Republic of Yemen
IBB University
Faculty of Science
Departments of
IT & CS
Compilers

الجمهورية اليمنية
جامعة إب
كلية العلوم التطبيقية
قسم : علوم الحاسوب
وتقنية المعلومات

مشروع مترجمات

مترجمات

د / خالد الكحسه

الطلاب /

أيمن محمد ناجي قمحان

حازم هزام جمال العمري

ضياء فضل الحضرمي

طارق فضل علي محمد العمري

علي محمد أحمد القواس

2025

تقرير فني شامل

مشروع المترجم العربي (Arabic Compiler)

المقدمة

يقدم هذا التقرير تحليلاً فنياً شاملاً لمشروع المترجم العربي (Arabic Compiler)، وهو مشروع طموح يهدف إلى توفير لغة برمجة تستخدم الكلمات المفتاحية والقواعد النحوية العربية .يركز التقرير على تحليل البنية المعمارية للمترجم، الخوارزميات المستخدمة في كل مرحلة، الأدوات والتقنيات المعتمدة، بالإضافة إلى أمثلة عملية لتنفيذ الكود.

1.تحليل البنية المعمارية والأدوات المستخدمة

يعتمد المشروع على بنية معمارية هجينة، حيث تم تقسيم المكونات الرئيسية إلى جزأين :المترجم الأساسي (Compiler)وبيئة التطوير المتكاملة .(IDE)

1.1. المترجم الأساسي (Compiler)

الشرح	القيمة	الخاصية
تم اختيار C++لسرعتها وكفاءتها في التعامل مع عمليات التحليل النحوي واللغوي المعقدة.	C++معيار (C++17)	لغة البرمجة
تستخدم لإدارة عملية بناء المشروع وتوليد ملفات البناء (Makefiles)على منصات مختلفة.	CMake	أداة البناء
تم التعامل مع الترميز العربي بشكل خاص في مرحلة التحليل اللغوي لضمان قراءة وفهم الكلمات المفتاحية العربية.	UTF-8/Windows-1256	الترميز
يتيح المترجم توليد عدة أشكال من الكود، مما يجعله مرناً وقابلاً للتطبيق على منصات مختلفة.	كود وسيط (Intermediate Code)، كود C، كود تجميعي (MIPS Assembly)	الناتج المستهدف

1.2.بيئة التطوير المتكاملة (IDE)

الشرح	القيمة	الخاصية
تم استخدام C#لتطوير واجهة المستخدم الرسومية (GUI)لبيئة التطوير المتكاملة.	C#	لغة البرمجة
يوفر الإطار الأساس لتشغيل واجهة المستخدم على أنظمة التشغيل التي تدعم .NET.	NET.إصدار (6.0)	إطار العمل
	واجهة مستخدم رسومية لتسهيل كتابة وتشغيل الكود العربي.	الهدف

2.مراحل عملية الترجمة والخوارزميات

يتبع المترجم البنية التقليدية للمترجمات، حيث يمر الكود المصدري بثلاث مراحل رئيسية:

2.1.المرحلة الأولى :التحليل اللغوي (Lexical Analysis)

الملف المسؤولArabicCompiler/Compiler/src/Lexer.cpp :

الخوارزمية: تعتمد هذه المرحلة على خوارزمية الآلة ذات الحالة المحدودة

(Finite State Machine - FSM).

1 قراءة الحرف: يتم قراءة الكود المصدري حرفاً بحرف باستخدام الدالة .advance()
2 تخطي المسافات والتعليقات: يتم تجاهل المسافات البيضاء والتعليقات)باستخدام skipWhitespace() و .(skipComment()
3 تحديد الرمز :(Tokenization) يتم استخدام دوال متخصصة لتحديد نوع الرمز:
   - :readIdentifier()لتحديد الكلمات المفتاحية العربية)برنامج، متغير، اذا (أو أسماء المتغيرات .يتم تخزين الكلمات المفتاحية في جدول (keywords)للمقارنة السريعة.
   - :readNumber()لقراءة الأعداد الصحيحة والحقيقية)مع دعم النقطة العشرية.(
   - :readString()لقراءة السلاسل النصية)مع دعم أحرف الهروب .(\n, \t, \''
4 التعامل مع الترميز: تم تضمين منطق خاص للتعامل مع ترميز UTF-8/Windows-1256لضمان قراءة الأحرف العربية بشكل صحيح.

2.2.المرحلة الثانية :التحليل النحوي (Syntax Analysis)

الملف المسؤولArabicCompiler/Compiler/src/Parser.cpp :

الخوارزمية: تعتمد هذه المرحلة على خوارزمية التحليل النحوي التنازلي (Top-Down Parsing)، وتحديداً تقنية التحليل النحوي التراجعي التنبئي .(Recursive Descent Parser)

5 بناء شجرة الاشتقاق النحوي :(AST) يتم بناء شجرة الاشتقاق النحوي - (Abstract Syntax Tree AST)لتمثيل البنية الهيكلية للكود.

6 التعابير الرياضية والمنطقية: يتم استخدام خوارزمية ترتيب العمليات (Operator Precedence) لتحليل التعابير المعقدة)مثل parseExpression()، parseTerm()، .(parseFactor()

7 تحليل الجمل: يتم تحليل جمل التحكم مثل parseIfStatement(), parseWhileStatement(), parseForStatement()و parseAssignment()، حيث يتم استهلاك الرموز المتوقعة)باستخدام consume()أو محاولة مطابقتها)باستخدام .match()

2.3.المرحلة الثالثة :توليد الكود (Code Generation)

الملف المسؤولArabicCompiler/Compiler/src/Compiler.cpp :

الخوارزمية: تستخدم هذه المرحلة خوارزمية المرور الواحد على شجرة

(Single-Pass AST Traversal) AST لتوليد الكود الوسيط.

8 جدول الرموز :(Symbol Table) يتم استخدام جدول الرموز لتتبع المتغيرات والثوابت وأنواعها أثناء عملية الترجمة)الدالة .(compileVariableDeclaration()

9 الكود الوسيط :(Intermediate Code) يتم توليد كود وسيط ثلاثي العناوين (Three-Address Code - TAC)يمثل العمليات الأساسية)مثل .(ADD, MUL, STORE, JUMP

10 توليد التعليمات: يتم استخدام دوال مثل compileExpression() و compileStatement() لزيارة عقد ASTالمختلفة وتوليد تعليمات TACالمقابلة.

11 توليد المخرجات: يتم تحويل تعليمات TACإلى كود Cأو كود تجميعي) MIPSباستخدام generateCCode() و .(generateAssembly()

3.أمثلة التنفيذ والشاشات

تم تنفيذ المترجم بنجاح على مثال بسيط لبرنامج يقوم بحساب قيمة رياضية.

