например, строка “aaaabbbbbc” будет закодирована в последовательность 4a5b1c. Ясно, что сжатие будет эффективным, если во входном потоке много длинных групп из одинаковых символов.

Разработать программу кодирования и декодирования файлов на основе следующего варианта RLE кодирования. Для повторяющихся групп длиной от 2 до 127 символов выходным кодом являются два байта (количество, символ). Для групп не повторяющихся символов длиной *n* < 127 байтов выходным кодом является n+1 байт (n+128, исходная группа). Более длинные группы в обоих случаях разбиваются на части длиной не более 127 символов, и каждая кодируется отдельно.

2. При арифметическом кодировании выполняется сопоставление цепочкам символов некоторой (конечной) двоичной дроби, которая строится на основе априорной информации о частотах появления каждого символа в исходном файле. Рассмотрим идею алгоритма на примере кодирования слова “программирование”. Частотный анализ дает следующий результат:

символ	количество	частота
а	2	0.001_2
в	1	0.0001_2
г	1	0.0001_2
е	1	0.0001_2
и	2	0.001_2
м	2	0.001_2
н	1	0.0001_2
о	2	0.001_2
п	1	0.0001_2
р	3	0.0011_2

Разобьем отрезок [0,1) на части, пропорциональные этим частотам и сопоставим каждой части соответствующий символ



Пусть элементы массивов lb[i] и rb[i] задают границы этих частей для символа с кодом i.

Опишем алгоритм построения кодирующей дроби:

```
left = 0;
right = 1;
while ( (k=NextSym()) != EOI )
 {
  d = right - left;
  right = left + d*rb[k];
  left  = left + d*lb[k];
 }
```

По окончании этого цикла в качестве кодирующей дроби можно взять любое число x, удовлетворяющее условию

left <= x < right.

Кодирование по сути представляет собой построение системы вложенных отрезков, где каждый следующий занимает в предыдущем то место, какое отведено данной букве в исходном разбиении отрезка [0,1]. Для обозначения конца кодируемой информации можно дополнительно передавать общее число исходных символов, либо заканчивать входной поток специальным символом (маркером) конца файла EOF. Для слова “программирование” первые несколько отрезков имеют в двоичном представлении следующий вид:

буква	left	right
п	0.1100	0.1101
р	0.11001101_2	0.11010000_2
о	0.110011101111_2	0.11001111010_2
г	0.1100111011111001_2	0.1100111011111100_2

а окончательное значение границ left и right есть

```
  left   = 0.11001110111101101111000111110101110010011111111110001,
  right = 0.11001110111101101111000111110101110010011111111110010.
```

Таким образом, исходная 16-и байтовая строка сжалась до 51 бита.

Декодирование производится обращением описанного выше про цесса:

```
  x = <кодирующая дробь>
  n = <общее число символов в исходном файле>
while (n- - )
 {
    k = Interval (x); /* найти номер интервала» содержащего x */
    Output(Symbol (k)); /* вывести к-й символ */
    d = rb[k]-lb[k];   /* длина интервала для к-го символа */
    x = x - lb[к]; /* преобразуем дробь для определения */
    x /= d;         /*              следующего символа                */
 }
```

Для выполнения кодирования и декодирования следует сначала определить таблицу частот символов в исходном файле. Таким образом, кодирование требует двух просмотров файла, а для декодирования нужна таблица частот, которую приходится помещать в сжатый файл.

Написать программу для кодирования и декодирования по алгоритму алгоритмического кодирования.

Идеи реализации. Следует заранее задать ограничение на длину двоичного представления кодирующей дроби (например, 6 байт).

Вычисление дроби ведется до тех пор, пока различие между битами двоичного представления границ l e f t и rig h t находится в пределах этого ограничения. Как только различие между представлениями выйдет из заданных границ, полученная дробь выводится в выход- ной поток и формирование дроби для следующих символов начинается сначала.

