КИПР

ТУСУР

Кафедра конструирования и производства радиоаппаратуры

УТВЕРЖДАЮ
Заведующий кафедрой КИПР
_____В.Н. ТАТАРИНОВ
“___” _____2012 г.

Разветвленные программы

Лабораторная работа по дисциплинам ***«***Информатика***»*** для студентов специальностей 211000.62 «Конструирование и технология электронных средств» (бакалавриат) и 162107.65 «Информатика и информационные технологии» (специалитет)

Разработчик:
Доцент кафедры КИПР
_____**Ю.П. Кобрин**

Томск 2012

СОДЕРЖАНИЕ

1 Цели работы

✓ Знакомство с операторами управления в программах на языке ***Borland Pascal***.

✓ Освоение приемов программирования и отладки разветвленных алгоритмов

✓ Закрепление навыков практической работы в интегрированной среде ***Borland Pascal***.

2 Порядок выполнения работы

1) Ознакомиться с разделами «Операторы управления программой», «Составление и отладка разветвленных программ» настоящей работы, а также, при необходимости, с дополнительной литературой [1] [2] [3] [4] [5] [6].
2) Ответить письменно на контрольные вопросы.
3) Получить у преподавателя индивидуальное задание. Разработать программу в соответствии с индивидуальным заданием.
4) Войти в свой личный каталог, загрузить и настроить систему ***Borland Pascal 7.0***.
5) Ввести шаблон программы, сделанный на прошлом занятии. Сделать его копию, записав под новым именем (разветвленной программы) с помощью команды ***Save as*** ...
6) Ввести разработанную Вами программу, корректируя и дополняя свой шаблон.
7) Освоить методику поиска причин и исправления синтаксических ошибок, а также отладки программы «по шагам» по тестовому примеру. Добиться, чтобы программа дала правильные результаты
8) Оформить отчет и защитить его у преподавателя.

3 Контрольные вопросы

Ответьте на следующие контрольные вопросы:

1) Что такое *составной оператор* и какие функции он выполняет?
2) Как выполняются *операторы безусловного перехода*?
3) Почему не рекомендуется использование в программах *операторов* **GoTo**?
4) Возможна ли передача управления внутрь подпрограммы с помощью *оператора* **GoTo** и почему?
5) Как выполняется работа *условного оператора* **if**?
6) Для каких целей используется *оператор* **case**?
7) В чем сущность процесса отладки?

4 Защита отчета

Отчет должен быть выполнен в соответствии с [11] и состоять из следующих разделов:

✓ Тема и цель работы.
✓ Индивидуальное задание.
✓ Ответы на контрольные вопросы.
✓ Текст программы (в электронном виде), тестовые исходные данные и предполагаемые результаты тестирования.

✓ Результаты выполнения программы и выводы.

При защите отчета по работе для получения зачета студент должен:

✓ уметь отвечать на контрольные вопросы;
✓ обосновать структуру выбранного алгоритма и показать его работоспособность;
✓ уметь пояснять работу программы;
✓ продемонстрировать навыки работы в среде ***Borland Pascal***.

5 Операторы управления программой в языке Borland Pascal

5.1 Постановка задачи

Для изменения *естественного порядка выполнения операторов* (так как выполняются операторы в линейной программе - один за другим) в языке ***Borland Pascal*** предусмотрены *структурные (управляющие) операторы*:

✓ *составной оператор* ***begin .. end;***
✓ *оператор безусловной передачи управления* ***GoTo;***
✓ *оператор альтернативы (условный)* ***if .. then .. else .. ;***
✓ *оператор выбора (варианта)* ***case .. of.***

5.2 Составные операторы

В большинстве структурных операторов правила построения конструкций языка ***Borland Pascal*** допускают использование *только одного оператора*. В тоже время, нередко алгоритм в этом месте программы предусматривает выполнение целой группы операторов. Чтобы обойти это ограничение применяют ***составной оператор*** *(рис. 5.1)*.

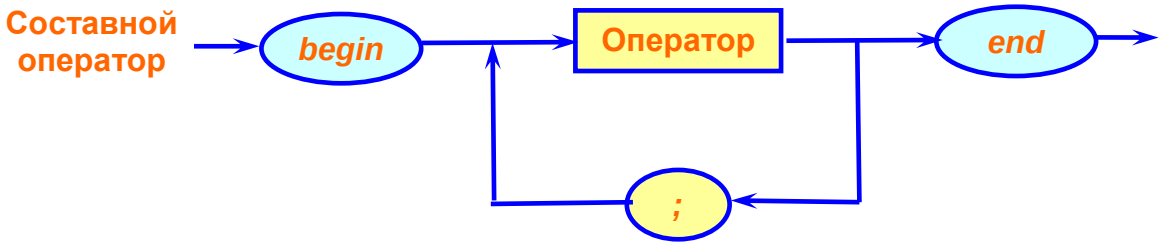


Рисунок 5.1 - Синтаксическая диаграмма составного оператора

Составной оператор объединяет группу операторов, после чего рассматриваемая группа операторов может считаться *одним оператором*:

```
begin  {начало составного оператора – открывающая операторная скобка}
   оператор 1; {группа операторов 1, 2, ..., N рассматриваемая как единое целое}
   оператор 2;
   …
   оператор N
end;  {конец составного оператора – закрывающая операторная скобка}
```

Таким образом, весь раздел операторов, обрамленный словами ***begin .. end***, представляет собой один составной оператор[1].

---

[1] С этой точки зрения любая программа или подпрограмма в ***Borland Pascal*** состоят из единственного оператора – составного.

Запомните! Поскольку зарезервированное слово ***end*** является закрывающей операторной скобкой, оно одновременно указывает и конец предыдущего оператора. Поэтому ставить перед ***end*** символ «***;***» **необязательно** (оператор ***end*** является разделителем такого же уровня, как и «***;***»).

Составные операторы - важный инструмент ***Borland Pascal***, дающий возможность писать программы по современной технологии структурного программирования (без операторов перехода ***GoTo***).

5.3 Пустые операторы

Присутствие точки с запятой перед ***end*** в предыдущих примерах означало, что между последним оператором и операторной скобкой ***end*** располагается *пустой оператор*. Пустой оператор не содержит никаких действий, просто в программу добавляется лишняя точка с запятой. В основном пустой оператор используется для передачи управления в конец составного оператора.

5.4 Условная передача управления

Нередко в зависимости от ситуации, возникшей в ходе решения задачи, нужно выбрать один из двух или более вариантов решения. Выбор той или иной ветви вычислительного процесса в алгоритмах ***с разветвляющейся структурой*** осуществляется в зависимости от выполнения поставленного условия.