3.1.الكود المصدري العربي (example1.arabic)

```
// example1.arabic
```

```
أساسي؛ برنامج

صحيح؛ : س متغير
صحيح؛ : ن متغير

؛(" :رقمًا أدخل")اطبع
؛(س)اقرأ

؛5 + 2 * س = ن
؛(" :الناتج")اطبع
```

```
؛(ن)اطبع
```

3.2.شاشة التنفيذ -مرحلة الترجمة

يوضح الإخراج التالي عملية التحليل والترجمة التي قام بها المترجم:

```
📂 الملف تحليل جاري: ArabicCompiler/Examples/example1.arabic
```

```
📏 النص طول: 214 حرف
✓ بنجاح تم اللغوي التحليل! (رمز 42)
... (النحوي التحليل تفاصيل)
✓ بنجاح تم النحوي التحليل!
🔧 المخرجات توليد جاري...
✓ في الوسيط الكود توليد تم:
ArabicCompiler/Examples/example1_output_intermediate.txt
📄 كود توليد تم في C: ArabicCompiler/Examples/example1_output.c
                    📄 تم توليد كود التجميع في:
            ArabicCompiler/Examples/example1_output.asm
```

```
🔧 المولد الوسيط الكود:
====================
0: PRINT STRING, str_0
1: READ س
2: LOAD t0, س
3: LOAD t1, 2
4: MUL t2, t0, t1
5: LOAD t3, 5
6: ADD t4, t2, t3
7: STORE ن, t4
8: PRINT STRING, str_1
9: PRINT VARIABLE, ن
10: HALT
...
```

```
🎉 بنجاح تمت الترجمة!
```

3.3. الكود الوسيط الناتج (Intermediate Code)

```
المولد الوسيط الكود:
```

```
====================
0: PRINT STRING, str_0
1: READ س
2: LOAD t0, س
3: LOAD t1, 2
4: MUL t2, t0, t1
5: LOAD t3, 5
6: ADD t4, t2, t3
7: STORE ن, t4
8: PRINT STRING, str_1
9: PRINT VARIABLE, ن
```

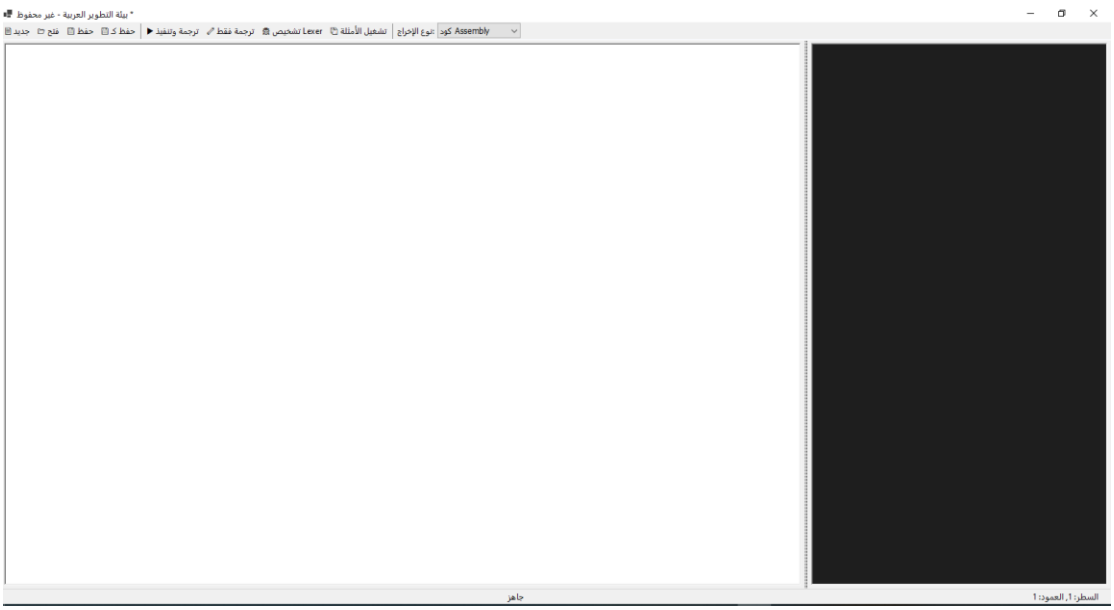
```
10: HALT
```

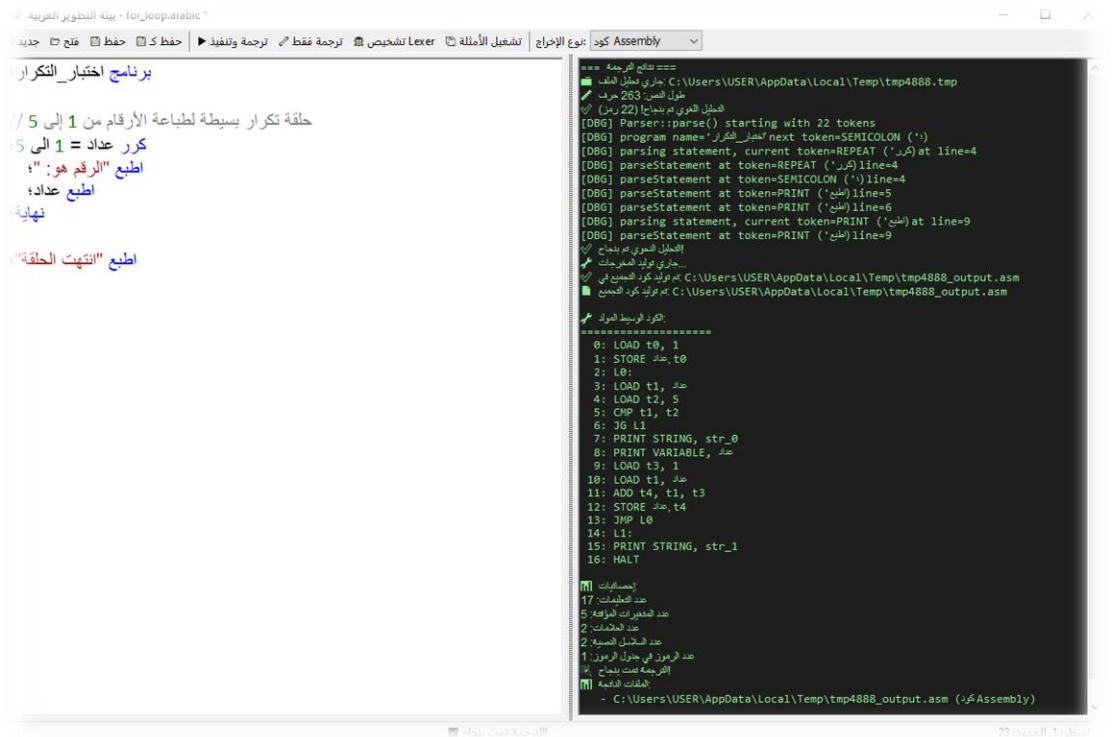
3.4. كود C الناتج (example1_output.c)