3. Алгоритм Хаффмена (иногда его называют CCITT кодирование) состоит в сопоставлении каждому символу входного потока некоторого кода переменной длины так, что наиболее часто встречающиеся символы кодируются наиболее короткими кодами. Для этого строится бинарное дерево (дерево Хаффмена), в концевых вершинах которого располагаются все возможные символы, а ветви, ведущие к этим вершинам, соответствуют кодам символов. Для построения дерева нужно иметь таблицу количеств появления каждого символа в исходном файле. Каждая вершина дерева имеет свой вес, который для концевых вершин равен количеству появлений символа, записанного в этой вершине.

Процесс построения дерева описывается следующей схемой.

1. Создаем концевые вершины, содержащие все символы входного алфавита, задаем каждой вершине вес в соответствии с таблицей количеств, объявляем все эти вершины свободными.

2. В множестве свободных вершин находим две вершины (обозначим их A и B), имеющие наименьшие веса, создаем новую родительскую вершину C , присоединяем A и B к C справа и слева, устанавливает вес *C* равным сумме весов A и B, исключаем *A* и B из множества свободных вершин, добавляем *C* к множеству свободных вершин.

3. Если в множестве свободных вершин более одного элемента, то перейти к п.2, иначе единственный оставшийся элемент есть корень дерева Хаффмена.

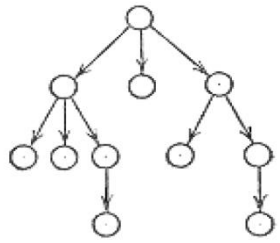
Далее, левым ребрам построенного дерева сопоставляется число 0, а правым ребрам - 1. Теперь, спускаясь от корня к конкретной концевой вершине и последовательно выписывая нули и единицы, сопоставленные проходимым ребрам, мы получаем код символа в этой вершине.

Кодирование и декодирование осуществляется заменой каждого символа его кодом и наоборот. При практической реализации биты кодов символов выписываются подряд, а затем делятся на группы по 8 бит (байты), которые и выдаются в выходной поток. При декодировании входной поток рассматривается как последовательность битов, определяющих направления спуска по ветвям дерева Хаффмена. При достижении концевой вершины соответствующий символ выдается в выходной поток и спуск опять начинается от корня.

Написать программу, реализующую кодировщик и декодировщик файлов по алгоритму Хаффмена. Таблица количеств появлений символов также должна сохраняться в сжатом файле.

г) реализация сложных структур данных

1. Формально дерево (рис. ниже)

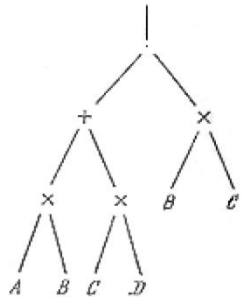


определяется как конечное множество *T*, состоящее из одного или более узлов таких, что

1) имеется один узел, называемый корнем дерева;

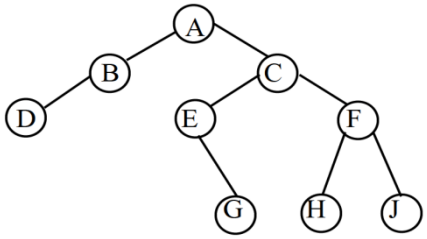
2) остальные узлы (исключая корень) содержатся в $m > 0$ попарно непересекающихся множествах *T*1, ..., *Tm* , каждое из которых в свою очередь является деревом; деревья *T*1, ..., *Tm* называются поддеревьями данного корня.

Деревья, как правило, дают хорошее представление о структурных отношениях между элементами данных. Так, например, на рис. ниже показано дерево, представляющее формулу (*AB* + *CD*)/*BC*.



Здесь ветвь, отходящая от вершины влево, представляет числитель дроби, а ветвь, отходящая вправо, - ее знаменатель и т.д.

Мы будем рассматривать только бинарные деревья. Бинарное дерево определяется как конечное множество узлов, которое либо пусто, либо состоит из корня и двух бинарных деревьев (рис. ниже).