В ***Borland Pascal*** включены два условных оператора ***If*** и ***case***. На практике *условным оператором* чаще всего называют оператор ***If***, а оператор ***case*** именуют *оператором выбора* или *оператором варианта.*

Синтаксическая диаграмма условного оператора ***If*** приведена на *рис. 5.2.*

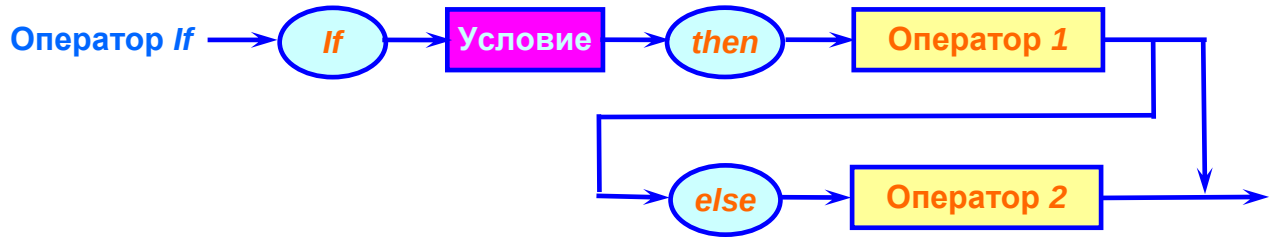


Рисунок 5.2 - Синтаксическая диаграмма оператора ***If***

Условный оператор if позволяет во время выполнения программы проверить некоторое *логическое условие*[2] и в зависимости от результатов проверки выбрать одно из двух направлений дальнейших вычислений (последовательности инструкций, которые должны быть выполнены):

```
if B    {если логическое условие B  - True (истина),}
  then Оператор1  {то выполняется оператор 1}
 else Оператор2;  {иначе, если логическое условие B – False
                   (ложь), выполняется оператор 2}
```

При помощи логических операций ***And*** (логическое "***И***") и ***Or*** (логическое "***Или***") из простых условий можно строить сложные.

---

[2] Результат проверки может быть только значение булевского типа (***True*** или ***False***)!

***Пример 5.1*.**

Выбрать наименьшее из чисел ***a*** и ***b*** и присвоить его переменной ***Min***.

Решение.

Блок-схема алгоритма решения этой задачи приведена на *рис. 5.3*.

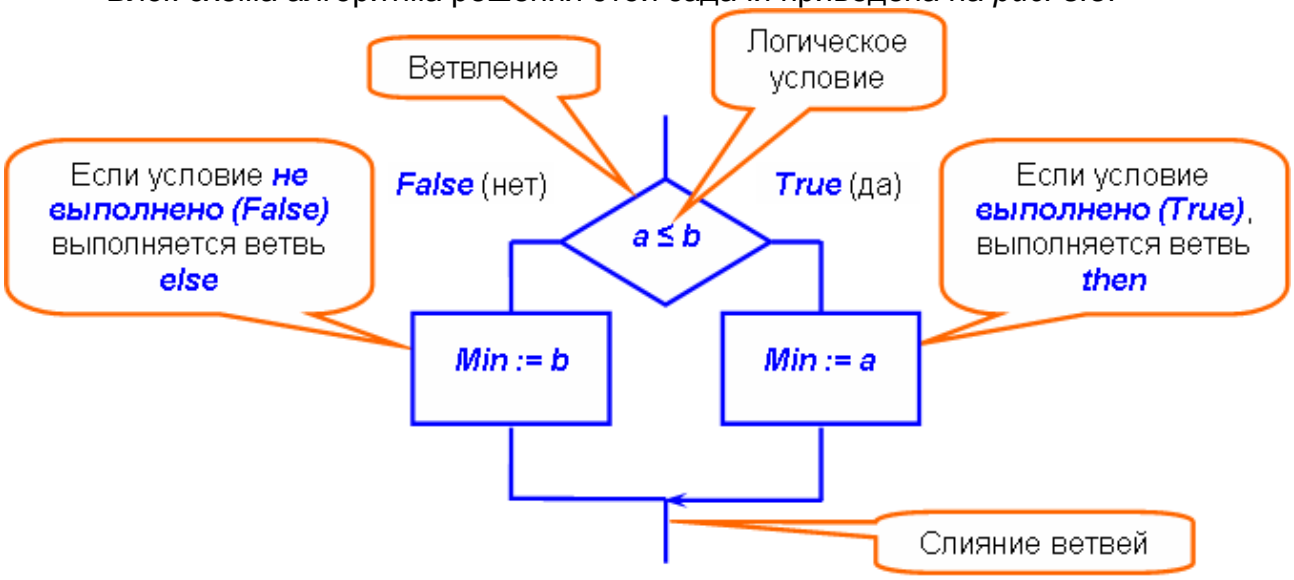


Рисунок 5.3 – Блок-схема решения задачи выбора наименьшего из двух чисел ***a*** и ***b***

Фрагмент программы определения наименьшего из двух чисел на ***Borland Pascal*** может выглядеть так:

```
...
If  a <=  b {если выполняется условие  a ≤ b, }
   then Min := a { то переменной Min присваивается значение переменной a}
   else Min := b; {иначе переменной Min присваивается значение b}
...
```

✓ Обратите внимание на некоторые особенности оператора ***If.***

Запомните! Перед ключевым словом ***else*** символ «***;***» ***никогда не ставится***, так как он *заканчивает любой оператор*, а оператора, который начинался бы с ***else*** в ***Borland Pascal*** нет!

Пример 5.2 (с ошибкой)

```
...
If  a <=  b
   then Min := a ;   ⇐ ошибка!
   else Min := b;
...
```

✓ Как видно из синтаксической диаграммы оператора **If** (см. *рис. 5.2*), допускается ***неполная форма условного оператора If*** (без ветви **else**):

If B {если проверяемое логическое условие **B** - ***True*** (истина),}
then Оператор1; {то выполняется ***Оператор 1***, иначе – никаких действий}

Отметим, что в качестве оператора в любой ветви условного оператора может применяться любой исполняемый оператор ***Borland Pascal***, в том числе и сам оператор **if**. В этом случае (особенно, когда применяются неполные формы **if**), может возникнуть неоднозначность – какому из **if** соответствует ветвь **else.**

Пример 5.3.

Как работает следующий фрагмент программы?

```
…
If выражение then
If выражение then
оператор
else оператор
…
```

Разобраться сложно, если не знать простое правило:

Правило: ключевое слово **else** связывается с ближайшим стоящим перед ним ключевым словом **if**, которое еще не было связано с каким-либо ключевым словом **else**.

Если проанализировать с помощью этого правила предложенный в *примере 5.3* фрагмент программы и на этой основе более четко обозначить структуру вложенности, записав **else** на одном уровне с **then**, которому оно соответствует, то получим совершенно понятный фрагмент:

```
…
if выражение
  then if выражение
            then оператор
            else оператор
…
```

✓ В операторе **if** после ключевых слов **then** и **else** может стоять *всего лишь один оператор*. Для снятия ограничения по количеству операторов следует употреблять составные операторы ***begin .. end***.

Заметим, что ключевые слова ***begin .. end*** точно очерчивают область действия каждой ветви. Поэтому, если вложенный оператор располагается в пределах составного оператора, то больших проблем в определении зоны действия вложенных операторов **If** не возникает.

Обобщенные формы записи оптимальной записи для стандартных случаев оператора **If** приведены в *табл. 5.1.*

Таблица 5.1 - Наиболее популярные варианты записи оператора *If*

Количество операторов в ветви		Обобщенная форма оператора *If*
then	*else*	
один	один	*If* выражение *then* оператор *else* оператор
несколько	один	*If* выражение *then begin* оператор; оператор; … оператор *end* *else* оператор
один	несколько	*If* выражение *then* оператор *else begin* оператор; оператор; … оператор *end*
несколько	несколько	*If* выражение *then begin* оператор; оператор; … оператор *end* *else begin* оператор; оператор; … оператор *end*

5.5 Оператор варианта (***case***)

Условный оператор ***if*** позволяет выбирать только одно из двух возможных действий в зависимости от результата вычисления логического выражения. Оператор выбора (варианта) ***case*** (*рис. 5.4*) является обобщением оператора ***if***. Он дает возможность выполнять одно или несколько действий в зависимости от значения выражения-переключателя (селектора).

