Федеральное государственное бюджетное образовательное учреждение
высшего образования
Московский государственный университет имени М.В. Ломоносова
Факультет вычислительной математики и кибернетики

УТВЕРЖДАЮ
декан факультета вычислительной
математики и кибернетики

_____/И.А. Соколов /
«27» сентября 2022г.

ФОНД ОЦЕНОЧНЫХ СРЕДСТВ

по дисциплине

Системы программирования

Уровень высшего образования:

бакалавриат

Направление подготовки / специальность:

01.03.02 "Прикладная математика и информатика" (3++)

Направленность (профиль) ОПОП:

Искусственный интеллект и анализ данных

Форма обучения:

очная

Рассмотрен и утвержден
на заседании Ученого совета факультета ВМК
(протокол №7, от 27 сентября 2022 года)

Москва 2022

1. ФОРМЫ И ОЦЕНОЧНЫЕ МАТЕРИАЛЫ ТЕКУЩЕГО КОНТРОЛЯ УСПЕВАЕМОСТИ И ПРОМЕЖУТОЧНОЙ АТТЕСТАЦИИ

В процессе и по завершении изучения дисциплины оценивается формирование у студентов следующих компетенций:

Планируемые результаты обучения по дисциплине (модулю)		
Содержание и код компетенции.	**Индикатор (показатель) достижения компетенции**	**Планируемые результаты обучения по дисциплине, сопряженные с индикаторами достижения компетенций**
ОПК-2. Способен использовать и адаптировать существующие математические методы и системы программирования для разработки и реализации алгоритмов решения прикладных задач	ОПК-2.1. Знание приемов написания и анализа алгоритмов и компьютерных программ; ОПК-2.2. Способность анализировать и конструировать конкретные алгоритмы на языке высокого уровня для решения разнообразных математических задач на компьютере. ОПК-2.3. Знание парадигм структурного, процедурно-модульного и объектно-ориентированного программирования на языке высокого уровня.	Знать: 1. возможности, назначение, состав и схему функционирования современных систем программирования; 2. основные принципы объектно-ориентированной парадигмы программирования, как наиболее распространенной и востребованной в настоящее время; 3. основные возможности и методы программирования на объектно-ориентированном языке программирования C++; 4. основы теории формальных языков и грамматик, на которых базируются современные трансляторы языков программирования; 5. основы теории трансляции; 6. основные подходы к построению трансляторов языков программирования. Уметь: 1. применять на практике метод декомпозиции задачи; 2. оперировать понятиями класса, объекта, абстрактного типа данных; 3. применять на практике принципы объектно-ориентированного программирования: - инкапсуляция, - наследование, - полиморфизм; 4. применять на практике

		механизм исключений, аппарат RTTI; 5. строить порождающие грамматики Хомского для описания формальных языков, определять тип грамматики и языка по Хомскому; 6. описывать регулярные языки с помощью конечного автомата, применять алгоритм преобразования недетерминированного конечного автомата к детерминированному, применять алгоритм построения лексических анализаторов; 7. применять метод рекурсивного спуска для синтаксического анализа по контекстно-свободной грамматике, определять применимость этого метода. Владеть: 1. навыками работы с современными системами программирования; 2. навыками программирования в актуальном на сегодняшний день объектно-ориентированном стиле; 3. навыками в построении трансляторов языков программирования;

1.1. Текущий контроль успеваемости

Текущий контроль успеваемости осуществляется путем оценки результатов выполнения заданий практических (семинарских) занятий, самостоятельной работы, предусмотренных учебным планом и посещения занятий/активность на занятиях.

В качестве оценочных средств текущего контроля успеваемости предусмотрены:

коллоквиум
контрольная работа

Вопросы к коллоквиуму.

1. Абстрактные типы данных, инкапсуляция, наследование, полиморфизм.
2. Класс, объект, состояние объекта, поведение объекта.
3. C++: Пространства имен. Пространство имен *std*.
4. C++: Конструкторы и деструкторы.

