Автономная некоммерческая общеобразовательная организация "Физтех-лицей" имени ПЛ. Капицы

Руководство пользователя
Язык программирования ExQu

Выполнили ученицы класса 11«И»:
Бочкова Валерия Андреевна
Мулина Виктория Павловна

Долгопрудный, 2024

Содержание

Аннотация **3**

4

Введение **4**

Руководство пользователя **5**

1. Общие сведения о компиляторе . . . 5

1.1 Что такое компилятор и для чего он нужен . . . 5

2. Основные типы данных функции компилятора . . . 6

2.1 Описание основных специальных знаков . . . 6

2.2 Описание типов данных языка . . . 6

Integer . . . 6

Double . . . 6

Line . . . 7

Bool . . . 7

Nochar . . . 8

Vector . . . 8

2.3 Описание основных функций языка . . . 8

Оператор if . . . 8

Оператор for . . . 9

Оператор while . . . 10

. . . 11

Оператор toggle . . . 11

3. Как использовать компилятор . . . 12

3.1 Как написать программу и подготовить её для компиляции . . . 12

3.2 Примеры простых команд для начала работы . . . 12

. . . 13

4. Работа с ошибками . . . 13

4.1 Что делать, если компилятор сообщает об ошибках в коде. . . . 13

4.2 Как читать сообщения об ошибках и исправлять их. . . . 13

. . . 14

5. Рекомендации и полезные советы . . . 14

5.1 Советы для начинающих, чтобы избежать ошибок . . . 14

6. Часто задаваемые вопросы . . . 15

6.1 Что делать, если программа не работает? . . . 15

6.2 Как найти ошибки? . . . 15

Приложение 1. Обработка ошибок **16**

Наши контакты **18**

Аннотация

Этот документ создан для того, чтобы помочь пользователям понять, как работать с компилятором, написанным на C++. Компилятор — это инструмент, который переводит текст программы, написанный человеком, в язык, понятный компьютеру. Этот документ подробно объясняет, как использовать компилятор, какие команды доступны и как настроить его под свои задачи.

Вот основные разделы документа:

1. **Общие сведения о компиляторе**
   - **Что такое компилятор и для чего он нужен.**
2. **Основные функции компилятора**

   Описание основных специальных знаков.
   - **Описание типов данных языка.**
   - **Описание основных функций языка.**
1. **Как использовать компилятор**
   0. **Как написать программу и подготовить её для компиляции.**
   0. **Примеры простых команд для начала работы.**
2. **Работа с ошибками**
   0. **Что делать, если компилятор сообщает об ошибках в коде.**
   0. **Как читать сообщения об ошибках и исправлять их.**
2. **Рекомендации и полезные советы**
   0. **Советы для начинающих, чтобы избежать ошибок.**
2. **Часто задаваемые вопросы (FAQ)**
   0. **Ответы на самые популярные вопросы пользователей: что делать, если программа не работает, как найти ошибки.**

Как использовать этот документ:

] Для выполнения своей первой компиляции перейдите к разделу «Как использовать компилятор» Если компилятор сообщает об ошибке, обратитесь к разделу «Работа с ошибками». Если в процессе работы с продуктом возникают вопросы, перейдите в раздел «Часто задаваемые вопросы». В случае, если вы не нашли ответ на свой вопрос, можете связаться с нами по почте: *valeriab7002@gmail.com*

Документ создан для пользователей от начинающего уровня. Он поможет вам освоить работу с компилятором, использовать его эффективно и решать возникающие вопросы.

Введение

В мире программирования компиляторы играют ключевую роль — они становятся мостом между человеком и машиной. Представьте, что ваш код — это язык, на котором вы разговариваете с компьютером. Однако машина не понимает его напрямую. Здесь на помощь приходит компилятор: он превращает ваши идеи, записанные в виде кода, в понятные для компьютера команды.

Наш компилятор, написанный на C++, — это мощный инструмент, созданный для того, чтобы упростить этот процесс и дать пользователю возможность сосредоточиться на главном: создании функционального и качественного программного обеспечения. Компилятор берет на себя всю сложность преобразования кода и делает это быстро, эффективно и точно.

