```java
public class Parser {

   public static void main(String[] args) {
      ProgramText programText = new ProgramText();
      Scanner scanner = new Scanner(programText);
      Token token = new Token(programText);
      Parser parser = new Parser(scanner,programText,token);
      for(int i=0;i<programText.progText.length();i++){
         parser.parse();
      }


   }
   private  Scanner scanner;
   public  ProgramText programText;
   private  Token token;
   Parser(Scanner scanner, ProgramText programText,Token token){

      this.scanner = scanner;
      this.programText = programText;
      this.token=token;
   }
   //this parse method will be used only for the first assignment
     void parse() {

         token = scanner.nextToken();

         //while we do not reach the end of the file, we will keep asking
         //the scanner for the next token
        if(!(token instanceof EOFToken)){
           if(token!=null){
              System.out.printf("Type: %s, text: %s\n",
token.getTokenType(), token.getText());
           }
        }




            //x = 0;
            //sum = 0;
            //while(x < 10){
            // sum = sum + x;
            //     x = x + 1;
            //}

            //output:
            //text: x, type: identifier
            //text: =, type: specialSymbol
            //text: 0, type: Numbe
            //text: ;, type: specialSymbol
            //...





   }
   /*
    * the actual parse method will be like the following:
    * void parse(){
    *         Token token = scanner.nextToken();
    *         P(token);
    * }
    *
    */
   //For every non-terminal symbol in the grammar write a method.
   void P(Token token) {//S(token);}

   }
   void S(Token token) {//if(token.text.equals("while") ...
               //if(token.text.equals("if")...
               //else there is a syntax error here
      }

   void T() {}
   //...
}
```

```
public class Scanner {
   private ProgramText source;
   public String string="";

   Scanner(ProgramText source){
      this.source = source;
   }

   boolean isSpecial(char chNext) {
      boolean control = false;
      if(!Character.isWhitespace(chNext)){
         for (TokenType type : TokenType.values()) {
            if (String.valueOf(chNext).equals(type.getText())) {
               control = true;
               break;
            }
         }
      }
      return control;
   }
   //Scanner will ask the Source for characters and one a sequence of
   //characters form a token it will return immediately.
   //Scanner needs to know some of rules (for example, what constitutes
   //a number, what constitutes an identifier and so forth)
   Token nextToken() {
      Token token;

         char chCur = source.curChar();
         char chNext = source.nextChar();

         if(!Character.isWhitespace(chCur)){
         //System.out.println(chCur+" "+chNext);
            for(TokenType type : TokenType.values()){
               if(String.valueOf(chCur).equals(type.getText())){
                  token = new
SpecialToken(source,String.valueOf(chCur),type);
                  return token;
               }



               }
            if(Character.isDigit(chCur)) {
               //number token
               string+=chCur;
               if(isSpecial(chNext)){
```

```java
                        token = new NumberToken(source,string,TokenType.NUMBER);
                        string="";
                        return token;
                    }

                }
                else if(Character.isLetter(chCur)) {
                    //identifier token
                    string+=chCur;
                    if(isSpecial(chNext)){
                        if(string.equals(TokenType.WHILE.getText())){
                            //System.out.println(TokenType.WHILE.getText());
                            token = new
KeywordToken(source,string,TokenType.WHILE);
                            string="";
                            return token;
                        }
                        else{
                            token = new
IdentifierToken(source,string,TokenType.IDENITIFIER);
                            string="";
                            return token;
                        }


                    }

                }
                else if(false) { //we want to check here
                    //if ch variable contains a special characters (+, -, ..,
                    //{,},;, ...

                    //token = new SpecialToken(source);
                }
                }