5. C++: Присваивание и инициализация.
6. C++: Ссылки в C++. Передача параметров по ссылке.
7. C++: Манипуляции с состоянием объекта.
8. C++: Работа с динамической памятью.
9. C++: Друзья класса.
10. C++: Статические члены класса.
11. Виды полиморфизма в C++ (статический, динамический, параметрический).
12. C++: Статический полиморфизм. Перегрузка бинарных операций:
    a. с помощью функции-члена класса
    b. с помощью функции-друга класса
13. C++: Статический полиморфизм. Перегрузка унарных операций:
    a. с помощью функции-члена класса
    b. с помощью функции-друга класса
14. C++: Специфика перегрузки операций инкремента и декремента, операции индексации.
15. C++: Статический полиморфизм. Перегрузка функций.
16. C++: Алгоритм поиска оптимально отождествляемой (best-matching) функции.
17. C++: Средства обработки ошибок. Исключения и обработка исключений.
18. Виды отношений между классами (ассоциация, наследование, агрегация, использование).
19. C++: Одиночное наследование. Правила наследования. Видимость при наследовании.
20. C++: Динамический полиморфизм. Виртуальные функции.
21. Принципы реализации виртуальных функций
22. C++: Абстрактные классы.
23. C++: Множественное наследование. Видимость при множественном наследовании. Виртуальные базовые классы.
24. C++: Динамическая информация о типе (RTTI).
25. C++: Параметрический полиморфизм. Шаблонные функции.
26. C++: Шаблонные классы.
27. Стандартная библиотека C++.
28. Стандартная библиотека шаблонов STL.
29. STL: контейнеры, итераторы, алгоритмы, аллокаторы.
30. STL: Шаблонные классы *vector* и *list*.

Контрольная работа

Вариант 1

1. Вычеркните только те строки функции *main ()*, которые приводят к синтаксической ошибке (если таковые имеются). Обязательно **объясните** причины ошибок. Что будет напечатано в результате работы получившейся программы.

```
struct A {
      int n;
      A(){ n = 9;}
      A operator+(const A & a){
            A t;
            t.n = n + a.n;
            return t;    }
};
struct B:A {
      B operator+(const B & a){
            B t;
            t.n = n + a.n + 10;
            return t;  }
};
```

```
int main () {
      A a;
      B b;
      a = a + b;
      cout << a.n << '\n';
      b = b + a;
      cout << b.n << '\n';
      return 0;
}
```

2. Исправьте **только описание класса A** так, чтобы в приведенной ниже программе не было ошибок,

а на экран напечаталось **f (int, int)**.

```
struct A {  int n;
            A (int  m) { n = m; }
            operator int () { return 1; }
};
void  f ( int i,  int  j )  { cout << "f(int,  int)\n";}
void  f ( A b,  A a )  { cout << "f(A,  A)\n";}

int main () {          A a (1);
                       f (a, 1);
                       return 0;   }
```

3. Опишите необходимые классы так, чтобы в заданной функции *main ()* не было ошибок, а на экран печаталось **310.**

```
int main () {
        T t (3);
        P <T> p1(&t), p2(0);
        cout  <<  p1 -> n  <<  p2 -> n  <<  endl;
        return 0 ; }
```

4. Добавьте в описание класса A **из задачи 1** метод

virtual A& **operator** - (){ n = - (n+1)**;** cout << n <<" A::**operator** - ()\n"**; return *this;**},

а в описание (производного от класса A) класса B из задачи 1 -

B& **operator** - () { cout << n << " B::**operator** - () \n"**; return *this;**}

Что напечатает программа с нижеприведенной функцией *main()* и получившимися классами A и B?

```
int main () {
        B b;
        A  a,  & ra = b;
        a = -a;
        ra = -ra;
        ra = a;
        ra = -ra;     return 0;
}
```

5. Что такое абстрактный класс в C++?
Обязательно приведите **пример** описания и использования абстрактного класса.