При работе с древовидными структурами наиболее часто приходится решать задачу обхода дерева - такого последовательного прохождения по узлам дерева, когда каждый узел встречается ровно один раз. Для обхода бинарного дерева можно воспользоваться одним из трех способов - можно проходить узлы:

- в префиксном порядке,
- в инфиксном порядке,
- в постфиксном порядке.

Префиксный порядок обхода дерева определяется в виде списка проходимых узлов следующим образом. Если дерево не пусто, то префиксный порядок - это:

- корень дерева;
- узлы левого поддерева в префиксном порядке;
- узлы правого поддерева в префиксном порядке.

Инфиксный порядок обхода дерева определяется следующим образом. Если дерево пусто, список узлов пуст. Если дерево не пусто, инфиксный порядок - это:

- узлы левого поддерева в инфиксном порядке;
- корень дерева;
- узлы правого поддерева в инфиксном порядке.

Постфиксный порядок обхода дерева определяется следующим образом. Если дерево пусто, список

узлов пуст. Если дерево не пусто, то постфиксный порядок - это:

- узлы левого поддерева в постфиксном порядке;
- узлы правого поддерева в постфиксном порядке;
- корень дерева.

Составить программу для:

- обхода заданного бинарного дерева в префиксном, инфиксном и постфиксном порядке;
- подсчета числа узлов заданного бинарного дерева.

2. Реализовать программу, работающую со стеками.

Структура *«стек»* (stack) моделируется цепочкой связанных узлов-записей типа TNode. Поле Next последнего элемента цепочки равно nil. *Вершиной стека* (top) считается первый элемент цепочки. Для доступа к стеку используется указатель на его вершину (для пустого стека данный указатель полагается равным nil). *Значением* элемента стека считается значение его поля Data.

Реализовать в программе следующие действия (функции):

а) даны указатели P_1 и P_2 на вершины двух непустых стеков. Надо переместить все элементы из первого стека во второй (в результате элементы первого стека будут располагаться во втором стеке в порядке, обратном исходному) и вывести адрес новой вершины второго стека. При перемещениях операции выделения и освобождения памяти не использовать;

б) даны указатели P_1 и P_2 на вершины двух непустых стеков. Надо перемещать элементы из первого стека во второй, пока значение вершины первого стека не станет четным (перемещенные элементы первого стека будут располагаться во втором стеке в порядке, обратном исходному). Если в первом стеке нет элементов с четными значениями, то переместить из первого стека во второй все элементы. Вывести адреса новых вершин первого и второго стека (если первый стек окажется пустым, то вывести для него константу nil). Операции выделения и освобождения памяти не использовать;

в) дан указатель P_1 на вершину непустого стека. Создать два новых стека, переместив в первый из них все элементы исходного стека с четными значениями, а во второй — с нечетными (элементы в новых стеках будут располагаться в порядке, обратном исходному; один из этих стеков может оказаться пустым). Вывести адреса вершин полученных стеков (для пустого стека вывести nil). Операции выделения и освобождения памяти не использовать;

г) дан указатель P_1 на вершину стека (если стек пуст, то P_1 = nil). Также дано число N (> 0) и набор из N чисел. Описать тип TStack — запись с одним полем Top типа PNode (поле указывает на *вершину стека)* — и процедуру Pushe(S,D), которая добавляет в стек S новый элемент со значением D (S — входной и выходной параметр типа TStack, D — входной параметр целого типа). С помощью процедуры Push добавить в исходный стек данный набор чисел (последнее число будет вершиной стека) и вывести адрес новой вершины стека.

д) дан указатель P_1 на вершину стека, содержащего не менее пяти элементов. Используя тип TStack , описать функцию Pop(S) целого типа, которая извлекает из стека S первый (верхний) элемент, возвращает его значение и освобождает память, которую занимал извлеченный элемент (S — входной и выходной параметр типа TStack). С помощью функции Pop извлечь из исходного стека пять элементов и вывести их значения. Вывести также указатель на новую вершину стека (если результирующий стек окажется пустым, то этот указатель должен быть равен nil).