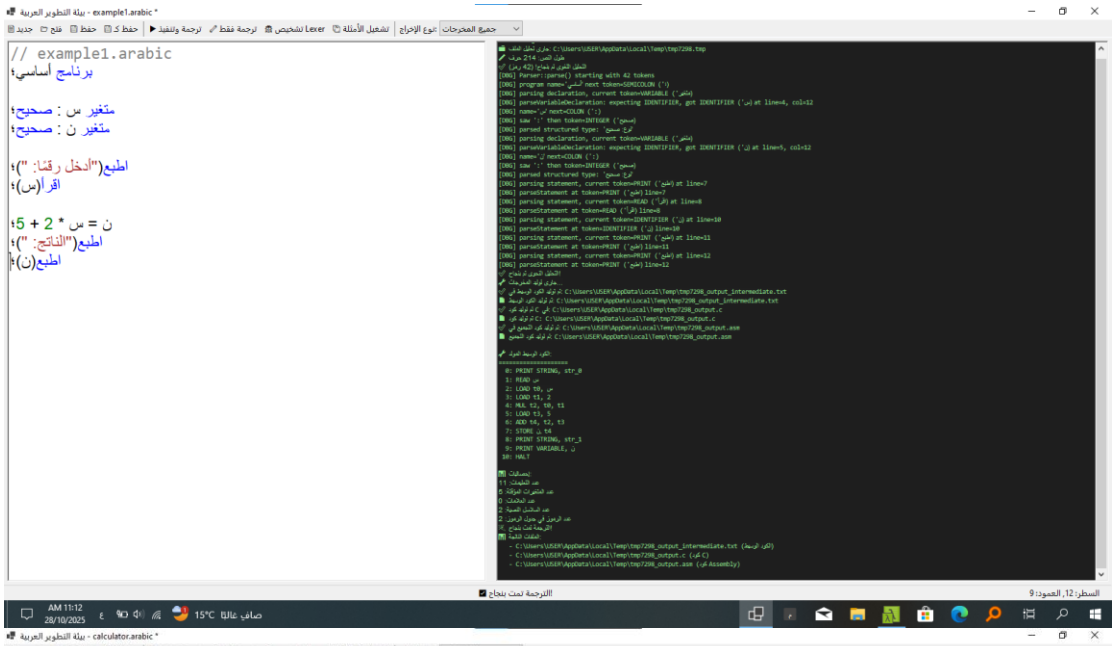
```
// كود C مولد للغة العربية
```

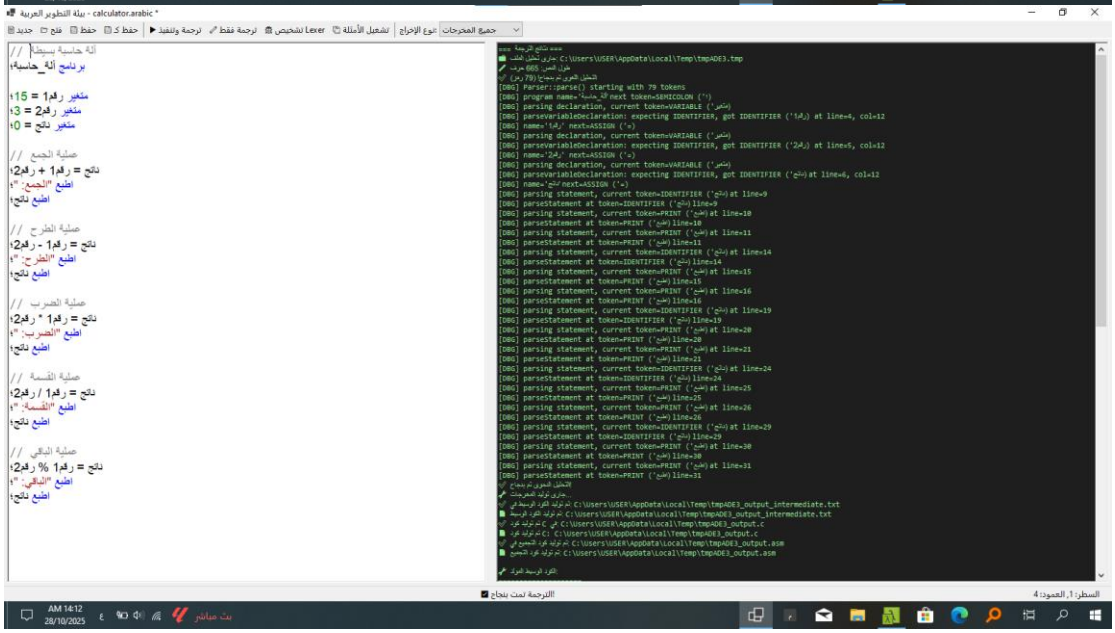
```
// ========================

#include <stdio.h>
#include <stdlib.h>

int main() {
int ن = 0;
int س = 0;
int t0 = 0;
// ... (المؤقتة المتغيرات بقية)
char* str_0 = "رقمًا أدخل: ";
char* str_1 = "الناتج: ";

printf("%s\n", str_0);
scanf("%d", &س);
t0 = س;
t1 = 2;
t2 = t0 * t1;
t3 = 5;
t4 = t2 + t3;
ن = t4;
printf("%s\n", str_1);
printf("%d\n", ن);
return 0;
```