6. Укажите все прототипы конструкторов и деструкторов в порядке их выполнения в следующей программе:

```
class A {};
class B: public A{};
struct  D { A a; };

int main () {
       D d;
       { d.a = B();
         B b; }
       A a1, a2 = a1;
       return 0;
}
```

7. Добавьте в следующую программу необходимые описания так, чтобы в ней не было ошибок.

```
int main () {
        const A a;
        a.x = 3;
        cout << a.y << a.x << a.f(1) << endl;
        return 0; }
```

8. Модифицируйте функцию *main()*, **ничего в ней не удаляя, не используя комментарии, goto и прочие переходы** так, чтобы программа завершалась нормально (не аварийно).

```
struct B { virtual void f () { cout << "B::f()\n"; } };
struct D : B { void f () { cout << "D::f()\n"; } };

int main () {
        D d, &rd  = d;
        B b, & rb = b, &rbd = d;
        rd = dynamic_cast <D&> (rbd);  rd.f();
        rd = dynamic_cast <D&> (rb);   rd.f();
        return 0;
}
```

9. C++11. Вычеркните неверные конструкции в следующей программе на C++11.
Обязательно **объясните** причины ошибок в вычеркнутых конструкциях.

```
struct A {
        int  i = 10;
};

void f (A && x) { }

int main () {
        A a;
        int && n, m;
        auto b = A();
        decltype (a) c;
        f (b);
        f ( A() );
        m = nullptr;
        return 0;
}
```

10. **STL.** Напишите шаблонную функцию, подсчитывающую сумму пяти последних элементов заданного контейнера (списка или вектора, константного или неконстантного). Тип элементов контейнера является числовым типом. Если в заданном контейнере меньше 5 элементов, функция должна вернуть 0 соответствующего типа.

Вариант 2

1. Вычеркните только те строки функции *main ()*, которые приводят к синтаксической ошибке (если таковые имеются). Обязательно **объясните** причины ошибок. Что будет напечатано в результате работы получившейся программы.

```
struct T {
        int i;
        T(){i = 3;}
        T(int t){ i = t;}
        T operator*(const T & a){
                T t;
                t.i = i * a.i;
                return t;     }
};
struct P:T {
        P operator*(const P & a){
                P t;
                t.i = i * a.i * 2;
                return t;   }
};
```

```
int main () {

        T a(5);
        P b;
        a = a * b;
        cout << a.i << '\n';
        b = b * a;
        cout << b.i << '\n';
        return 0;
}
```

2. Исправьте **только описание класса A** так, чтобы в приведенной ниже программе не было

ошибок, а на экран напечаталось **f (A, A)**.

```
struct A {  int n;
            A (int  m) { n = m; }
            operator int () { return 1; }
};
void  f ( int i,  int  j )  { cout << "f(int,  int)\n";}
void  f ( A b,  A a )  { cout << "f(A,  A)\n";}

int main () {          A a (1);
                       f (a, 1);
                       return 0;   }
```

3. Опишите необходимые классы так, чтобы в заданной функции *main ()* не было ошибок, а на экран печаталось **520.**

```
int main () {
        C c (5);
        B <C> b1(&c), b2(0);
        cout  <<  (*b1). n  <<  (*b2). n  <<  endl;
        return 0 ; }
```

4. Добавьте в описание класса T **из задачи 1** метод

virtual T& **operator --** (){ i = i - 1; cout << i <<" T::**operator** -- () \n"; **return *this;** } ,

а в описание (производного от класса T) класса P из задачи 1 -

P& **operator --** () { cout << i << " P::**operator** -- () \n"; **return *this;** }

Что напечатает программа с нижеприведенной функцией *main()* и получившимися классами T и P?

```
int main () {
        P b;
        T  a(5),  & rt = b;
        --a;
        --rt;
        rt = a;
        --rt;    return 0;
}
```

5. Что такое функция-друг в языке Си++? Приведите **пример** описания и использования функции-друга.