Этот документ подготовлен для тех, кто хочет не просто использовать компилятор, а уверенно им пользоваться. Также в документе есть разделы, которые помогут проще начать осваивать данный язык программирования, учитывая тонкости работы компилятора.

Работа с компилятором — это не только шаг в сторону реализации ваших идей, но и способ узнать больше о том, как работают технологии. Мы надеемся, что этот документ станет вашим надежным помощником, который направит вас на каждом этапе работы, от установки до решения сложных задач.

Руководство пользователя

1. Общие сведения о компиляторе

1.1 Что такое компилятор и для чего он нужен

Компилятор – это программное обеспечение, которое преобразует исходный код, написанный на одном языке программирования (высокоуровневом), в машинный код или промежуточное представление, понятное компьютеру. Этот процесс необходим для того, чтобы программы могли выполняться на конкретной аппаратной платформе.

Основное назначение компилятора – обеспечить связь между программистом и машиной. Он помогает разработчикам писать код на удобном для человека языке, автоматически преобразуя его в инструкции, которые способен выполнить процессор.

Назначение компилятора

Компиляторы выполняют ряд важных задач:

[F0B7? **Преобразование кода: перевод исходного текста программы в исполняемый файл или инструкции.**

2. **Обнаружение ошибок: проверка кода на наличие синтаксических и семантических ошибок.**

Основные функции компилятора

1. **Анализ исходного кода: компилятор проверяет структуру программы, чтобы убедиться в правильности её написания.**

- ***Лексический анализ*** **– разбиение кода на лексемы.**

1. - ***Синтаксический анализ*** **– проверка на соответствие правилам языка.**
   - ***Семантический анализ*** **– проверка логики и смысла кода.**

2. **Генерация объектного кода: преобразование кода в промежуточное или машинное представление.**

3. **Оптимизация: минимизация использования ресурсов, таких как процессорное время и память.**

4. **Сборка программы: компилятор объединяет сгенерированный объектный код с библиотеками и другими модулями.**

Компиляторы являются важнейшей частью инструментария разработчиков программного обеспечения. Они позволяют автоматизировать перевод идей программиста в понятные компьютеру команды, ускоряя процесс разработки и обеспечивая высокое качество конечных продуктов.

2. Основные типы данных функции компилятора

2.1 Описание основных специальных знаков

! - знак разделения операторов программы.

? - знак для выделения области комментирования. Первое употребление знака — начало области комментирования, второе — окончание области комментирования.

- знак определения конца кода программы.

2.2 Описание типов данных языка

Integer Тип данных int в C++ представляет собой один из примитивных типов, используемых для хранения целых чисел. Этот тип является стандартным и обеспечивает удобное представление для работы с арифметическими операциями.

Основные характеристики типа int:

- Размер: Обычно int занимает 4 байта (32 бита), но это может варьироваться в зависимости от архитектуры системы. На некоторых платформах это может быть 2 байта или даже 8 байт.

- Диапазон значений: Для 4-байтового int диапазон значений составляет от -4,294,967,294 до 4,294,967,294. Для положительных чисел диапазон будет от 0 до 4,294,967,294.

- Арифметические операции: С переменными типа int можно выполнять стандартные арифметические операции, такие как сложение, вычитание, умножение, деление и остаток от деления.

- Использование: int часто используется для счетчиков, индексов массивов, а также в случаях, когда требуется работа с целыми числами.

```
integer main() {
    integer a = 10!
    integer b = 20!
    integer sum = a + b!
    output(The amount: ")!
    output(sum)!
    return 0!
}
#
```

В этом примере объявлены две переменные типа integer, их значения суммируются, и результат выводится на экран.

Double Тип данных double в C++ используется для хранения чисел с плавающей запятой двойной точности. Он позволяет представлять более точные значения, чем тип float, и часто используется в научных и инженерных расчетах.

Основные характеристики типа double:

- Размер: Обычно double занимает 8 байт (64 бита), что позволяет хранить более точные значения по сравнению с float, который занимает 4 байта.

- Диапазон значений: Диапазон значений для double составляет примерно от -1.7E+308 до 1.7E+308, с точностью около 15-17 значащих цифр.