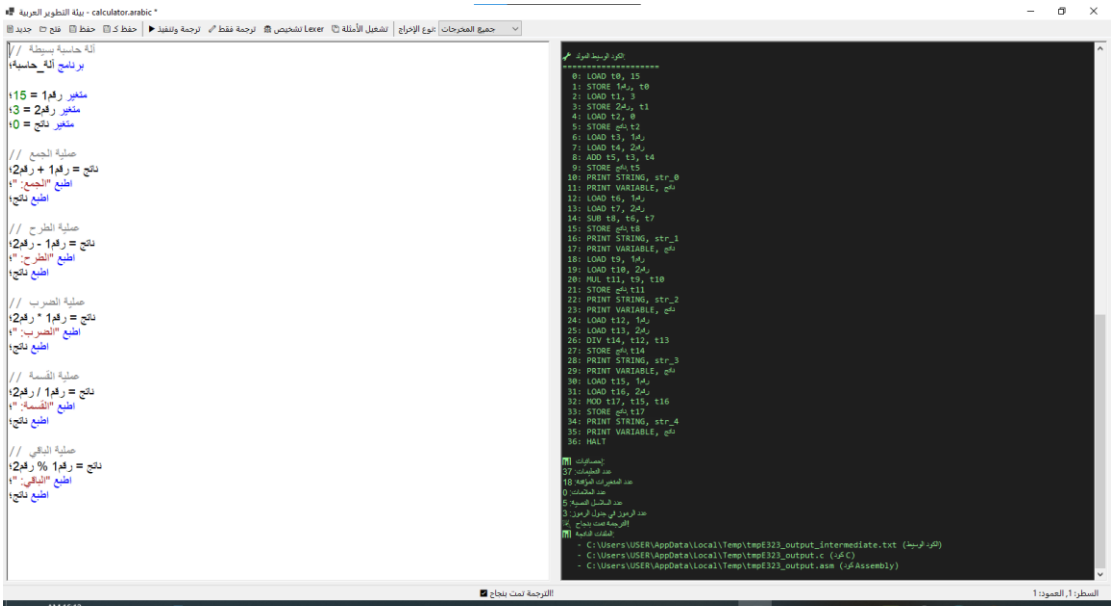
```
}
```

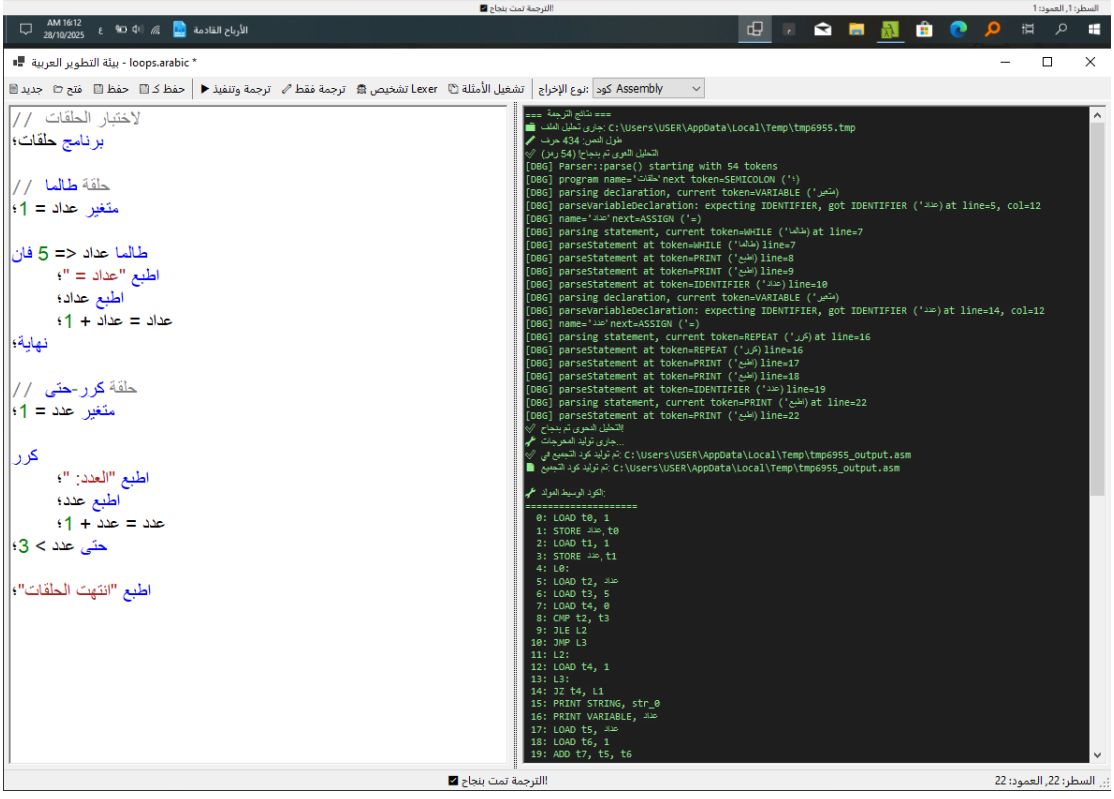

بيئة التطوير العربية - غير محفوظ *
جديد
فتح
حفظ
حفظ ك
ترجمة وتنفيذ
ترجمة فقط
تشخيص Lexer
تشغيل الأمثلة
نوع الإخراج:
كود Assembly
جاهز
السطر: 1, العمود: 1

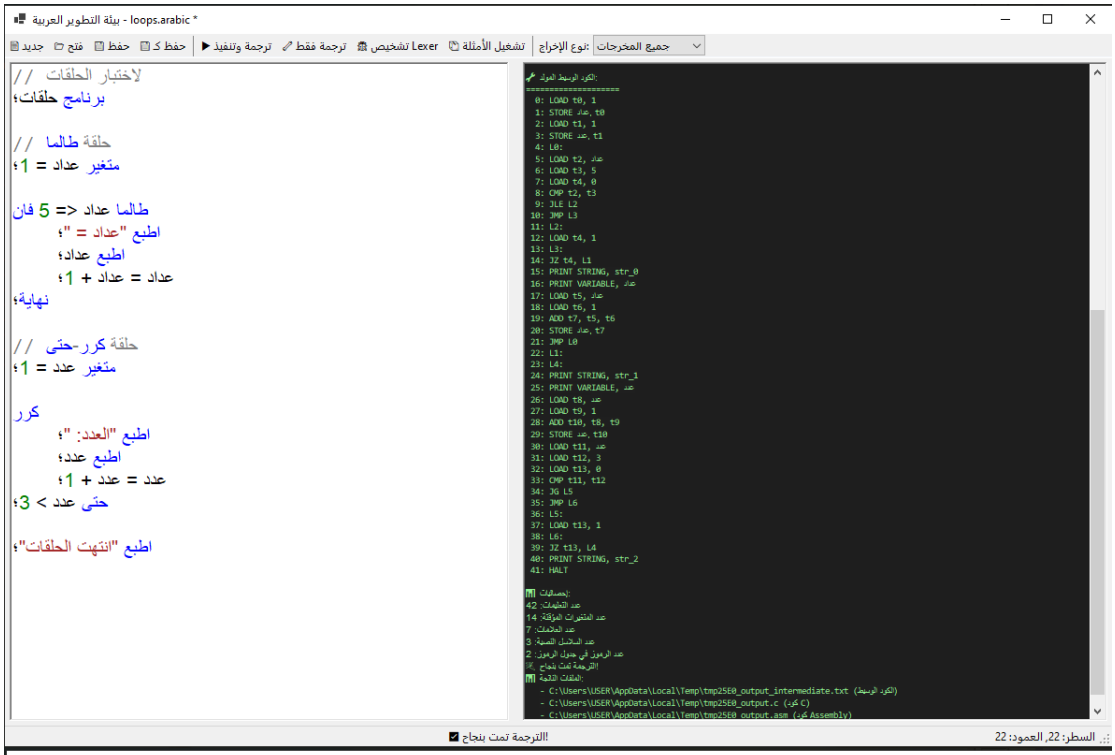

بيئة التطوير العربية - for_loop.arabic *
جديد
فتح
حفظ
حفظ ك
ترجمة وتنفيذ
ترجمة فقط
تشخيص Lexer
تشغيل الأمثلة
نوع الإخراج:
كود Assembly
برنامج اختبار_التكرار
حلقة تكرار بسيطة لطباعة الأرقام من 1 إلى 5 /
كرر عداد = 1 الى 5
اطبع "الرقم هو: "؛
اطبع عداد؛
نهاية
اطبع "انتهت الحلقة"
=== تكامُع الترجمة ===
جاري تحليل الملف: C:\Users\USER\AppData\Local\Temp\tmp4888.tmp
طول النص: 263 حرف
التحليل اللغوي تم بنجاح! (22 رمز)
[DBG] Parser::parse() starting with 22 tokens
[DBG] program name='اختبار_التكرار' next token=SEMICOLON ('؛')
[DBG] parsing statement, current token=REPEAT ('كرر') at line=4
[DBG] parseStatement at token=REPEAT ('كرر') line=4
[DBG] parseStatement at token=SEMICOLON ('؛') line=4
[DBG] parseStatement at token=PRINT ('اطبع') line=5
[DBG] parseStatement at token=PRINT ('اطبع') line=6
[DBG] parsing statement, current token=PRINT ('اطبع') at line=9
[DBG] parseStatement at token=PRINT ('اطبع') line=9
التحليل النحوي تم بنجاح
جاري توليد المخرجات...
تم توليد كود التجميع في: C:\Users\USER\AppData\Local\Temp\tmp4888_output.asm
تم توليد كود التجميع: C:\Users\USER\AppData\Local\Temp\tmp4888_output.asm
الكود الوسيط المولد:
==========
0: LOAD t0, 1
1: STORE عداد, t0
2: L0:
3: LOAD t1, عداد
4: LOAD t2, 5
5: CMP t1, t2
6: JG L1
7: PRINT STRING, str_0
8: PRINT VARIABLE, عداد
9: LOAD t3, 1
10: LOAD t1, عداد
11: ADD t4, t1, t3
12: STORE عداد, t4
13: JMP L0
14: L1:
15: PRINT STRING, str_1
16: HALT
إحصائيات:
عدد التعليمات: 17
عدد المتغيرات المؤقتة: 5
عدد العلامات: 2
عدد السلاسل النصية: 2
عدد الرموز في جدول الرموز: 1
الترجمة تمت بنجاح
الملفات الناتجة:
- C:\Users\USER\AppData\Local\Temp\tmp4888_output.asm (كود Assembly)