6. Укажите все прототипы конструкторов и деструкторов в порядке их выполнения в следующей программе:

```
class A {};
class B: public A{};
struct T { B * pb; };

int main () {
      A a;
      { a = B();
        T t; }
      B b1, b2 = b1;
      return 0; }
```

7. Добавьте в следующую программу необходимые описания так, чтобы в ней не было ошибок.

```
int main () {
        B::y = 3 ;
        const B b;
        cout << b.x-B::h(2)<< b.h(0)<< endl;
        return 0; }
```

8. Модифицируйте функцию *main()*, **ничего в ней не удаляя, не используя комментарии, goto и прочие переходы** так, чтобы программа завершалась нормально (не аварийно).

```
struct B { virtual void  g() { cout << "B::g () \n"; } };
struct D : B { };

int main () {
        D d, * pd1, *pd2;
        B b, * pb = &b, * pbd = &d;
        pd1 = dynamic_cast < D* > (pbd);
        pd2 = dynamic_cast < D* > (pb);
        if ( typeid (*pd1) == typeid (*pd2) )    pb -> g () ;
        return 0; }
```

9.C++11. Вычеркните неверные конструкции в следующей программе на C++11.
Обязательно **объясните** причины ошибок в вычеркнутых конструкциях.

```
struct S {
        int  x = 1;
};

void  g (S && s) { cout << s.x << endl;}

int main () {
        int && i;
        auto k = nullptr;
        decltype ( S ( ) ) a;
        char * s = k;
        k = s;
        g (a);
        g ( S() );
        return 0;
}
```

10. **STL.** Напишите шаблонную функцию, подсчитывающую сумму первых семи элементов заданного контейнера (списка или вектора, константного или неконстантного). Тип элементов контейнера является числовым типом. Если в заданном контейнере меньше 7 элементов, функция должна вернуть 0 соответствующего типа

1.2. Промежуточная аттестация

Промежуточная аттестация осуществляется в форме экзамена
В качестве средств, используемых на промежуточной аттестации предусматривается:
Билеты

1.3. Типовые задания для проведения промежуточной аттестации

Экзаменационная работа (письменный экзамен)

Вариант 1

1. Дана грамматика *G*:

$S \to aS\ |Sb|aAb|\ \varepsilon$
$A \to aaAbb|aabb$
$A \to A$

(а) Описать язык *L*(*G*) в виде теоретико-множественной формулы:

Ответ:
L(*G*) =

(б) Каким из перечисленных классов грамматик принадлежит *G*? **Ответ:**

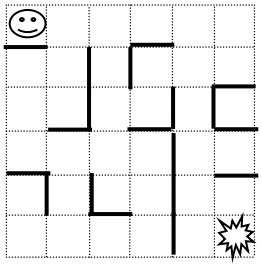
Класс П	*G* ∈ П? (да/нет)
регулярные	
контекстно-зависимые	
контекстно-свободные	
грамматики типа 0	
неукорачивающие	

(в) Тип языка: найти такое целое *k*, что *L(G)* является языком типа *k*, но не языком типа *k*+1.
Ответ: *k* =

2. Является ли однозначной данная грамматика *G*, порождающая язык цепочек в алфавите {0,1}, в которых символов 0 и 1 поровну? Ответ обосновать.

G: $S \to 0S1S\ |1S0S\ |\ 01BB|\ \varepsilon$
$B \to BB|B01$

3. Тесей должен пройти из левого верхнего угла лабиринта в правый нижний, чтобы убить Минотавра. Нить Ариадны оставляет след перемещений Тесея. Он умеет за один шаг перемещаться на клетку вниз (след обозначается символом *a*), вверх (след *b*) или вправо (след *c*). Влево не умеет. Каждая клетка посещается не более одного раза. Постройте грамматику, порождающую язык в алфавите {*a, b, c*} всевозможных путей Тесея, приводящих к цели. Например, цепочка *ccccaaaaac* приведет Тесея к Минотавру. В грамматике должно быть не более 4-х правил вывода, считая альтернативы.



4. Дан список слов: *в, и, исправление, обнаружение, ошибки, программа*. Составьте из него, употребив слова в нужном порядке и форме, определение термина (понятия) из раздела СП и назовите сам термин (он не входит в заданный список).

5. Можно ли считать утилиту *make* системы *Unix* компилятором? Обоснуйте ответ.

6. По заданной грамматике *G* = {{*a, b*}, {*B, S*}, *P, S*} получить эквивалентную неукорачивающую контекстно-свободную грамматику (использовать алгоритм устранения правил с пустой правой частью).