Оператор варианта ***case*** состоит из выражения-переключателя[3] и списка операторов, каждому из которых предшествует одна или более констант (они называются константами выбора), и (возможно) ветви ***else***.

---

[3] Переключатель (селектор) должен иметь порядковый тип (***shortint, integer, word, byte*** и ***Char***) размером в байт или слово. К любому из них можно применить функцию ***Ord(X)***, возвращающую порядковый номер значения выражения ***X***.

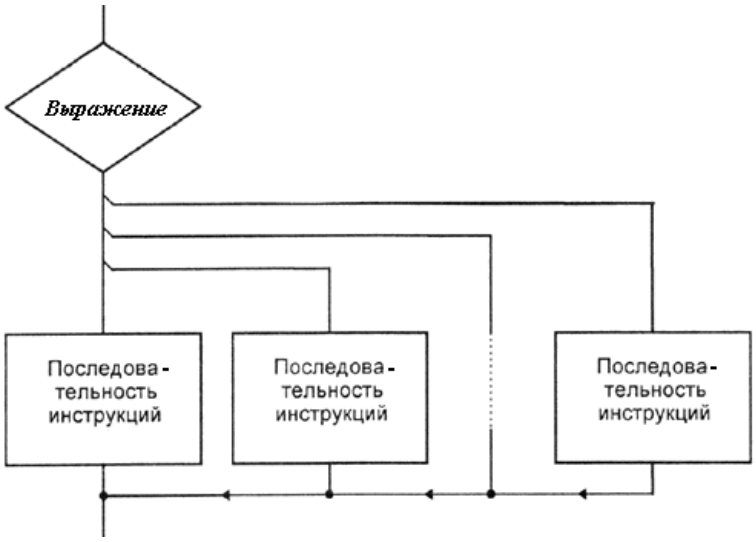


Рисунок 5.4 - Блок-схема оператора выбора ***case***

Значение порядкового типа не должно превышать 65535, и, следовательно, вещественный, строковый и длинный целый типы являются недопустимыми типами переключателя. Все константы выбора должны быть уникальными и иметь порядковый тип, совместимый с типом переключателя.

Общий вид оператора ***case***:

```
case <Переключатель> of {<Переключатель> – выражение порядкового типа}
   <Список констант выбора 1>: <Оператор 1>;
   <Список констант выбора 2>: <Оператор 2>;
   …
   <Список констант выбора N>: <Оператор N>
else  <Оператор E>
end;
```

Оператор варианта ***case*** приводит к выполнению оператора, которому предшествует константа выбора, равная *вычисленному значению переключателя,* или *диапазону выбора*[4], в котором находится значение переключателя. Если такой константы выбора или такого диапазона выбора не существует, но присутствует ветвь ***else***, то выполнятся оператор, следующий за ключевым словом ***else*** (Оператор E). Если же ветвь ***else*** отсутствует, то никакой оператор не выполняется.

Приведем характерные примеры использования оператора варианта:

1. Выполнение над операндами ***A*** и ***B*** операции, заданной значением переключателя Operator:

```
case Operator of  {Operator – переменная символьного типа Char}
   '+':  C := A + B;    {выполняется сложение, если Operator = '+'}
   '-':  C := A - B;    {выполняется вычитание, если Operator = '-'}
   '*':  C:= A * B;     {выполняется умножение, если Operator = '*'}
   '/':  C := A / B     {выполняется деление, если Operator = '/'}
end;
```

[4] При описании диапазона (интервала) значений какого-либо порядкового типа указывают наименьшее и наибольшее значения, допустимые для этого типа, разделенные знаком **..** (две точки). Например, ***-128 .. 127, 'a' .. 'z', 1 .. 100.***

2. Анализ целого числа в диапазоне от 0 .. 100:

```
case I of {I – переменная целого типа Integer}
   0, 2, 4, 6, 8: Writeln('Четная цифра');
   1, 3, 5, 7, 9: Writeln('Нечетная цифра');
       10..100: Writeln('Цифра между 10 и 100')
           else Writeln('Цифра не принадлежит диапазону 0 .. 100')
end;
```

3. Возможны 3 варианта использования оператора выбора **Case** для определения дня недели.

Пусть переменная **Num_day** типа **Integer** хранит номер дня недели (от 1 до 7), а переменная **day** типа **String** – строку символов.

```
case Num_day of
1, 2, 3, 4, 5: day := 'Рабочий день. ' ; {Перечисляем рабочие дни}
              6: day := 'Суббота!';
              7: day := 'Воскресенье!'
end;
```

```
case Num_day of
  1..5: day := 'Рабочий день.'; {Задаем диапазон рабочих дней}
     6: day := 'Суббота!';
     7: day := 'Воскресенье!'
end;
```

```
case Num_day of
     6: day := 'Суббота!';
     7: day := 'Воскресенье!';
 else day := 'Рабочий день.'  {Рабочие дни – ни 6-й и ни 7-й}
end;
```

5.6 Безусловное изменение естественного порядка выполнения операторов

Можно теоретически показать, что рассмотренных операторов вполне достаточно для написания программ любой сложности. В этом отношении наличие в языке операторов перехода кажется излишним. Более того, современная технология структурного программирования основана на принципе «программировать без **GoTo**» [5].

---

[5] Правила структурного программирования требуют минимального использования операторов безусловного перехода, которые делают программу запутанной, малопонятной и сложной в отладке. Кроме того, при их употреблении резко возрастает вероятность появления ошибок в логике программы. Богатый выбор условных и

Организацию безусловного перехода к оператору с меткой ***N*** осуществляют с помощью оператора безусловного перехода ***GoTo***.

Метка в ***Borland Pascal*** - это произвольный идентификатор, позволяющий именовать некоторый оператор программы и таким образом ссылаться на него. В целях совместимости со стандартным языком Паскаль в языке ***Borland Pascal*** допускается в качестве меток использование также целых чисел без знака в диапазоне от 0 до 9999.

Метка располагается непосредственно перед помечаемым оператором и отделяется от него двоеточием. Оператор можно помечать несколькими метками, которые в этом случае отделяются друг от друга двоеточием. Перед тем как появиться в программе, метка должна быть описана с помощью предложения ***Label N,*** где ***N*** - метка.

Обязательным является требование, чтобы метка, указанная в операторе перехода, располагалась в том же блоке или модуле, что и сам оператор ***GoTo***. Следовательно, запрещены как передача управления внутрь процедуры или функции, так и передача управления из подпрограммы в другую подпрограмму или вызывающую программу.

Оператор, следующий непосредственно за оператором безусловной передачи управления ***GoTo N***, обязательно должен иметь метку, так как иначе он не сможет выполниться никогда.

Пример 5.4. Программа, использующая операторы безусловного перехода

```
Program DemoGoTo; {Демонстрация работы операторов безусловного перехода}
Label  {Раздел объявления меток}
   Metka1, Metka2;
Begin {Начало основного блока программы DemoGoTo}
  writeln('Демонстрация применения операторов безусловного перехода');
  GoTo Metka2; {Изменение естественного порядка выполнения операторов программы - безусловный переход к оператору с меткой Metka2}
  Metka1: writeln('Работает оператор с меткой Metka1');
          Halt; {Завершение работы программы DemoGoTo}
  Metka2: writeln('Работает оператор с меткой Metka2');
          GoTo Metka1 {Безусловный переход к оператору с меткой Metka1}
end. {Конец основного блока программы DemoGoTo}
```

6 Составление и отладка разветвленных программ

6.1 Разветвление вычислительного процесса более чем на два направления

В общем случае количество ветвей в алгоритме разветвляющейся структуры *не обязательно равно двум*. Для того, чтобы получить большее количество ветвей можно каждую из ветвей расщепить с помощью условного оператора ***if .. then .. else*** *еще на два направления*. Следует помнить, что после любого такого разветвления расщепленные ветви должны обязательно снова *слиться*.