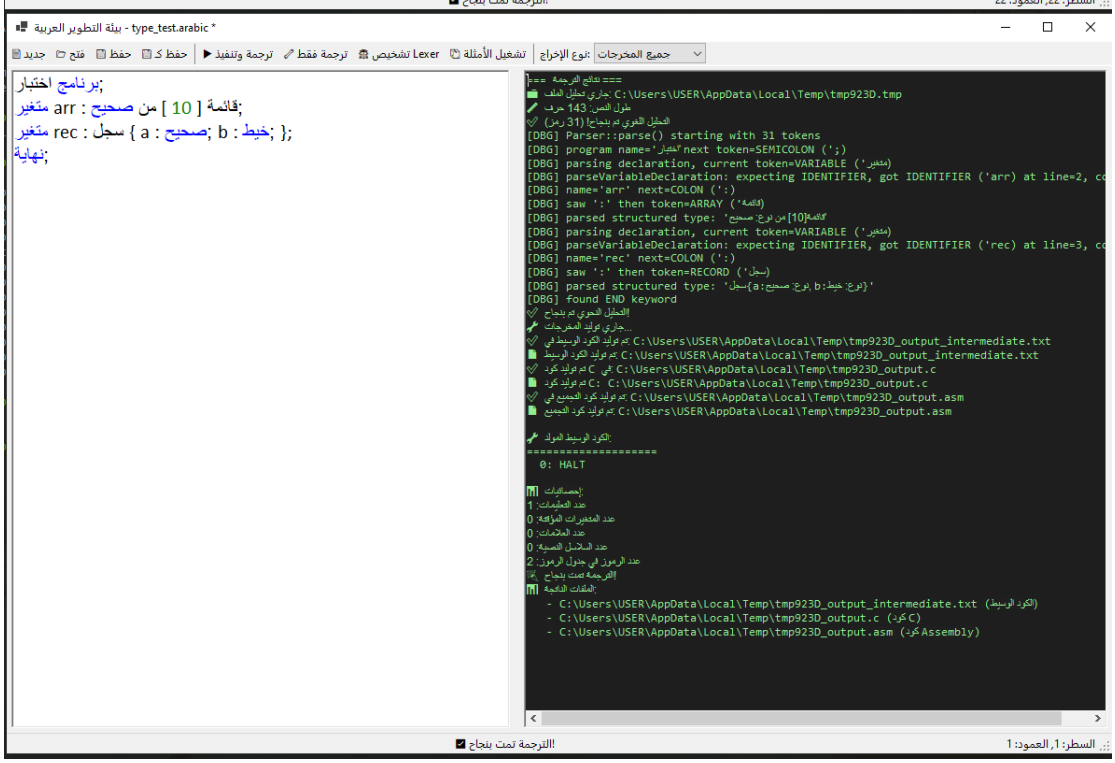

بيئة التطوير العربية - example1.arabic *
جديد
فتح
حفظ
حفظ ك
تشخيص Lexer
ترجمة فقط
ترجمة وتنفيذ
تشغيل الأمثلة
نوع الإخراج
جميع المخرجات
// example1.arabic
برنامج أساسي؛
متغير س : صحيح؛
متغير ن : صحيح؛
اطبع("أدخل رقمًا: ")؛
اقرأ(س)؛
ن = س * 2 + 5؛
اطبع("الناتج: ")؛
اطبع(ن)؛
الترجمة تمت بنجاح
السطر: 12، العمود: 9
AM 11:12
28/10/2025
15°C صافي غالبًا

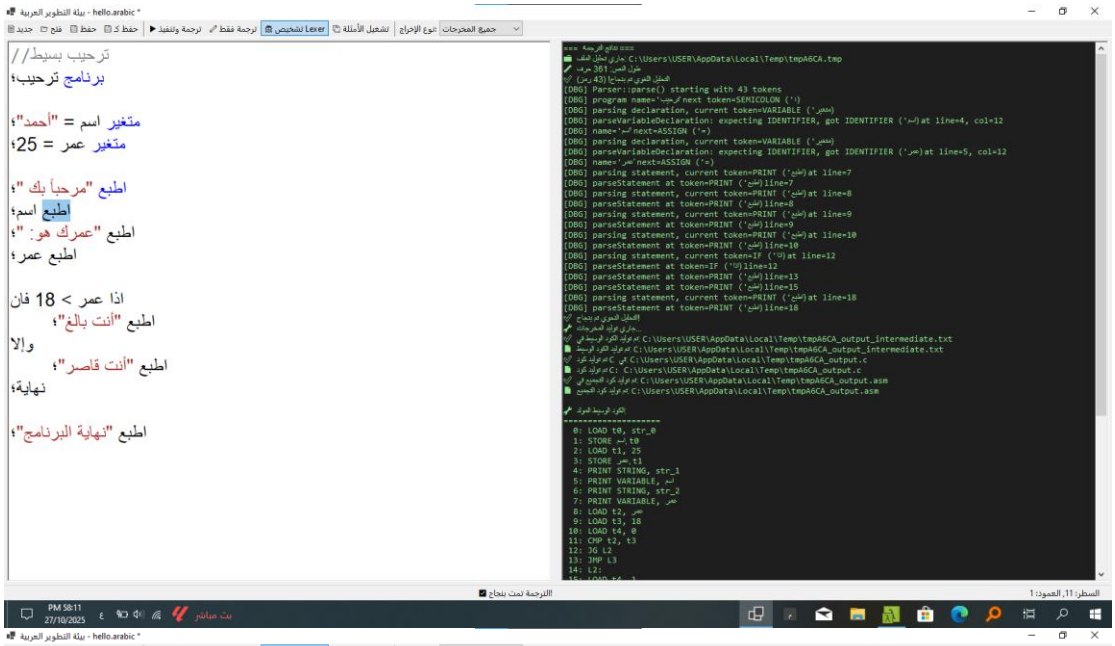

بيئة التطوير العربية - calculator.arabic *
جديد
فتح
حفظ
حفظ ك
تشخيص Lexer
ترجمة فقط
ترجمة وتنفيذ
تشغيل الأمثلة
نوع الإخراج
جميع المخرجات
// آلة حاسبة بسيطة
برنامج آلة_حاسبة؛
متغير رقم1 = 15؛
متغير رقم2 = 3؛
متغير ناتج = 0؛
// عملية الجمع
ناتج = رقم1 + رقم2؛
اطبع "الجمع: "؛
اطبع ناتج؛
// عملية الطرح
ناتج = رقم1 - رقم2؛
اطبع "الطرح: "؛
اطبع ناتج؛
// عملية الضرب
ناتج = رقم1 * رقم2؛
اطبع "الضرب: "؛
اطبع ناتج؛
// عملية القسمة
ناتج = رقم1 / رقم2؛
اطبع "القسمة: "؛
اطبع ناتج؛
// عملية الباقي
ناتج = رقم1 % رقم2؛
اطبع "الباقي: "؛
اطبع ناتج؛
الترجمة تمت بنجاح
السطر: 1، العمود: 4
AM 14:12
28/10/2025
بث مباشر