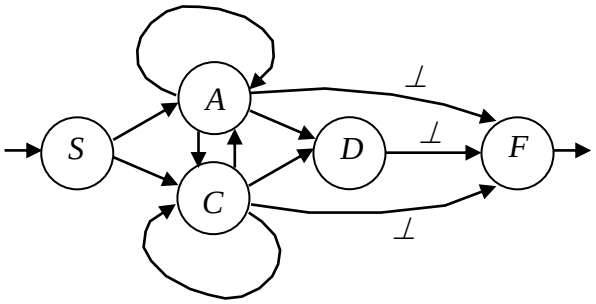
P: $S \to aBb\ |\ \varepsilon$ **Ответ:**
$B \to aB\ |\ bS\ |\ SS$

7. Даны 1) функция-анализатор на языке Си++ для леволинейной грамматики G_L:

```
bool scan_G ()
{   enum state { H, B, S, ER };  // множество состояний
```

```
        state CS= H;                    // CS -- текущее состояние
        do { gc ();   // считывает символ в глобальный объект c
             switch (CS) {
                 case H: if ( c == 'a' ) CS = B;
                         else CS = ER; break;
                 case B: if ( c == 'b' );
                     else if ( c == '⊥' ) CS = S;
                          else CS = ER; break;
                                }
             }
        while ( CS != S && CS != ER);
        return CS == S; // true, если CS != ER,  иначе false
    }
```

2) заготовка диаграммы состояний праволинейной грамматики G_R:



(а) Расставить в заготовке диаграммы для G_R терминальные символы так, чтобы грамматики G_R и G_L были эквивалентны.

(б) Восстановить грамматику G_R. **Ответ:**

(в) Сколько состояний будет иметь конечный автомат, полученный алгоритмом детерминизации диаграммы для G_R ? **Ответ:____**

8. Дана КС-грамматика G, порождающая язык L.

Вставить в грамматику действия вида ⟨*cout* << "символы" ;⟩ так, чтобы в процессе рекурсивного спуска был реализован перевод $\tau = \{(\omega, a^k b^{k+n}) \mid \omega \in L,\ k = |\omega|_a,\ n = |\omega|_b\}$.

G:

$S \to aA \mid bB \mid \varepsilon$

$A \to cA \mid S$

$B \to cB \mid S$

9. Дана грамматика G. Докажите, что метод рекурсивного спуска неприменим к ней. Можно ли вычеркнуть один терминальный символ в правилах грамматики так, чтобы к получившейся грамматике метод был бы применим?

G: $S \to aSb \mid Yb \mid bb$

$Y \to cSY \mid bd \mid \varepsilon$

10. Постройте ПОЛИЗ для фрагмента программы на Си. Префиксный ++ в польской записи обозначается как +#, постфиксный как #+.

```
for ( i = 0, j = 10;  i == j ? 0 : 1; --j) a += 2 < i++ > j;
```

ПОЛИЗ

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17

18	19	20	21	22	23	24	25	26	27	28	29	30	31	32	33	34	35	36	37	38	39

Вариант 2

1. Дана грамматика G:

$S \to cA \mid cAb \mid cb \mid \varepsilon$

$cAb \to cAbb \mid cb \mid ccAb$

$ccAbb \to ccAbb$

(а) Описать язык $L(G)$ в виде теоретико-множественной формулы:

Ответ:

(б) Каким из перечисленных классов грамматик принадлежит G? **Ответ:**

Класс П	$G \in$ П? (да/нет)
регулярные	

$L(G)$ =

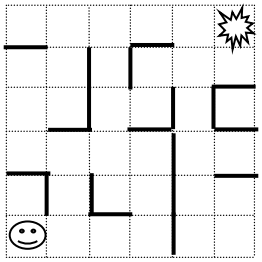
контекстно-зависимые	
контекстно-свободные	
грамматики типа 0	
неукорачивающие	

(в) Тип языка: найти такое целое k, что $L(G)$ является языком типа k, но не языком типа $k+1$.
Ответ: k =

2. Является ли однозначной данная грамматика G , порождающая язык цепочек в алфавите $\{a,b\}$? Ответ обосновать.