циклических операторов на Паскале позволяет писать логически четкие программы практически *без меток и операторов* ***GoTo***.

Пример 6.1. Вычислить значение функции, заданной формулой:

$$z = \begin{cases} \sin x, & \text{если } x \le a; \\ \cos x, & \text{если } a < x < b; \\ tg\ x, & \text{если } x > b. \end{cases}$$

Решение.

Вычислительный процесс здесь следует разделить на три ветви. С помощью условного блока «развилка» можно выбрать первую из ветвей расчета: при выполнении условия ***x ≤ a*** вычисляется ***z = sin(x)***. В случае невыполнения этого условия (т.е. ***x > a***), нужно применить еще один условный блок (*рис. 6.1*), позволяющий расщепить ветвь ***else*** еще на два направления. После вычислений по любой из формул осуществляется слияние всех ветвей к общему продолжению.

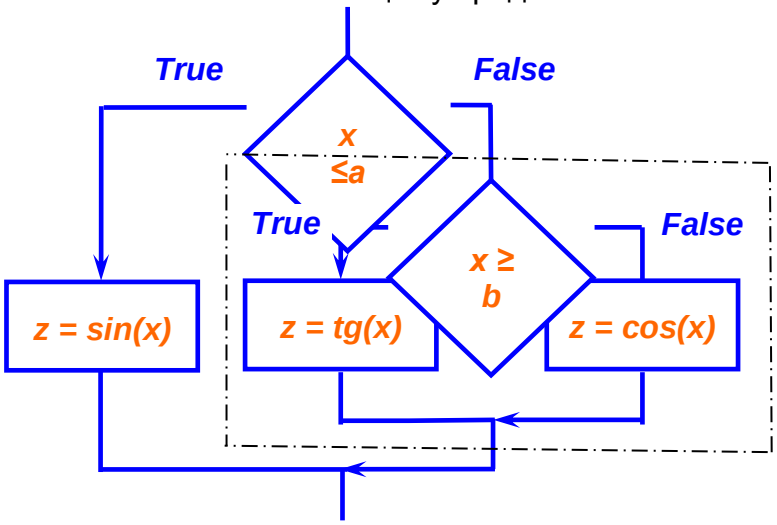


Рисунок 6.1 - Разделение вычислительного процесса на три направления

Фрагмент программы на ***Borland Pascal*** (без ввода-вывода и объявления переменных) может выглядеть так:

```
...
if  x <= a   {проверяем первое условие x <= a}
   then z := sin(x)  {если  первое условие выполняется, то z := sin(x)}
  else if  x >= b      {иначе проверяем третье условие}
         then z  :=  tg(x)  {если третье условие выполняется, то z  :=  tg(x) }
         else z := cos(x); {иначе, если третье условие не выполняется, z := cos(x)}
...
```

6.2 Ошибки в Вашей программе

Текст новой программы на диск необходимо обязательно сохранить до (!) первого запуска программы на выполнение, так как из-за ошибок в программе или сбоя компьютера набранный текст может быть безвозвратно утрачен…

Отладка - это процесс поиска и исправления ошибок в программе, препятствующих корректной работе программы. Существует три основных типа ошибок: ошибки этапа компиляции, ошибки этапа выполнения и логические ошибки.

Ошибки этапа компиляции (*Compile-time error*) или синтаксические ошибки происходят, когда Ваша программа записана с нарушением правил синтаксиса Пас-

каля. Когда компилятор встречает оператор, который он не может распознать, соответствующий файл выводится в окне редактирования, курсор позиционируется на то место, которое не понял компилятор, и выводится сообщение об ошибке. Перечень ошибок времени выполнения приведен в [4].

Весьма часто на этапе компиляции выявляются ошибки набора (опечатки), пропущенные точки с запятой, ссылки на неописанные переменные, передача неверного числа (или типа) параметров процедуры или функции и присваивание переменной значений неверного типа.

Неочевидны ошибки, связанные с неверной записью составного оператора. Каждой открывающей скобке составного оператора ***begin*** должна соответствовать и закрывающая скобка ***end***. Если ее нет, может курсор позиционироваться на место, где с Вашей точки зрения ошибок нет. Во избежание таких ошибок разумно вносить в программу обе операторные скобки сразу, непосредственно друг за другом.

После исправления синтаксических ошибок Вы должны повторить компиляцию снова.

Ошибки этапа выполнения (семантические ошибки) происходят, когда Ваша откомпилированная программа при выполнении делает что-то недопустимое. Другими словами, программа содержит допустимые операторы Паскаля, но при выполнении операторов что-то происходит неверно. Например, Ваша программа может пытаться открыть для ввода несуществующий файл или выполнить деление на ноль, или не выполняет преобразования (например, строки символов в число), если преобразуемая величина не может быть приведена к требуемому виду.

Когда программа ***Borland Pascal*** обнаруживает такую ошибку, она завершает выполнение и выводит сообщение следующего вида:

Run-time error ## at seg:ofs

Здесь ## номер ошибки, а ***seg:ofs*** адрес сегмента и смещения оператора, вызвавшего ошибку (как и в случае синтаксических ошибок). Перечень ошибок времени выполнения приведен в [4].

Логические (алгоритмические) ошибки - это ошибки проектирования и реализации программы. Компиляция программы, в которой есть алгоритмическая ошибка, завершается успешно. При пробных запусках программа ведет себя нормально, однако при анализе результата выясняется, что он неверный. Другими словами, Ваши операторы допустимы и что-то делают, но не то, что Вы предполагали.

Для того чтобы устранить алгоритмическую ошибку," приходится анализировать алгоритм, вручную "прокручивать" его выполнение. Эти ошибки часто трудно отследить, поскольку **IDE** не может найти их автоматически, как синтаксические и семантические ошибки. К счастью, **IDE** включает в себя средства отладки, помогающие Вам найти логические ошибки.

Этап отладки можно считать законченным, если программа правильно работает на одном-двух наборах тестовых входных данных. Этап тестирования особенно важен, если вы предполагаете, что Вашей программой будут пользоваться другие. В этом случае следует проверить, как ведет себя программа на как можно большем количестве входных наборов данных, в том числе и на заведомо неверных.

6.3 Методы отладки программ в интегрированной среде Borland Pascal

Иногда, когда программа делает что-то непредвиденное, причина достаточно очевидна, и Вы можете быстро внести исправления в исходный текст программы. Но многие ошибки более трудноуловимы и зависят от работы различных частей программы. В этих случаях лучше всего остановить Вашу программу в заданной точке, и затем пройти ее шаг за шагом, просматривая состояние переменных и выражений на каждом шагу. Такое управляемое выполнение - важнейший элемент отладки.

В меню ***Run*** имеются команды выполнения *по шагам* ***Step Over*** и *трассировки* ***Trace Into***, которые дают возможность *построчного выполнения программы*. Подсвечивая строку выделенным цветом, встроенный отладчик всегда сообщает Вам, какую строку Вы выполняете следующей (строка выполнения).