- Арифметические операции: С переменными типа double можно выполнять стандартные арифметические операции, такие как сложение, вычитание, умножение, деление и другие.

- Использование: double часто используется в случаях, когда требуется высокая точность, например, в финансовых расчетах, научных моделях и графических приложениях.

Пример использования:

```
integer main() {
    double a = 2456!
    double b = 37!
    double result_division= a / b!
    output(The result of the division: ")!
    output(result_division)!
```

```
    return 0!
}
#
```

В этом примере объявлены две переменные типа double, значение одной делится на другую, и результат сохраняется в дополнительную переменную и выводится на экран.

Line Тип данных line используется для хранения последовательностей символов, т.е. строк. Это встроенный тип, который позволяет удобно работать с текстовой информацией.

Основные характеристики типа line:

- Динамическое выделение памяти: Объект line управляет памятью автоматически, что позволяет добавлять, изменять и удалять символы без необходимости ручного управления памятью.

- Удобные методы: У line есть встроенный метод для работы со строками, такой как len(), характеризующий размер объекта этого типа.

- Конкатенация: line поддерживает операцию конкатенации с помощью оператора +, что позволяет легко объединять строки.

Пример использования:

```
integer main() {
    line name = "Andrew"!
    line greeting = "Hi, " + name + "!"!
    output(greeting)!
    return 0!
}
#
```

В этом примере создаются две строки: name и greeting. Затем строка greeting формируется путем объединения других строк, и результат выводится на экран.

Bool Тип данных bool используется для хранения логических значений. Он может принимать только два состояния: истина (true) и ложь (false). Этот тип данных полезен для работы с условными операциями и логическими выражениями.

Основные характеристики типа bool:

- Значения: bool может хранить только два значения: true и false.

- Размер: По стандарту, bool занимает по меньшей мере один байт памяти, хотя доступная реализация может выделять больше места для оптимизации.

- Использование в условиях: Тип bool часто используется в условных операторах (например, if, while), чтобы проверять, истинно или ложно условие.

Пример использования:

```
integer main() {
    bool isSunny = true!
    bool isRaining = false!
    if (isSunny) {
       output("It's sunny today!")!
    } else {
       if (isRaining) {
          output("It's rainy today!")!
       } else {
          output("The weather is unclear today!")!
       }
    }
    return 0!
}
#
```

В этом примере переменные isSunny и isRaining определяют, какая погода за окном, и программа выводит соответствующее сообщение в зависимости от их значений.

Nochar Тип данных nochar используется для представления отдельных символов. Каждый nochar занимает один байт памяти и может хранить символы из набора ASCII, а также другие символы из расширенных наборов.

Основные характеристики типа nochar:

- Размер: nochar имеет фиксированный размер в 1 байт, что соответствует 8 битам.

- Значения: nochar может хранить как символы, так и числовые значения, которые представляют эти символы. Например, символ 'A' имеет числовое представление 65 в ASCII.

- Специальные символы: nochar может хранить специальные символы.

Пример использования:

```
integer main() {
    nochar letter = 'A'!
    nochar digit = '5'!
    nochar specialNoChar = ' !'!
    output("Letter: ")!
    output(letter)!
    output("Digit: ")!
    output(digit)!
    output("Special character: ")!
    output(specialNoChar)!
   return 0!
}
#
```

В этом примере переменные letter, digit и specialChar хранят различные символы, и программа выводит их на экран.

Vector vector — это динамический массив, который автоматически управляет своей памятью. Он позволяет хранить элементы одного типа, обеспечивая при этом гибкость и удобство работы с массивами.

Основные характеристики вектора:

- Динамический размер: Вектор может изменять свой размер в процессе выполнения программы. Вы можете добавлять и удалять элементы, и вектор будет автоматически корректировать свою емкость.

- Представление элементов: Вектор хранит элементы в смежных участках памяти, что обеспечивает быструю произвольную доступность.

- Шаблонный класс: vector является шаблом, что позволяет использовать его для хранения объектов любых типов.

2.3 Описание основных функций языка

Оператор if Оператор if используется для выполнения условных проверок. Он позволяет выполнять блок кода, если заданное условие истинно (имеет значение true).

Синтаксис

Основной синтаксис оператора if выглядит следующим образом:

if (условие) {

? код, который выполняется, если условие истинно ?