G: $S \to aSaS \mid aBa \mid aSa \mid A$
$B \to BB \mid Ba$
$A \to b \mid \varepsilon$

3. Баба-Яга дала Иванушке клубочек, который через лесной лабиринт проведет его к сундуку, где находится смерть Кащеева. Клубочек (а за ним и Иванушка) умеет за один шаг перемещаться на клетку вниз (оставляемый след обозначается символом a), вверх (след b) или вправо (след c). Влево не умеет. Каждая клетка посещается не более одного раза. Постройте грамматику, порождающую язык в алфавите $\{a, b, c\}$ всевозможных путей Иванушки, приводящих к цели. Иванушка начинает путь из левого нижнего угла. Цель – в правом верхнем. Например, цепочка *cbbcbbbccc* приведет Иванушку к цели. В грамматике должно быть не более 4-х правил вывода, считая альтернативы.



4. Дан список слов: *входные*, *заранее*, *данные*, *для*, *известный, программа, работа*, *результат,с*. Составьте из него, употребив слова в нужном порядке и форме, определение термина (понятия) из раздела СП и назовите сам термин (он не входит в заданный список).

5. Можно ли считать утилиту *make* системы *Unix* интерпретатором? Обоснуйте ответ.

6. По заданной грамматике $G = \{\{a, c\}, \{B, S\}, P, S\}$ получить эквивалентную неукорачивающую контекстно-свободную грамматику (использовать алгоритм устранения правил с пустой правой частью).

Ответ:

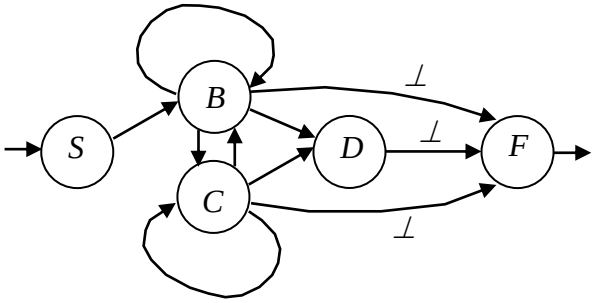
P: $S \to aBc \mid \varepsilon$
$B \to aB \mid cS \mid SaS$

7. Даны 1) функция-анализатор на языке Си++ для леволинейной грамматики G_L:

```
bool scan_G ()
{   enum state { H, A, S, ER };  // множество состояний
    state CS= H;                 // CS –– текущее состояние
    do { gc (); // считывает символ в глобальный объект c
         switch (CS) {
             case H: if ( c == 'a'||c =='b' ) CS = A;
                     else CS = ER; break;
             case A: if ( c == 'a' );
                   else if ( c == '⊥' ) CS = S;
                      else CS = ER; break;
                          }
         }
    while ( CS != S && CS != ER);
```

```
        return CS == S; // true, если CS != ER,  иначе false
    }
```

2) заготовка диаграммы состояний праволинейной грамматики G_R:



(а) Расставить в заготовке диаграммы для G_R терминальные символы так, чтобы грамматики G_R и G_L были эквивалентны.

(б) Восстановить грамматику G_R. **Ответ:**

(в) Сколько состояний будет иметь конечный автомат, полученный алгоритмом детерминизации диаграммы для G_R ? **Ответ:____**

8. Дана КС-грамматика G, порождающая язык L.

Вставить в грамматику действия вида ⟨*cout* << "символы" ;⟩ так, чтобы в процессе рекурсивного спуска был реализован перевод $\tau = \{(\omega, a^k d^{k-n}) \mid \omega \in L, k = |\omega|, n = |\omega|_d\}$.

G:

$S \to aA \mid dD \mid \varepsilon$

$A \to cA \mid S$

$D \to cD \mid S$

9. Дана грамматика G. Докажите, что метод рекурсивного спуска неприменим к ней. Можно ли вычеркнуть один терминальный символ в правилах грамматики так, чтобы к получившейся грамматике метод был бы применим?

G: $S \to aSb \mid Xa \mid bb$

$X \to cSX \mid bd \mid \varepsilon$

10. Постройте ПОЛИЗ для фрагмента программы на Си. Префиксный ++ в польской записи обозначается как +#, постфиксный как #+.

```
i = 0, j = 10; do a += 2 < i++ > --j; while ( i == j ? 0 : 1 );
```

ПОЛИЗ

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17

18	19	20	21	22	23	24	25	26	27	28	29	30	31	32	33	34	35	36	37	38	39

Вопросы к экзамену.