Выполнение программы по шагам или ее трассировка могут помочь Вам найти ошибки в алгоритме программы. Например, операторы, где происходит деление на ноль, переполнение и т.п. Тем не менее, кроме этого обычно хотелось бы знать, что происходит на каждом шаге со значениями некоторых переменных, сравнивая их со значениями этих же переменных в заранее рассчитанном тестовом примере.

Встроенный отладчик позволяет Вам просматривать значения переменных, выражений и структур данных. С помощью команды ***Watch***[6] в меню ***Debug*** в окне ***Watches*** (Просмотр) Вы можете просматривать текущие значения отслеживаемых элементов, добавлять их или удалять[7].

Еще один вариант наблюдения за состоянием объектов программы использование диалогового окна ***Evaluate and Modify*** (Вычисление и модификация). Оно вызывается командой ***Debug/Evaluate/Modify*** или комбинацией клавиш ***Ctrl+F4***. В этом диалоговом окне Вы можете не только проверять значения переменных и выражений, но и изменять значения любых переменных, что позволяет Вам проверять реакцию программы на различные условия.

Наименьшим выполняемым элементом при отладке является *строка*. Поэтому, если на одной строке программы содержится несколько операторов Паскаля, Вы не сможете отладить эти операторы индивидуально. С другой стороны, с целью отладки Вы можете разбить оператор на несколько строк, которые будут выполняться за один шаг.

Выполнение по шагам - это простейший способ выполнения программы по элементарным фрагментам. Выбор команды ***Run/Step Over*** или нажатие клавиши ***F8*** вызывает выполнение отладчиком всего кода в операторе, указанном строкой выполнения, включая любые вызываемые на ней процедуры или функции, пока управление не вернется обратно к вам. После этого строка выполнения указывает следующий выполняемый оператор.

Трассировка программы во многом аналогична ее выполнению по шагам. Единственное отличие команды трассировки ***Run/Trace Into*** (альтернатива - нажатие клавиши ***F7***) состоит в том, что когда встречается оператор вызова процедуры или функции, то выполняется также трассировка *по шагам* и этой процедуры или функции, а по завершении выполнения подпрограммы управление передается на следующую после ее вызова строку. Таким образом, пошаговое выполнение или трассировка выполняют фактически одно и то же действие, кроме того случая, когда строка выполнения находится под строкой вызова процедуры или функции.

Часть программы может быть выполнена *автоматически*, до определенной строки (*точки останова*), а затем возможна ее трассировка или выполнение по шагам. Существует два способа указания, что программу нужно выполнить автоматически до определенной точки, а затем остановить.

---

[6] Чтобы добавить в окно просмотра некоторую переменную, необходимо установить на нее курсор и затем нажать ***Ctrl+F7*** (или использовать команду меню ***Debug/Add Watch..***).

[7] Чтобы на экране одновременно можно было следить за перемещением строки выполнения в исходном тексте отлаживаемой программы и контролировать изменения значений переменных в окне ***Watches***, воспользуйтесь командой ***Window/Tile***.

Первый способ состоит в том, чтобы установить курсор на строку в программе, где вы хотите остановиться, затем выбрать в меню ***Run*** команду ***Go to Cursor*** (***F4*** - Выполнение до позиции курсора). В точке останова Вы можете проверить значения и далее продолжать выполнение непрерывно или по шагам.

Второй способ состоит в том, чтобы установить в определенной строке Вашей программы точку останова (***BreakPoint***) с помощью команды ***Ctrl+F8***. При этом строка будет выделена **красным цветом**. Повторное нажатие ***Ctrl+F8*** позволяет убрать точку останова из этой строки. Если Вы запустите свою программу на автоматическое выполнение (***Ctrl+F9***), то она остановится перед выполнением оператора с точкой останова. В отличие от первого способа, Вы можете установить в своей программе *несколько* точек останова. Список точек останова можно просматривать и корректировать с помощью команды ***Debug /BreakPoints…***

Но даже если Вы и не введете точки останова, то все равно сможете войти в отладчик из **IDE** при выполнении программы. В любой момент работы программы нажмите клавиши ***Ctrl+Break***. Отладчик находит позицию в исходном коде, где Вы прервали программу. Конечно, место остановки работы программы будет абсолютно случайным. Как и в случае обычной точки останова, Вы можете затем выполнить программу по шагам или трассировать ее.

При выполнении программы по шагам часто полезно держать под наблюдением вывод программы (работу операторов ***Write***), который осуществляется на *экран пользователя*. В прикладной программе (приложении)операционной системы ***Windows*** это достаточно просто, так как программа сама уже выполняется в отдельном окне. Однако при работе в **MS DOS**, не поддерживающей многозадачный режим работы, это не так просто. Но разработчики ***Borland Pascal*** предоставили Вам несколько способов просмотра экрана пользователя.

В любой момент сеанса отладки Вы можете выполнять переключение экрана **IDE** и экрана пользователя. Чтобы вывести экран пользователя, нажмите клавиши ***Alt+F5***. Чтобы вернуться в **IDE**, нажмите любую клавишу или щелкните "мышью".

При выполнении программы отладчик также может переключать экраны автоматически. Управлять характером переключения экранов вы можете с помощью параметров ***Display Swapping*** (Переключение экрана) диалогового окна ***Debugger*** (вызывается командой меню ***Options/Debugger..***). По умолчанию задано эффективное переключение (***Smart***). Это означает, что экран пользователя выводится только в том случае, если выполняемый оператор выводит информацию на экран или вызывает процедуру (даже если эта процедура ничего на экран не выводит). После завершения вывода экран переключается обратно в **IDE**.

Вы можете также сообщить отладчику, что переключать экран нужно на каждой строке, независимо от вывода, или не переключать их вовсе. Переключение экранов для каждой строки полезно использовать, если Ваша программа посылает информацию непосредственно на экран, что может затереть содержимое экрана **IDE**.

Кроме этого, **IDE** для **MS DOS** предусматривает для экрана пользователя окно, которое называется окном вывода. Выбрав команду меню ***Debug/Output***, Вы можете открыть (вывести на переднем плане) активное окно, содержащее вывод программы. Настроить размер этого окна можно аналогично окну редактирования.

В ходе сеанса отладки иногда хотелось бы начать все сначала. Выберите команду ***Run/Reset Program*** или нажмите клавиши ***Ctrl+F2***. Это приведет к полному сбросу, так что выполнение по шагам, или трассировка начнется в начале основной программы.

7 Варианты заданий для составления разветвленных программ

7.1 Условие задания и рекомендации по его выполнению

Импульсный источник напряжения ***U(t)*** с внутренним сопротивлением R_U, заданный на периоде $0 .. T_U$, подключен к сопротивлению нагрузки R_H (*рис. 7.1*).

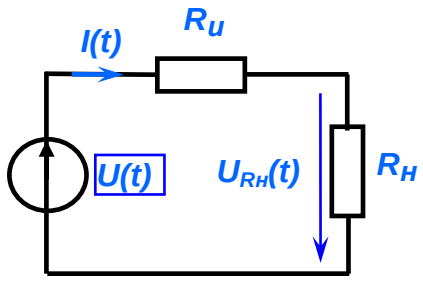


Рисунок 7.1 – Расчетная электрическая схема

По формуле

$$U_{RH}(t) = U(t) * RH / (RH + Ru)$$

рассчитать напряжение на нагрузке $U_{RH}(t)$ для любого веденного времени t ($0 < t < T_U$). Варианты функции напряжения ***U(t)*** импульсного источника заданы графически и приведены ниже.