}

Пример использования

```
integer main() {
    integer number = 5!
    ?Checking if the number is greater than zero?
    if (number > 0) {
       output("The number is positive.")!
    }
    return 0!
}
#
```

В этом примере, если переменная number больше нуля, программа выведет сообщение "Число положительное".

```
if (condition) {
   ? the code that is executed if the condition is true ?
} else {
   ? the code that is executed if the condition is false ?
}
```

Пример с else

```
integer main() {
   integer number = 5!
   ?Checking if the number is greater than zero?
   if (number > 0) {
       output("The number is positive.")!
   } else {
       output("The number is not positive.")!
   }
   return 0!
}
#
```

В этом случае, так как number меньше нуля, программа выведет "Число неположительное".

Множественные условия с else if

Для проверки нескольких условий можно использовать конструкции else if:

```
if (condition1) {
    ? the code for the first condition ?
} else {
    if (condition2) {
       ? the code for the second condition ?
    } else {
       ? the default code ?
    }
}
```

Таким образом, оператор if является основным инструментом для управления потоком выполнения программы на основе определенных условий.

Оператор for Оператор for используется для выполнения повторяющихся действий несколько раз. Он позволяет инициализировать переменные, задавать условие продолжения и обновлять переменные на каждой итерации в одном конструктивном выражении.

Синтаксис

Основной синтаксис оператора for выглядит следующим образом:

for (инициализация! условие! обновление) {

? код, который будет выполняться в цикле ?
}

- Инициализация - выполняется один раз перед началом цикла.
- Условие - проверяется перед каждой итерацией цикла, и выполнение продолжается, пока оно истинно.
- Обновление - выполняется после каждого прохода цикла.

Пример использования

Вот простой пример использования оператора for для вывода чисел от 1 до 5:

```
integer main() {
    for (integer i = 1! i <= 5! i+=1) {
        output("Number: " )!
        output(i)!
    }
    return 0!
}
#
```

В этом примере переменная i инициализируется значением 1, цикл продолжается, пока i меньше или равно 5, и на каждой итерации i увеличивается на 1.

Использование в массиве

Цикл for часто используется для перебора элементов массива:

```
integer main() {
     integer numbers[5]!
     for (int i = 0! i < 5! i+=1) {
       numbers[i] = i + 1!
     }
     return 0!
}
#
```

В этом примере массив numbers перебирается с использованием цикла for, и каждому элементу присваивается значение.

Обратите внимание на бесконечный цикл

Если условие цикла for всегда истинно, это может привести к бесконечному циклу:

```
for (int i = 0! i < 5! i-=1) {
    output("It's an endless cycle.")!
}
```

Поскольку i уменьшается, условие $i < 5$ всегда будет истинным, что вызовет бесконечный цикл.

Оператор for является мощным инструментом для управления циклическими процессами и предоставляет пользователю гибкость в настройке условий и итераций.

Оператор while Оператор while используется для выполнения блока кода несколько раз, пока заданное условие истинно. Он позволяет выполнять цикл до тех пор, пока условие не станет ложным.

Синтаксис

Основной синтаксис оператора while выглядит следующим образом:

```
while (condition) {
    ? the code that will be executed in the loop ?
}!
```

- Условие - выражение, которое проверяется перед каждой итерацией. Цикл будет продолжаться, пока это условие истинно.

Пример использования

Вот пример использования оператора while для вывода чисел от 1 до 5:

```
integer main() {
    integer i = 1!
    while (i <= 5) {
      Output("Number: ")!
      Output(i)!
        i+=1! ? update variable ?
    }!
    return 0!
}
#
```

В этом примере переменная i инициализируется значением 1, и цикл продолжается, пока i меньше или равно 5. На каждой итерации i увеличивается на 1.

Использование в случае, когда условие изначально ложно

Если условие изначально ложно, блок кода внутри цикла while не выполнится ни разу:

```
integer main() {
    integer j = 6!
    while (j < 5) {
      output("It will never come out.")!
    }!
    return 0!
}
#
```

В этом примере переменная j равна 6 и условие j $<$ 5 ложно, поэтому цикл не выполняется.