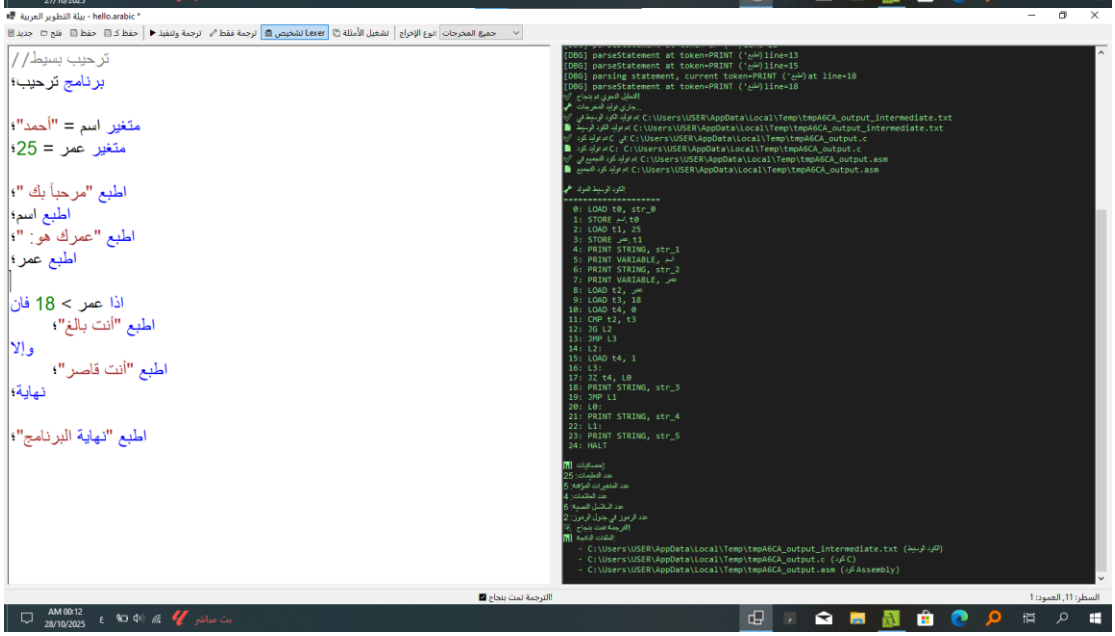

بيئة التطوير العربية - calculator.arabic *
// آلة حاسبة بسيطة
برنامج آلة_حاسبة؛
متغير رقم1 = 15؛
متغير رقم2 = 3؛
متغير ناتج = 0؛
// عملية الجمع
ناتج = رقم1 + رقم2؛
اطبع "الجمع: "؛
اطبع ناتج؛
// عملية الطرح
ناتج = رقم1 - رقم2؛
اطبع "الطرح: "؛
اطبع ناتج؛
// عملية الضرب
ناتج = رقم1 * رقم2؛
اطبع "الضرب: "؛
اطبع ناتج؛
// عملية القسمة
ناتج = رقم1 / رقم2؛
اطبع "القسمة: "؛
اطبع ناتج؛
// عملية الباقي
ناتج = رقم1 % رقم2؛
اطبع "الباقي: "؛
اطبع ناتج؛
الترجمة تمت بنجاح!
السطر: 1, العمود: 1


بيئة التطوير العربية - loops.arabic *
// لاختبار الحلقات
برنامج حلقات؛
// حلقة طالما
متغير عداد = 1؛
طالما عداد <= 5 فان
اطبع "عداد = "؛
اطبع عداد؛
عداد = عداد + 1؛
نهاية؛
// حلقة كرر-حتى
متغير عدد = 1؛
كرر
اطبع "العدد: "؛
اطبع عدد؛
عدد = عدد + 1؛
حتى عدد > 3؛
اطبع "انتهت الحلقات"؛
الترجمة تمت بنجاح!
السطر: 22, العمود: 22


بيئة التطوير العربية - loops.arabic *
جديد
فتح
حفظ
حفظ ك
ترجمة وتنفيذ
ترجمة فقط
تشخيص Lexer
تشغيل الأمثلة
نوع الإخراج:
جميع المخرجات
// لاختبار الحلقات
برنامج حلقات؛
// حلقة طالما
متغير عداد = 1؛
طالما عداد <= 5 فان
اطبع "عداد = "؛
اطبع عداد؛
عداد = عداد + 1؛
نهاية؛
// حلقة كرر-حتى
متغير عدد = 1؛
كرر
اطبع "العدد: "؛
اطبع عدد؛
عدد = عدد + 1؛
حتى عدد > 3؛
اطبع "انتهت الحلقات"؛
الكود الوسيط المولد:
0: LOAD t0, 1
1: STORE عداد, t0
2: LOAD t1, 1
3: STORE عدد, t1
4: L0:
5: LOAD t2, عداد
6: LOAD t3, 5
7: LOAD t4, 0
8: CMP t2, t3
9: JLE L2
10: JMP L3
11: L2:
12: LOAD t4, 1
13: L3:
14: JZ t4, L1
15: PRINT STRING, str_0
16: PRINT VARIABLE, عداد
17: LOAD t5, عداد
18: LOAD t6, 1
19: ADD t7, t5, t6
20: STORE عداد, t7
21: JMP L0
22: L1:
23: L4:
24: PRINT STRING, str_1
25: PRINT VARIABLE, عدد
26: LOAD t8, عدد
27: LOAD t9, 1
28: ADD t10, t8, t9
29: STORE عدد, t10
30: LOAD t11, عدد
31: LOAD t12, 3
32: LOAD t13, 0
33: CMP t11, t12
34: JG L5
35: JMP L6
36: L5:
37: LOAD t13, 1
38: L6:
39: JZ t13, L4
40: PRINT STRING, str_2
41: HALT
إحصائيات:
عدد التعليمات: 42
عدد المتغيرات المؤقتة: 14
عدد العلامات: 7
عدد السلاسل النصية: 3
عدد الرموز في جدول الرموز: 2
الترجمة تمت بنجاح!
الملفات الناتجة:
- C:\Users\USER\AppData\Local\Temp\tmp25E0_output_intermediate.txt (الكود الوسيط)
- C:\Users\USER\AppData\Local\Temp\tmp25E0_output.c (كود C)
- C:\Users\USER\AppData\Local\Temp\tmp25E0_output.asm (كود Assembly)
الترجمة تمت بنجاح!
السطر: 22, العمود: 22


بيئة التطوير العربية - type_test.arabic *
جديد
فتح
حفظ
حفظ ك
ترجمة وتنفيذ
ترجمة فقط
تشخيص Lexer
تشغيل الأمثلة
نوع الإخراج:
جميع المخرجات
برنامج اختبار;
متغير arr : صحيح من [10] قائمة;
متغير rec : سجل { a : صحيح; b : خيط; };
نهاية;
=== تفاصيل الترجمة ===
جاري تحليل الملف: C:\Users\USER\AppData\Local\Temp\tmp923D.tmp
طول النص: 143 حرف
التحليل اللغوي تم بنجاح! (31 رمز)
[DBG] Parser::parse() starting with 31 tokens
[DBG] program name='اختبار' next token=SEMICOLON (';')
[DBG] parsing declaration, current token=VARIABLE ('متغير')
[DBG] parseVariableDeclaration: expecting IDENTIFIER, got IDENTIFIER ('arr') at line=2, co
[DBG] name='arr' next=COLON (':')
[DBG] saw ':' then token=ARRAY ('قائمة')
[DBG] parsed structured type: 'قائمة[10] من نوع:صحيح'
[DBG] parsing declaration, current token=VARIABLE ('متغير')
[DBG] parseVariableDeclaration: expecting IDENTIFIER, got IDENTIFIER ('rec') at line=3, co
[DBG] name='rec' next=COLON (':')
[DBG] saw ':' then token=RECORD ('سجل')
[DBG] parsed structured type: 'سجل{a:صحيح, نوع: b:خيط, نوع:خيط}'
[DBG] found END keyword
التحليل النحوي تم بنجاح
جاري توليد المخرجات...
تم توليد الكود الوسيط في: C:\Users\USER\AppData\Local\Temp\tmp923D_output_intermediate.txt
تم توليد الكود الوسيط: C:\Users\USER\AppData\Local\Temp\tmp923D_output_intermediate.txt
تم توليد كود C في: C:\Users\USER\AppData\Local\Temp\tmp923D_output.c
تم توليد كود C: C:\Users\USER\AppData\Local\Temp\tmp923D_output.c
تم توليد كود التجميع في: C:\Users\USER\AppData\Local\Temp\tmp923D_output.asm
تم توليد كود التجميع: C:\Users\USER\AppData\Local\Temp\tmp923D_output.asm
الكود الوسيط المولد:
0: HALT
إحصائيات:
عدد التعليمات: 1
عدد المتغيرات المؤقتة: 0
عدد العلامات: 0
عدد السلاسل النصية: 0
عدد الرموز في جدول الرموز: 2
الترجمة تمت بنجاح!
الملفات الناتجة:
- C:\Users\USER\AppData\Local\Temp\tmp923D_output_intermediate.txt (الكود الوسيط)
- C:\Users\USER\AppData\Local\Temp\tmp923D_output.c (كود C)
- C:\Users\USER\AppData\Local\Temp\tmp923D_output.asm (كود Assembly)
الترجمة تمت بنجاح!
السطر: 1, العمود: 1