Для определения варианта формулы расчета на основании анализа значения времени ***t*** воспользуйтесь расщеплением одной из ветвей условного оператора ***if .. then .. else*** (см. *пример 6.1*).

Еще один способ разветвления – с помощью оператора выбора ***case*** по номеру расчетного интервала ***Nu*** = ***0..5.*** Так как в нашем случае все интервалы одинаковы, то его нетрудно получить отбрасыванием дробной части ***Nu*** = ***t / A***.

Элементы, из которых состоят варианты функций источника напряжений - отрезки прямых и части окружностей. Напомним, что уравнение прямой, проходящей через начало координат ***y = k * x.*** В нашем случае угловой коэффициент может принимать значения ***k = {-1, +1}.***

Уравнение окружности с центром в начале координат $y = \pm\sqrt{R^2 - x^2}$, где ***R*** – радиус, ***x, y*** – координаты окружности. Знак "+" соответствует верхней полуокружности, знак "-" - нижней.

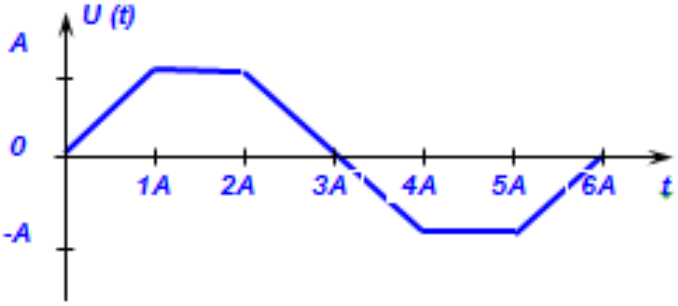
Значение параметра ***A*** следует ввести с клавиатуры. Для всех вариантов период импульса $T_U = 6 * A.$

Для сдвига кривых по осям координат используйте формулы переноса координат.

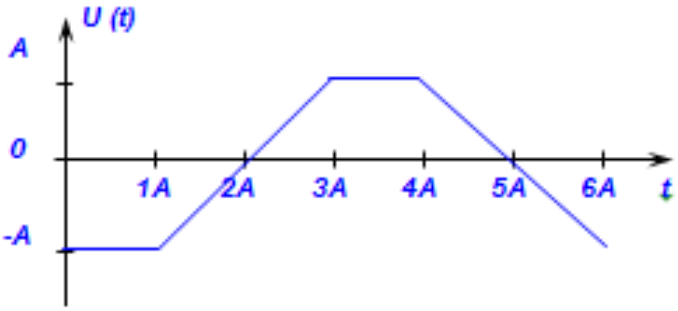
Обязательно проверяйте правильность расчета напряжения предварительной подстановкой соответствующих значений ***t*** на границах интервалов определения составляющих расчетной функции.