Обратите внимание на бесконечный цикл

Если условие всегда истинно, это может привести к бесконечному циклу:

```
integer main() {
    integer k = 1!
    while (k <= 5) {
       output("It's an endless cycle.")!
        ? There is no change in the k variable at this point ?
    }!
    return 0!
}
#
```

Поскольку в этом примере значение k никогда не изменяется, условие k $<=$ 5 всегда будет истинным, что вызовет бесконечный цикл.

Оператор while является полезным инструментом для управления потоками выполнения в C++. Он особенно удобен, когда количество итераций заранее неизвестно и зависит от динамического условия.

Оператор toggle Оператор toggle используется для проверки значения переменной (или выражения) и выполнения различного кода в зависимости от этого значения. Это упрощает выбор из множества возможных вариантов по сравнению с использованием нескольких операторов if.

Синтаксис

Основной синтаксис оператора toggle выглядит следующим образом:

```
toggle (expression) {
  choice (value1) {
     ? the code for the value1 ?
  }
  choice (value2) {
    ? the code for the value2 ?
  }
  default! {
```

```
    ? code if none of the values match ?
  }!
}!
```

- выражение - это переменная или выражение, которое будет оценено.
- choice - используется для задания конкретных значений, с которыми будет сравниваться выражение.
- default - блок, который выполняется, если ни одно из значений choice не совпадает с выражением.

Пример использования

Вот пример использования оператора toggle для выбора дня недели:

```
integer main() {
  integer day = 3!
  toggle (day) {
    case (1) {
      output("Monday")!
    }
    case (2) {
      output("Tuesday")!
    }
    case (3) {
      output("Wednesday")!
    }
   case (4) {
      output("Thursday")!
    }
   case (5) {
      output("Friday")!
    }
   default! {
     output("Unknown day")!
    }!
  }!
  return 0;
}
#
```

В этом примере переменная day равна 3, поэтому будет выведено "Среда".

Когда использовать toggle

Оператор toggle может быть удобен, когда имеется несколько фиксированных значений, с которыми нужно сравнивать одну и ту же переменную или выражение. Однако для более сложных условий или диапазонов значений лучше использовать if-else конструкции.

3. Как использовать компилятор

3.1 Как написать программу и подготовить её для компиляции

Чтобы скомпилировать свою первую программу, нужно найти файл с именем code.txt в папке проекта и написать в нем свою программу. Компилятор будет интерпретировать весь код, написанный в этом файле и выводить результат в терминале.

3.2 Примеры простых команд для начала работы

Вы можете воспользоваться нижеприведенными примерами для компиляции своей первой программы на языке ExQu:

```
integer main() {
   integer a!
   input(a)!
   while (a > 0) {
```

```
5         output(a)!
6         a = a - 1!
7     }!
8     return 0!
9 }
10 #
```

Данная программа предлагает пользователю ввести целочисленное число. После ввода его пользователем начнется цикл, который будет выводить все числа от введенного пользователем до нуля.

```
1 integer main() {
2     integer a!
3     input(a)!
4     if (a > 0) {
5       output("positive")!
6     }
7     else {
8      output("negative")!
9     }!
10 return 0!
11 }
12 #
```

Данная программа также предлагает пользователю ввести целочисленное число. Результатом программы будет определение: положительное это число или отрицательное.

4. Работа с ошибками

4.1 Что делать, если компилятор сообщает об ошибках в коде.

1. Проверьте сообщения об ошибках.

Первое, что нужно сделать, это внимательно прочитать сообщения об ошибках, которые предоставляет компилятор. Они содержат информацию о том, в чем именно проблема и на какой строке кода она возникла.

2. Проверьте синтаксис.

Убедитесь, что у вас правильный синтаксис. Ошибки в написании переменных, неправильные скобки или пропущенные точки с запятой или восклицательные знаки могут вызывать проблемы.

3. Проверьте типы данных.

Убедитесь, что используете правильные типы данных. Например, если ожидается целое число, а передается строка, это может вызвать ошибку.

4. Ищите в приложении.

Если ошибка не ясна, обратитесь к приложению 1. В данном приложении описываются все ошибки, которые обрабатываются компилятором. Вы можете посмотреть, что ожидала на вход программа и определить, в чем заключается ошибка.