е) дан указатель P_1 на вершину стека. Используя тип TStack (см. задание Pointerll), описать функции StacklsEmpty(S) логического типа (возвращает True, если стек S пуст, и False в противном случае) и Peek(S) целого типа (возвращает значение вершины непустого стека S, не удаляя ее из стека). В обеих функциях переменная S является входным параметром типа TStack. С помощью этих функций, а также функции Pop из задания Pointerl2, извлечь из исходного стека пять элементов (или все содержащиеся в нем элементы, если их менее пяти) и вывести их значения. Вывести также значение функции StacklsEmpty для результирующего стека и, если результирующий стек не является пустым, значение и адрес его новой вершины.

3. Реализовать программу работающую с очередями.

Структура *«очередь»* (queue) моделируется цепочкой связанных узлов-записей типа TNode . Поле Next последнего элемента цепочки равно nil. *Началом очереди* («головой», head) считается первый элемент цепочки, *концом* («хвостом», tail) — ее последний элемент. Для возможности быстрого добавления в конец очереди нового элемента удобно хранить, помимо указателя на начало очереди, также и указатель на ее конец. В случае пустой очереди указатели на ее начало и конец полагаются равными nil. Как и для стека, *значением* элемента очереди считается значение его поля Data.

Реализовать в программе следующие действия (функции):

а) дан набор из l0 чисел. Создать две очереди: первая должна содержать все нечетные, а вторая —

все четные числа из исходного набора (порядок чисел в каждой очереди должен совпадать с порядком чисел в исходном наборе). Вывести указатели на начало и конец первой, а затем второй очереди (одна из очередей может оказаться пустой; в этом случае вывести для нее две константы nil);

б) дно число *D* и указатели P_1 и P_2 на начало и конец очереди (если очередь является пустой, то $P_1 = P_2$ = nil). Добавить элемент со значением *D* в конец очереди и вывести новые адреса начала и конца очереди;

в) дано число *D* и указатели *P*l и *P*2 на начало и конец очереди, содержащей не менее двух элементов. Добавить элемент со значением *D* в конец очереди и извлечь из очереди первый (начальный) элемент. Вывести значение извлеченного элемента и новые адреса начала и конца очереди. После извлечения элемента из очереди освободить память, занимаемую этим элементом;

г) дано число *N* (> 0) и указатели *P*l и *P*2 на начало и конец непустой очереди. Извлечь из очереди *N* начальных элементов и вывести их значения (если очередь содержит менее *N* элементов, то извлечь все ее элементы). Вывести также новые адреса начала и конца очереди (для пустой очереди дважды вывести nil). После извлечения элементов из очереди освобождать память, которую они занимали;

д) даны указатели *P*l и *P*2 на начало и конец непустой очереди. Извлекать из очереди элементы, пока значение начального элемента очереди не станет четным, и выводить значения извлеченных элементов (если очередь не содержит элементов с четными значениями, то извлечь все ее элементы). Вывести также новые адреса начала и конца очереди (для пустой очереди дважды вывести nil). После извлечения элементов из очереди освобождать память, которую они занимали;

е) даны две очереди ; адреса начала и конца первой равны *P*l и *P*2, а второй — *P*3 и *P*4 (если очередь является пустой, то соответствующие адреса равны nil). Переместить все элементы первой очереди (в порядке от начала к концу) в конец второй очереди и вывести новые адреса начала и конца второй очереди. Операции выделения и освобождения памяти не использовать.

Критерии выполнения курсовой работы

Результаты курсового проектирования оцениваются с учетом:

1) качество разработанных и документированных алгоритмов;
2) наличия работающей программы;
3) качества и полноты выполнения пояснительной записки;
4) уровня ответов обучающегося при защите работы.

Составил
к.т.н., доцент кафедры
«Информационная безопасность» _____ Ю.М. Кузьмин