7.2 Варианты функций источника напряжения

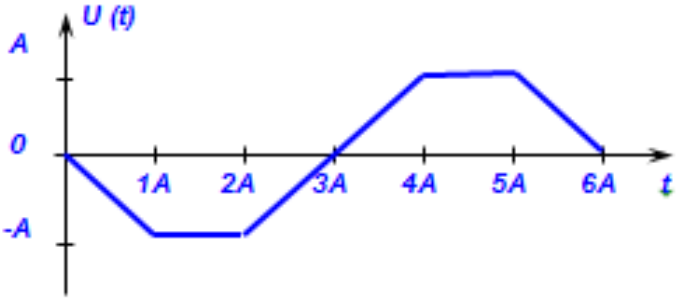
1



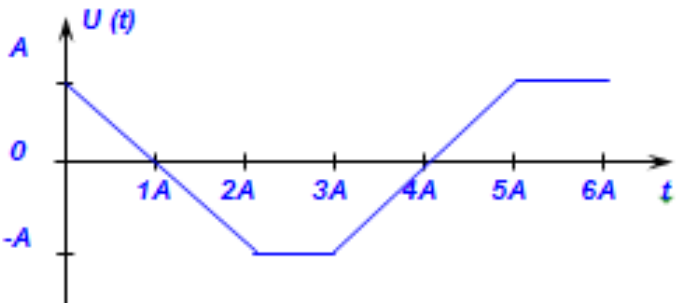
2



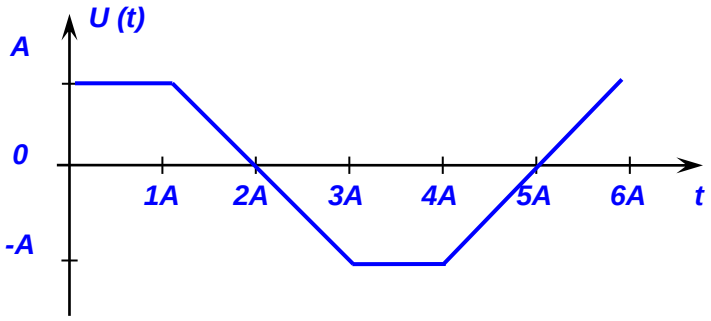
3



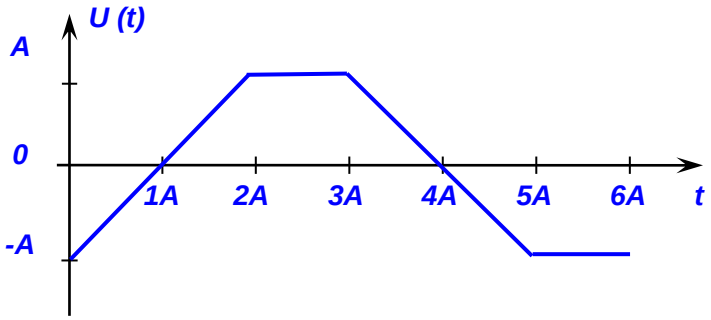
4



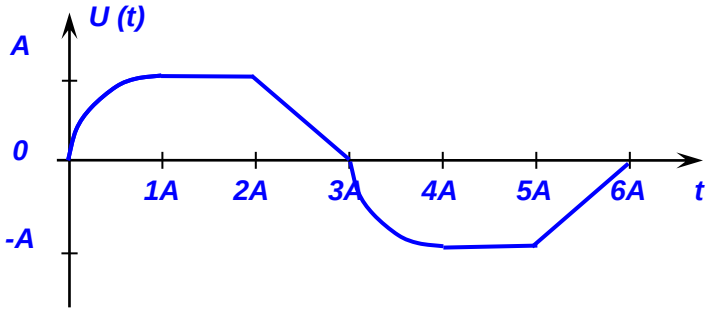
5


U (t)
A
0
-A
1A
2A
3A
4A
5A
6A
t

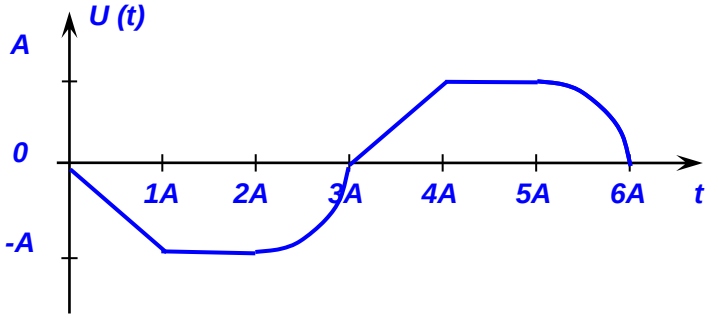
6


U (t)
A
0
-A
1A
2A
3A
4A
5A
6A
t

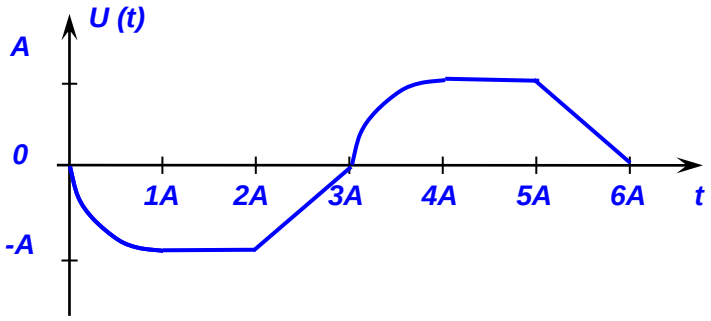
7


U (t)
A
0
-A
1A
2A
3A
4A
5A
6A
t

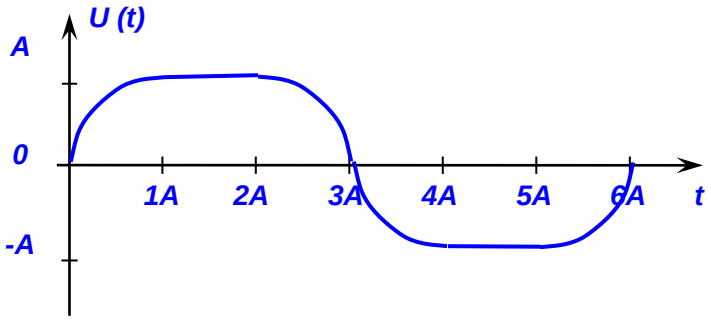
8


U (t)
A
0
-A
1A
2A
3A
4A
5A
6A
t

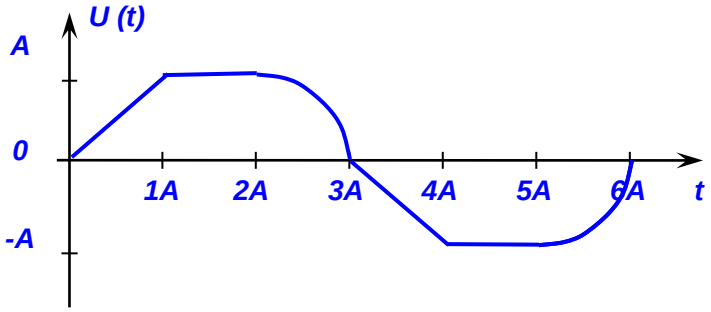
9


U (t)
A
0
-A
1A
2A
3A
4A
5A
6A
t

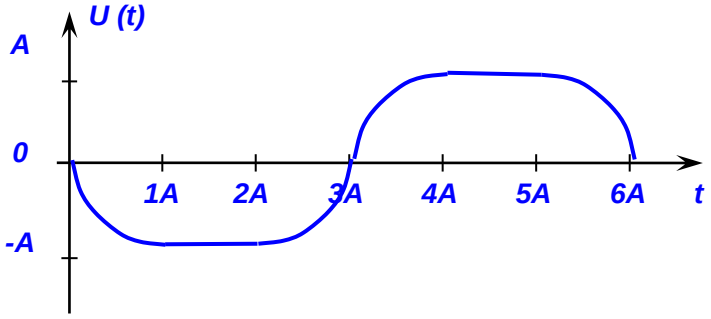
10


U (t)
A
0
-A
1A
2A
3A
4A
5A
6A
t

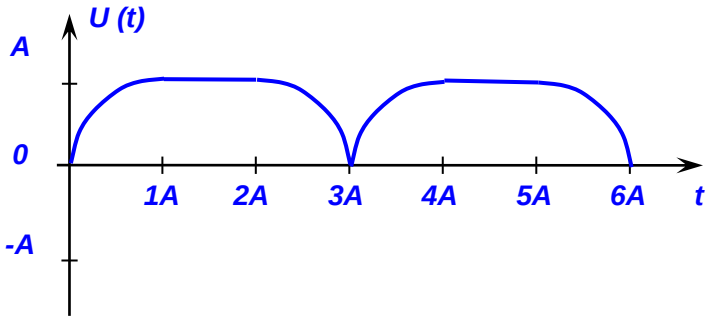
11


U (t)
A
0
-A
1A
2A
3A
4A
5A
6A
t

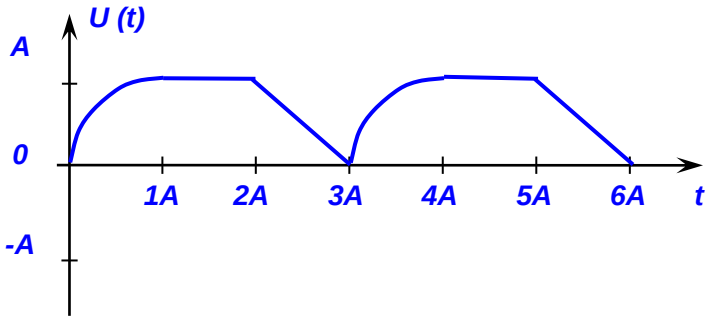
12


U (t)
A
0
-A
1A
2A
3A
4A
5A
6A
t

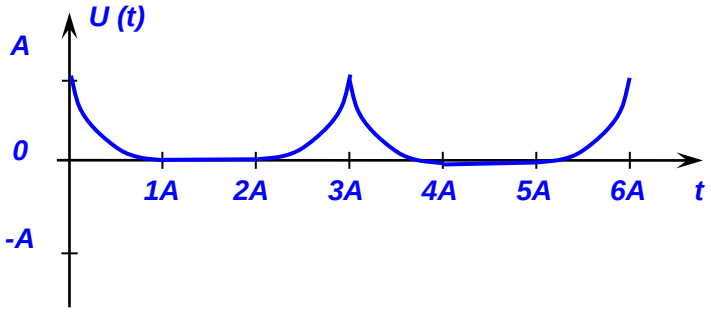
13


U (t)
A
0
-A
1A
2A
3A
4A
5A
6A
t

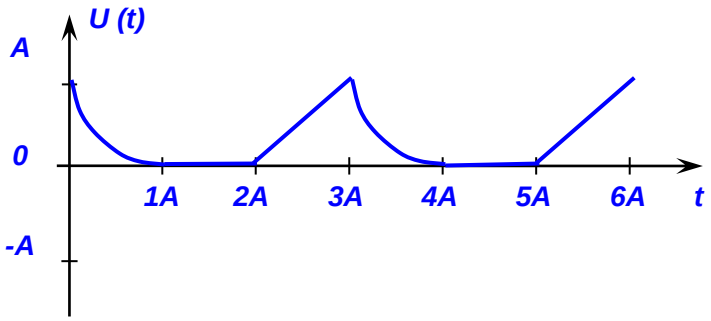
14


U (t)
A
0
-A
1A
2A
3A
4A
5A
6A
t

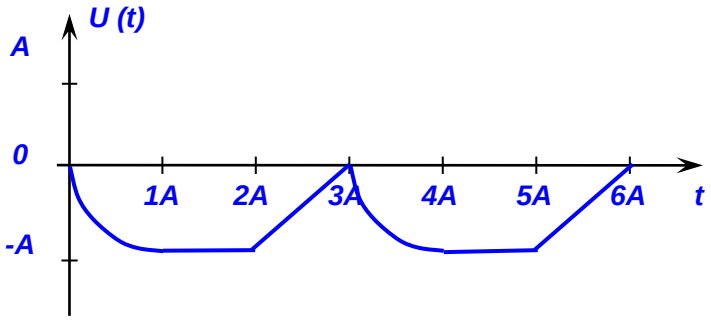
15


U (t)
A
0
-A
1A
2A
3A
4A
5A
6A
t

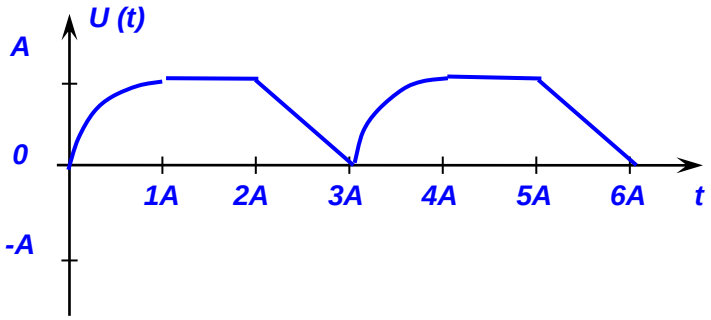
16


U (t)
A
0
-A
1A
2A
3A
4A
5A
6A
t

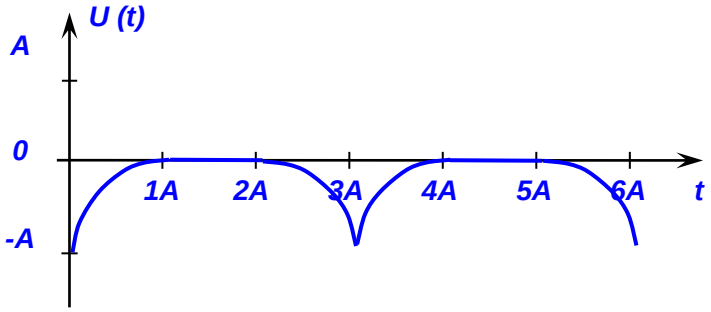