5. Пишите тесты.

Тестирование кода может помочь выявить ошибки. Пишите юнит-тесты для проверки функциональности отдельных частей кода.

6. Поиск ошибок

Если вы не можете найти ошибку, попробуйте упростить код. Уберите ненужные части и проверьте, сохраняется ли ошибка. Это может помочь вам локализовать проблему.

4.2 Как читать сообщения об ошибках и исправлять их.

Понимание структуры сообщения об ошибках

Сообщения об ошибках обычно содержат следующие элементы:

- Тип ошибки: Это может быть синтаксическая ошибка, ошибка компиляции, ошибка времени выполнения и т.д.
- Номер строки: На какой строке кода произошла ошибка.
- Описание ошибки: Краткое и понятное описание проблемы.

Чтение сообщения

Внимательно прочитайте сообщение об ошибке. Например:

Ошибка: "В строке 15 ожидалась круглая открывающаяся скобка! Получено: }'"

Это указывает, что в строке 15 вы, вероятно, использовали неправильный тип скобки.

Исправление синтаксических ошибок

Если ошибка синтаксическая:

1. Найдите строку, указанную в сообщении.
2. Проверьте на наличие опечаток, пропущенных символов (например, скобок, точек с запятой или восклицательных знаков) и корректность написания команд.

Исправление логических ошибок

Если ошибка логическая или компиляционная:

1. Прочитайте описание ошибки и постарайтесь понять, какой логический вывод компилятор пытается сделать.
2. Проанализируйте код, ищите ошибки в логике (например, неверные условия или неинициализированные переменные).

Используйте приложение 1

Если сообщение об ошибке неясно, обратитесь к приложению 1 документации языка программирования. Там можно найти объяснения.

Примеры исправлений

- Если ошибка указывает на "неопределенная переменная", убедитесь, что вы объявили эту переменную.

int x! ? Объявление переменной ?

x = 10! ? Использование переменной ?

- Если сообщение о "несоответствии типов", проверьте, какие типы вы используете.

5. Рекомендации и полезные советы

5.1 Советы для начинающих, чтобы избежать ошибок

Понимание основ

- Изучите базовые принципы языка программирования, который вы выбираете. Это поможет понять синтаксис и структуру.

Чистый код

- Пишите чистый и структурированный код. Используйте понятные имена переменных и следите за отступами. Это поможет лучше понять код в будущем.

Рефакторинг

- Регулярно пересматривайте и улучшайте свой код. Избавляйтесь от избыточных строк и структур, которые больше не нужны.

Комментарии

- Оставляйте комментарии в коде, чтобы пояснить сложные участки. Это поможет не только вам, но и другим, кто может работать с вашим кодом.

Деление на мелкие задачи

- Разбивайте проект на мелкие задачи. Это облегчает фокусировку и позволяет легче находить и исправлять ошибки.

Использование систем контроля версий
- Используйте Git или другие системы контроля версий. Это поможет вам отслеживать изменения и возвращаться к предыдущим версиям кода в случае ошибок.

Тестирование
- Пишите тесты для вашего кода. Это поможет убедиться, что все работает как надо и упростит выявление ошибок.

Ищите помощь

- Не бойтесь обращаться за помощью к сообществу программистов. Используйте форумы, чаты или локальные группы для обмена опытом и знаниями.

Практика
- Постоянно практикуйтесь. Чем больше вы кодируете, тем более уверенно будете себя чувствовать. Решайте задачи на платформах вроде LeetCode или HackerRank.

Читайте код других
- Изучайте код опытных программистов. Это поможет вам увидеть разные подходы к решению задач и улучшить свои навыки.

Не бойтесь делать ошибки
- Ошибки — это часть процесса обучения. Относитесь к ним как к возможности улучшить свои навыки и учитесь на своих промахах.

6. Часто задаваемые вопросы

6.1 Что делать, если программа не работает?

1. Проверка ошибок
- Внимательно прочитайте сообщения об ошибках. Они могут дать подсказки о том, что пошло не так.

2. Изоляция проблемы
- Попробуйте упростить код, чтобы изолировать проблему. Удалите ненужные части и протестируйте, работает ли программа.