                //we have the next non-white space character which is the
beginning
                //of a new token





            //...
            return null;


    }


}
```

```java
import java.io.IOException;
import java.nio.file.Files;
import java.nio.file.Paths;

//the purpose of the ProgramText class is to abstract away
//from where the program is coming. ProgramText provides a
//single character to the Scanner class when asked for.
//it reads the program (from a file or as String) line by line
//from top to bottom
```

```java
public class ProgramText {

   //private BufferedReader reader;
   public String progText;
   private int curPos,rez=0;
   public static char EOF = '₺';

   ProgramText(){

      curPos = -1;

      try {
         progText = readWholeProgram();

      } catch (IOException e) {
         // TODO Auto-generated catch block
         e.printStackTrace();
      }

   }
   private String readWholeProgram() throws IOException {
      return new String(Files.readAllBytes(Paths.get("program1.txt")));

   }
   char curChar() {
      if(curPos == -1)
         curPos++;

      return progText.charAt(curPos);
   }
   char nextChar() {
      curPos++;
      rez=curPos;
      if(rez == progText.length())
         return EOF;

      for(int i=rez;i<progText.length();i++){
         if(Character.isWhitespace(progText.charAt(rez))){
            rez++;
         }
      }

      if(rez == progText.length())
         return EOF;

      return progText.charAt(rez);
   }



}
```

```java
public class Token {
   //we get say "x" as an "identifier" (token's type)
   //we get say "foo" as an "identifier" (token's type)
   //x = y + 5;
   // C -> 'if' E 'then' S 'else' S ';'
   public TokenType type;
   public String text;
   private ProgramText source;
```

```java
    Token(ProgramText source){
        this.source = source;
    }
    public TokenType getTokenType() {
        return type;
    }
    public String getText() {
        return text;
    }
    //abstract public Token extract();
}
```

```java
public class IdentifierToken extends Token{

    IdentifierToken(ProgramText source,String text, TokenType type) {
        super(source);
        this.text=text;
        this.type=type;


    }




}
```

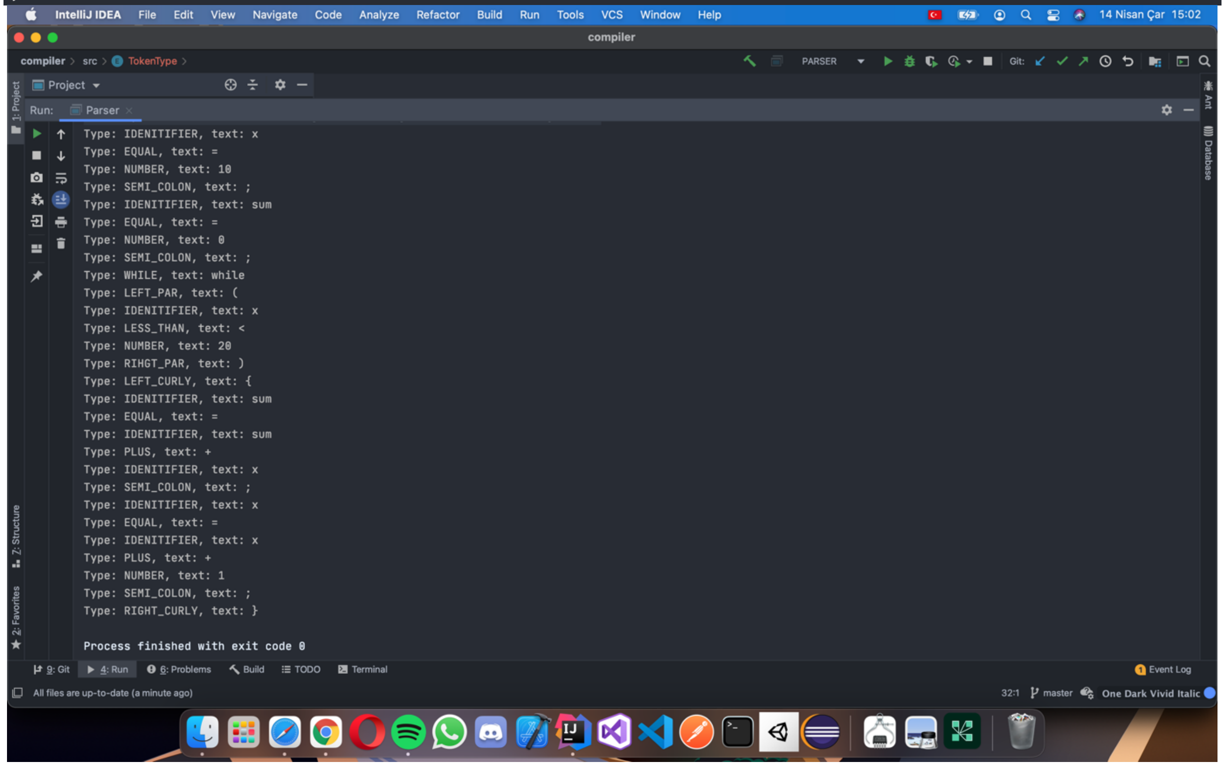
```java
public class KeywordToken extends Token {
     KeywordToken(ProgramText source,String text, TokenType type) {
         super(source);
         this.text=text;
         this.type=type;
     }
}
```

```java
public class NumberToken extends Token{

    NumberToken(ProgramText source,String text,TokenType type) {
        super(source);
        this.text=text;
        this.type=type;

    }
}
```

```java

public class SpecialToken extends Token{

    SpecialToken(ProgramText source,String text, TokenType Specialtype){
        super(source);
        this.text=text;
        this.type=Specialtype;
        // TODO Auto-generated constructor stub
    }

}
```

```java
public class EOFToken extends Token{

    EOFToken(ProgramText source) {
        super(source);
    }

}
```

```java
public enum TokenType {
    LEFT_CURLY("{"), RIGHT_CURLY("}"), LEFT_PAR("("), RIHGT_PAR(")"),
    EQUAL("="), PLUS("+"),  SEMI_COLON(";"), LESS_THAN("<"),

    WHILE("while"),
    IDENITIFIER, NUMBER;

    public String getText() {
        return text;
    }

    private String text;
    TokenType(String text) {
        this.text = text;
    }
    TokenType(){
        this.text = this.toString();
    }
}
```



```
Type: IDENITIFIER, text: x
Type: EQUAL, text: =
Type: NUMBER, text: 10
Type: SEMI_COLON, text: ;
Type: IDENITIFIER, text: sum
Type: EQUAL, text: =
Type: NUMBER, text: 0
Type: SEMI_COLON, text: ;
Type: WHILE, text: while
Type: LEFT_PAR, text: (
Type: IDENITIFIER, text: x
Type: LESS_THAN, text: <
Type: NUMBER, text: 20
Type: RIHGT_PAR, text: )
Type: LEFT_CURLY, text: {
Type: IDENITIFIER, text: sum
Type: EQUAL, text: =
Type: IDENITIFIER, text: sum
Type: PLUS, text: +
Type: IDENITIFIER, text: x
Type: SEMI_COLON, text: ;
Type: IDENITIFIER, text: x
Type: EQUAL, text: =
Type: IDENITIFIER, text: x
Type: PLUS, text: +
Type: NUMBER, text: 1
Type: SEMI_COLON, text: ;
Type: RIGHT_CURLY, text: }

Process finished with exit code 0
```

IntelliJ IDEA File Edit View Navigate Code Analyze Refactor Build Run Tools VCS Window Help
14 Nisan Çar 15:02
compiler
compiler > src > TokenType
PARSER
Project
Run: Parser
9: Git
4: Run
6: Problems
Build
TODO
Terminal
Event Log
All files are up-to-date (a minute ago)
32:1 master One Dark Vivid Italic