17


U (t)
A
0
-A
1A
2A
3A
4A
5A
6A
t

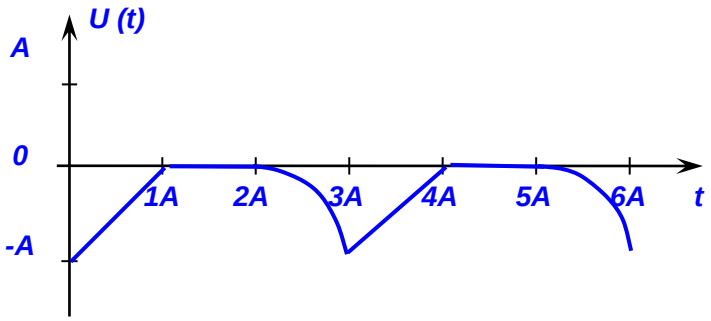
18


U (t)
A
0
-A
1A
2A
3A
4A
5A
6A
t

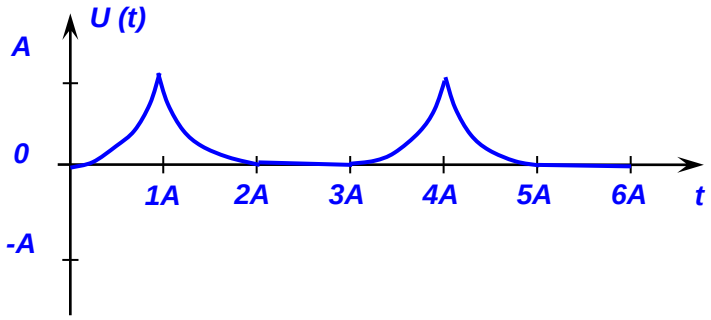
19


U (t)
A
0
-A
1A
2A
3A
4A
5A
6A
t

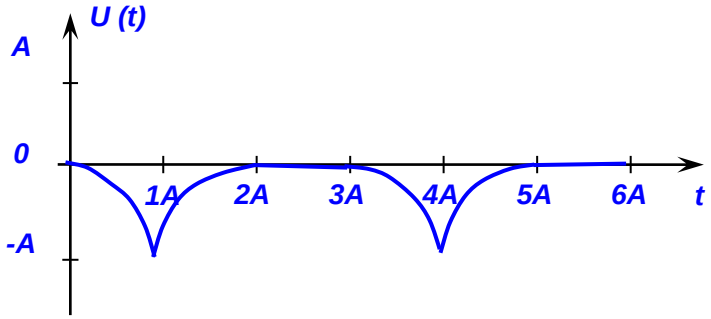
20


U (t)
A
0
-A
1A
2A
3A
4A
5A
6A
t

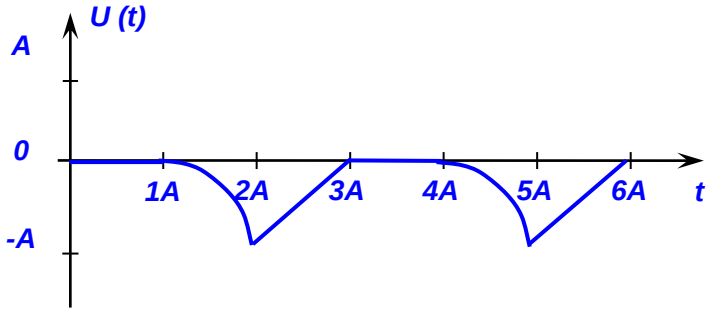
21


U (t)
A
0
-A
1A
2A
3A
4A
5A
6A
t

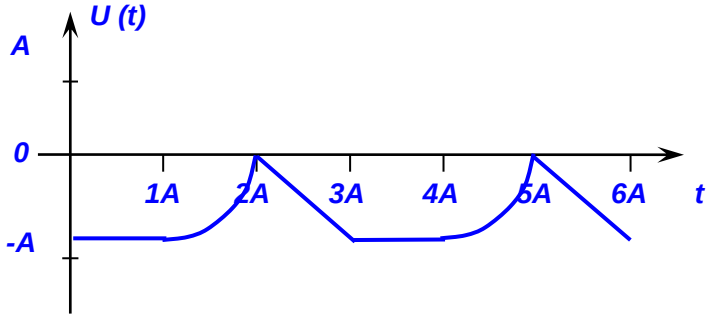
22


U (t)
A
0
-A
1A
2A
3A
4A
5A
6A
t

23


U (t)
A
0
-A
1A
2A
3A
4A
5A
6A
t

24


U (t)
A
0
-A
1A
2A
3A
4A
5A
6A
t

8 Список литературы

1. **Рапаков Г.Г., Ржеуцкая С.Ю.** *Программирование на языке Pascal.* — СПб. : БХВ-Петербург, 2004. - 480 с.
2. **Фаронов , В.В.** *Программирование на персональных ЭВМ в среде Турбо-Паскаль. :, .* - М. : Изд-во МГТУ, 1990.-580 с.
3. **Павловская , Т.А.** *Паскаль. Программирование на языке высокого уровня: Учебник для вузов.* — СПб. : Питер, 2007. — 393 с.
4. **Фаронов , В.В.** *Турбо Паскаль 7.0. Начальный курс. Учебное пособие.* -М. : ОМД Групп, 2003. - 616 с.
5. **Рютген Т., Франкен Г.** *Турбо Паскаль 7.0 .* — К. : Торгово-издательское бюро BHV, 1996-448 с.
6. **Попов, В.Б.** *Паскаль и Дельфи. Самоучитель.* — СПб. : Питер, 2004. — 544 с.