بيئة التطوير العربية - hello.arabic *
جديد
فتح
حفظ
حفظ ك
ترجمة وتنفيذ
ترجمة فقط
تشخيص Lexer
تشغيل الأمثلة
نوع الإخراج
جميع المخرجات
// ترحيب بسيط
برنامج ترحيب؛
متغير اسم = "أحمد"؛
متغير عمر = 25؛
اطبع "مرحباً بك "؛
اطبع اسم؛
اطبع "عمرك هو: "؛
اطبع عمر؛
اذا عمر > 18 فان
اطبع "أنت بالغ"؛
وإلا
اطبع "أنت قاصر"؛
نهاية؛
اطبع "نهاية البرنامج"؛
=== نتائج الترجمة ===
[DBG] Parser::parse() starting with 43 tokens
[DBG] parsing declaration, current token=VARIABLE ('متغير')
[DBG] parsing statement, current token=PRINT ('اطبع')at line=7
[DBG] parsing statement, current token=PRINT ('اطبع')at line=8
[DBG] parsing statement, current token=PRINT ('اطبع')at line=9
[DBG] parsing statement, current token=PRINT ('اطبع')at line=10
[DBG] parsing statement, current token=IF ('اذا')at line=12
[DBG] parsing statement, current token=PRINT ('اطبع')at line=13
[DBG] parsing statement, current token=PRINT ('اطبع')at line=15
[DBG] parsing statement, current token=PRINT ('اطبع')at line=18
[DBG] parseStatement at token=PRINT ('اطبع')line=18
; LOAD t0, str_0
; STORE اسم, t0
; LOAD t1, 25
; STORE عمر, t1
; PRINT STRING, str_1
; PRINT VARIABLE, اسم
; PRINT STRING, str_2
; PRINT VARIABLE, عمر
; LOAD t2, عمر
; LOAD t3, 18
; LOAD t4, 0
; CMP t2, t3
; JG L2
; JMP L3
; L2:
السطر: 11, العمود: 1
PM 58:11
27/10/2025
بث مباشر


بيئة التطوير العربية - hello.arabic *
// ترحيب بسيط
برنامج ترحيب؛
متغير اسم = "أحمد"؛
متغير عمر = 25؛
اطبع "مرحباً بك "؛
اطبع اسم؛
اطبع "عمرك هو: "؛
اطبع عمر؛
اذا عمر > 18 فان
اطبع "أنت بالغ"؛
وإلا
اطبع "أنت قاصر"؛
نهاية؛
اطبع "نهاية البرنامج"؛
[DBG] parseStatement at token=PRINT ('اطبع')line=13
[DBG] parseStatement at token=PRINT ('اطبع')line=15
[DBG] parsing statement, current token=PRINT ('اطبع')at line=18
[DBG] parseStatement at token=PRINT ('اطبع')line=18
; LOAD t0, str_0
; STORE اسم, t0
; LOAD t1, 25
; STORE عمر, t1
; PRINT STRING, str_1
; PRINT VARIABLE, اسم
; PRINT STRING, str_2
; PRINT VARIABLE, عمر
; LOAD t2, عمر
; LOAD t3, 18
; LOAD t4, 0
; CMP t2, t3
; JG L2
; JMP L3
; L2:
; LOAD t4, 1
; L3:
; JZ t4, L0
; PRINT STRING, str_3
; JMP L1
; L0:
; PRINT STRING, str_4
; L1:
; PRINT STRING, str_5
; HALT
- C:\Users\USER\AppData\Local\Temp\tmpA6CA_output_intermediate.txt
- C:\Users\USER\AppData\Local\Temp\tmpA6CA_output.c
- C:\Users\USER\AppData\Local\Temp\tmpA6CA_output.asm
السطر: 11, العمود: 1
AM 00:12
28/10/2025
بث مباشر

تفصل لوظائف الدوال في مشروع المترجم العربي

(Arabic Compiler)

تحليلاً مفصلاً لوظيفة كل دالة رئيسية ومساعدة ضمن المكونات الأساسية للمترجم
(Lexer, Parser, Compiler)، مع التركيز على دورها في مراحل عملية الترجمة.

1.الدوال في مرحلة التحليل اللغوي (Lexer)

توجد هذه الدوال في ملف ArabicCompiler/Compiler/src/Lexer.cpp، وهي مسؤولة عن تحويل الكود المصدري إلى رموز مميزة .(Tokens)

الوصف الوظيفي	الدالة
البناء: تهيئة المحلل اللغوي بتخزين الكود المصدري، وتعيين المؤشرات (position, line, column)إلى البداية، وتحميل قائمة الكلمات المفتاحية العربية في جدول البحث .(keywords)	Lexer::Lexer(const std::string &source, bool debugFlag)
تحليل الترميز: دالة مساعدة تستخدم لأغراض التصحيح .(Debugging)تقوم بتحليل أول 50بايت من الكود المصدري لتقدير عدد الأحرف العربية المحتملة وطباعة قيمها الـ Hexللمساعدة في تحديد مشاكل الترميز.	Lexer::analyzeEncoding()
تصحيح الحرف: دالة مساعدة لطباعة معلومات مفصلة عن الحرف الحالي (ASCII, Hex, Position)عند تفعيل وضع التصحيح.	Lexer::debugChar(char c)
نظرة مسبقة: تعيد الحرف الحالي في موقع المؤشر دون تحريكه .تستخدم لاتخاذ قرارات بشأن الرمز التالي دون استهلاكه.	Lexer::peek()
التقدم: تستهلك الحرف الحالي)بإعادته (وتزيد موقع المؤشر (position)، مع تحديث أرقام السطر والعمود بشكل دقيق، وتتعامل مع أنماط نهاية السطر المختلفة .(\n, \r, \r\n)	Lexer::advance()
تخطي المسافات: تتخطى جميع المسافات البيضاء)فراغات، تاب، أسطر جديدة (وأي أحرف تحكم غير مرئية بما في ذلك علامة ترتيب البايتات -(UTF 8 BOM).	Lexer::skipWhitespace()
تخطي التعليقات: تتخطى التعليقات أحادية السطر التي تبدأ بـ .//	Lexer::skipComment()
التحقق من حرف المعرف: تحدد ما إذا كان الحرف يمكن أن يكون جزءاً من اسم متغير أو كلمة مفتاحية، مع دعم الأحرف الأبجدية الرقمية والأحرف العربية بناءً على نطابقات الترميز	Lexer::isIdentifierChar(char c)