1. Этапы жизненного цикла программного продукта.
2. Состав и схема функционирования классической системы программирования.
3. Типы трансляторов, особенности интерпретаторов и компиляторов. Смешанная стратегия трансляции.
4. Общая схема работы компилятора.
5. Основные понятия теории формальных грамматик и языков.
6. Эквивалентные грамматики.
7. Классификация формальных грамматик и языков по Хомскому.
8. Соотношения между типами грамматик.
9. Соотношения между типами языков.
10. Разбор цепочек по КС-грамматикам. Задача разбора. Дерево вывода.
11. Неоднозначность грамматик и языков.
12. Недостижимые и бесплодные символы грамматики. Алгоритмы удаления недостижимых и бесплодных символов. Приведенная грамматика. Алгоритм приведения грамматики.
13. Алгоритм удаления правил с пустой правой частью.
14. Определение недетерминированного конечного автомата (НКА).

15. Диаграмма состояний (ДС) конечного автомата.
16. Леволинейные регулярные грамматики и конечные автоматы.
17. Определение детерминированного конечного автомата (ДКА).
18. Алгоритм построения детерминированного конечного автомата по НКА.
19. Задачи лексического анализа.
20. Лексический анализ на основе регулярных грамматик.
21. Задачи синтаксического анализа.
22. Метод рекурсивного спуска (РС-метод): назначение, семантика процедур метода рекурсивного спуска.
23. Достаточные условия применимости метода рекурсивного спуска для грамматик без ε-альтернатив и для грамматик с ε-альтернативами.
24. Критерий применимости РС-метода.
25. Задачи семантического анализа. Грамматики с действиями.
26. Свойства языка внутреннего представления программы, примеры таких языков.
27. Синтаксически управляемый перевод: идея, принципы организации, примеры. Определение формального перевода.
28. ПОЛИЗ выражений.
29. ПОЛИЗ операторов языков программирования.
30. Генерация ПОЛИЗа выражений и операторов.
31. Интерпретация ПОЛИЗа.
32. Основные стратегии распределения памяти.
33. Машинно-независимая и машинно-зависимая оптимизация. Примеры оптимизирующих преобразований.
34. Интегрированная среда разработки программного обеспечения (ИСР).
35. Основные функции редактора текстов в рамках ИСР. Примеры его интегрированности с другими компонентами ИСР.
36. Отладчики, их возможности. Примеры интегрированности отладчика с другими компонентами ИСР.
37. Редактор внешних связей, его назначение и принципы работы. Загрузчик.
38. Профилировщики. Назначение. Принцип работы.
39. Библиотеки. Основные типы библиотек.
40. Критерии проектирования стандартных библиотек.
41. Способы подключения библиотек функций.

2. КРИТЕРИИ ОЦЕНКИ ПО ДИСЦИПЛИНЕ

ШКАЛА И КРИТЕРИИ ОЦЕНИВАНИЯ результатов обучения (РО) по дисциплине				
Оценка виды оценочных средств	**2 (не зачтено)**	**3 (зачтено)**	**4 (зачтено)**	**5 (зачтено)**
Знания **(виды оценочных средств: приведены в п. 1.2.)**	Отсутствие знаний	Фрагментарные знания	Общие, но не структурированные знания	Сформированные систематические знания
Умения **(виды оценочных средств: приведены в п. 1.2.)**	Отсутствие умений	В целом успешное, но не систематическое умение	В целом успешное, но содержащее отдельные пробелы умение (допускает неточности непринципиального характера)	Успешное и систематическое умение
Навыки **(владения, опыт деятельности)** **(виды оценочных средств: приведены в п. 1.2..)**	Отсутствие навыков (владений, опыта)	Наличие отдельных навыков (наличие фрагментарного опыта)	В целом, сформированные навыки (владения), но используемые не в активной форме	Сформированные навыки (владения), применяемые при решении задач