6.2 Как найти ошибки?

1. Чтение кода
- Внимательно прочитайте свой код. Часто ошибки можно найти, просто просмотрев его.

2. Тестирование
- Пишите тесты для различных частей вашего кода. Это поможет выявить ошибки на ранних стадиях.

3. Сравнение с рабочими примерами
- Сравните свой код с примерами, которые работают. Это может помочь выявить различия и ошибки.

4. Обсуждение с коллегами
- Попросите кого-то другого взглянуть на ваш код. Свежий взгляд может помочь найти ошибки, которые вы могли пропустить.

5. Поиск в интернете
- Если вы столкнулись с конкретной ошибкой, попробуйте поискать её в интернете. Возможно, кто-то уже сталкивался с подобной проблемой и нашел решение.

Приложение 1. Обработка ошибок

Перечень ошибок, которые обрабатываются компилятором

1. **Ошибки со скобками:**
   - **WrongRoundBracket_open — ожидалась круглая открывающаяся скобка.**
   - **WrongRoundBracket_close — ожидалась круглая закрывающаяся скобка.**
   - **WrongCurlyBracket_open — ожидалась фигурная открывающаяся скобка.**
   - **WrongCurlyBracket_close — ожидалась фигурная закрывающаяся скобка.**
   - **WrongSquareBracket_open — ожидалась квадратная открывающаяся скобка.**
   - **WrongSquareBracket_close — ожидалась квадратная закрывающаяся скобка.**
2. **Синтаксические ошибки:**
   - **Wrong_id — ожидался идентификатор.**
   - **WrongSpecial_word_type — ожидался тип данных (integer, bool, line, valid, nochar).**
   - **WrongSpecial_word_type_bool — ожидались bool-значения (true или false).**
   - **WrongSpecial_word — ожидалось специальное слово.**
   - **WrongSpecial_word_len — ожидалось специальное слово len.**
   - **WrongSpecialWord_choice — ожидалось специальное слово choice.**
   - **Wrong_String — ожидалась строка.**
   - **WrongVariable — ошибка в работе с переменной.**
2. **Ошибки сравнения и арифметики:**
   - **WrongComprasion_sk1 — ожидался символ $<$.**
   - **WrongComprasion_sk2 — ожидался символ $>$.**
   - **WrongComprasion — ожидалась операция сравнения.**
   - **WrongEqual — ожидался символ $=$.**
   - **WrongExMark — ожидался символ !.**
   - **WrongEqually_NotEqually — ожидалось равенство или неравенство.**
   - **WrongPlus — ожидалась операция $+$.**
   - **WrongOperationObject — ожидался объект арифметики.**
   - **Wrong_arit — ожидалась арифметическая операция.**
   - **WrongNumber — ожидалось число.**
2. **Ошибки работы с массивами:**
   - **WrongArraySize — ожидалось число для объявления массива.**
   - **Wrong_Array_cell — некорректный тип для обращения к ячейке массива.**
   - **WrongArray — некорректное использование массива.**
   - **WrongArray_name — не найден массив.**
   - **WrongArray_arith — попытка выполнить неприменимую к массиву операцию.**
2. **Ошибки при работе с функциями:**
   - **WrongNo_return — функция должна возвращать значение.**
   - **WrongVoid — у void-функции не должно быть возвращаемого значения.**
   - **WrongReturn_array — попытка вернуть массив из функции.**

- **WrongCount_parametries — несоответствие количества формальных и фактических параметров в функции.**
- **WrongType_parametries — несоответствие типов формальных и фактических параметров.**

2. Ошибки типов:

- **Wrong_type — несоответствие типов данных.**
- **WrongChar_arith — некорректное использование типа nochar.**

2. Ошибки области видимости и инициализации:

- **Wrong_field_of_view — множественное объявление объекта.**
- **WrongInitialization — использование необъявленного идентификатора.**

2. Прочие ошибки:

- **WrongOverFlow — не обнаружен символ конца программы.**
- **Wrong_pop_add — ожидалась операция add или pop.**

Наши контакты

Почта: *valeriab7002@gmail.com*
Телефон: *+7 (777)-777-77-77*