الوصف الوظيفي	الدالة
قراءة الأرقام: تقرأ متتالية من الأرقام، وتتحقق من وجود نقطة عشرية لتحديد ما إذا كان الرمز الناتج هو رقم صحيح (NUMBER)أو رقم حقيقي (REAL_LITERAL).	Lexer::readNumber()
قراءة السلاسل النصية: تقرأ الكود المحصور بين علامتي اقتباس (")تتعامل مع أحرف الهروب (Escape Sequences)مثل \n و \t، وتطلق خطأ إذا كانت السلسلة غير مغلقة أو تحتوي على سطر جديد غير متوقع.	Lexer::readString()
قراءة المعرفات: تقرأ متتالية من أحرف المعرف .بعد القراءة، تتحقق مما إذا كانت السلسلة تطابق إحدى الكلمات المفتاحية المحجوزة .إذا طابقت، تعيد رمز الكلمة المفتاحية، وإلا تعيد رمز المعرف .(IDENTIFIER)	Lexer::readIdentifier()
الوظيفة الرئيسية: هي حلقة التكرار الرئيسية التي تستدعي جميع دوال القراءة والتخطي الأخرى لإنتاج قائمة نهائية من الرموز المميزة (Tokens)من الكود المصدري.	Lexer::tokenize()
التحقق من الحرف العربي: دالة مساعدة للتحقق من أن الحرف يقع ضمن نطاقات الأحرف العربية في ترميز) Windows-1256الذي يستخدمه المشروع	Lexer::isArabicChar(char c)

.2الدوال في مرحلة التحليل النحوي (Parser)

توجد هذه الدوال في ملف ArabicCompiler/Compiler/src/Parser.cpp، وهي مسؤولة عن بناء شجرة الاشتقاق النحوي .(AST)

الوصف الوظيفي	الدالة
البناء: تهيئة المحلل النحوي بقائمة الرموز المميزة التي تم إنشاؤها بواسطة المحلل اللغوي، وتعيين مؤشر البداية .(current = 0)	Parser::Parser(const std::vector<Token> &tokens)
المطابقة الاختيارية: تتحقق مما إذا كان الرمز الحالي يطابق النوع المطلوب .إذا كان كذلك، تستهلكه (advance())وتعود بـ true.إذا لم يطابق، تعود بـ false دون إطلاق خطأ.	Parser::match(TokenType type)
الاستهلاك الإلزامي: تتحقق مما إذا كان الرمز الحالي يطابق النوع المطلوب .إذا لم يطابق، تطلق خطأ نحوي (Parse Error)مع رسالة مخصصة، مما يضمن أن الكود يتبع القواعد النحوية الصارمة.	Parser::consume(TokenType type, const std::string &message)
التحقق: تعيد true إذا كان الرمز الحالي يطابق النوع المطلوب دون استهلاكه.	Parser::check(TokenType type)
التقدم: تزيد مؤشر الرمز الحالي وتعود بالرمز الذي تم تجاوزه للتو.	Parser::advance()

الوصف الوظيفي	الدالة
نهاية الملف: تعيد true إذا وصل المؤشر إلى نهاية قائمة الرموز المميزة.	Parser::isAtEnd()
نظرة مسبقة: تعيد الرمز الحالي في موقع المؤشر دون تحريكه.	Parser::peek()
الرمز السابق: تعيد الرمز الذي تم استهلاكه في الخطوة السابقة.	Parser::previous()
المزامنة: دالة للتعافي من الأخطاء النحوية .تحاول تخطي الرموز حتى تجد نقطة يمكن استئناف التحليل النحوي منها)مثل فاصلة منقوطة أو بداية جملة تحكم.(	Parser::synchronize()
نقطة الدخول: تبدأ عملية التحليل النحوي الرئيسية باستدعاء parseProgram() وتتعامل مع استثناءات الأخطاء النحوية.	Parser::parse()
تحليل البرنامج: تبدأ بتحليل الكلمة المفتاحية برنامج واسم البرنامج، ثم تقوم بتحليل التعريفات والجمل بشكل متكرر.	Parser::parseProgram()
تحليل التعريفات: تحدد نوع التعريف)متغير أو ثابت (وتستدعي الدالة المناسبة.	Parser::parseDeclaration()
تحليل تعريف المتغير: تحلل اسم المتغير، ونوعه الاختياري :) صحيح(، وقيمته الابتدائية الاختيارية =)قيمة.(	Parser::parseVariableDeclaration()
تحليل تعريف الثابت: تحلل اسم الثابت وقيمته الإلزامية.	Parser::parseConstantDeclaration()
تحليل الجمل: تحدد نوع الجملة الحالية)تعيين، طباعة، شرط، حلقة (وتستدعي دالة التحليل المناسبة.	Parser::parseStatement()
تحليل جملة التعيين: تحلل تعيين قيمة لتعبير ما إلى متغير أو عنصر في قائمة .(Array Indexing)	Parser::parseAssignment()
تحليل جملة الطباعة: تحلل الكلمة المفتاحية اطبع والتعبير المراد طباعته.	Parser::parsePrintStatement()
تحليل جملة القراءة: تحلل الكلمة المفتاحية اقرأ واسم المتغير الذي ستُخزن فيه القيمة المدخلة.	Parser::parseReadStatement()
تحليل جملة الشرط)**اذا**:(تحلل التعبير الشرطي، ثم كتلة الكود الخاصة بـ فان، وكتلة وإلا الاختيارية.	Parser::parseIfStatement()
تحليل حلقة)**طالما**:(تحلل التعبير الشرطي وكتلة الكود التي يجب تكرارها طالما كان الشرط صحيحاً.	Parser::parseWhileStatement()
تحليل حلقة)**كرر...حتى**:(تحلل كتلة الكود التي يجب تكرارها والتعبير الشرطي الذي يحدد متى يجب التوقف.	Parser::parseRepeatStatement()
تحليل حلقة)**كرر...الى**:(تحلل حلقة التكرار المحددة)عداد، قيمة بداية، قيمة نهاية، خطوة اختيارية.(	Parser::parseForStatement()

الوصف الوظيفي	الدالة
تحليل الكتلة: تحلل مجموعة من الجمل المحصورة بين أقواس أو كلمات مفتاحية تحدد بداية ونهاية الكتلة.	Parser::parseBlock()
تحليل النوع: تحلل نوع البيانات)سواء كان نوعاً بدائياً مثل صحيح أو نوعاً مركباً مثل قائمة أو سجل.(	Parser::parseType()
تحليل نوع القائمة: تحلل تعريف نوع القائمة وحجمها.	Parser::parseArrayType()
تحليل نوع السجل: تحلل تعريف نوع السجل وحقوله.	Parser::parseRecordType()
تحليل التعبير: تبدأ عملية تحليل التعابير بأقل أسبقية)التعيين.(	Parser::parseExpression()
تحليل العمليات المنطقية: تحلل عمليات أو و و المنطقية.	Parser::parseLogicalOr(), parseLogicalAnd()
تحليل عمليات المساواة والمقارنة: تحلل عمليات ==, !=, >, <, >=, <=.	Parser::parseEquality(), parseComparison()
تحليل العمليات الرياضية: تحلل عمليات الجمع والطرح (parseTerm)، وعمليات الضرب والقسمة .(parseFactor)	Parser::parseTerm(), parseFactor()
تحليل العمليات الأحادية: تحلل العمليات التي تعمل على معامل واحد مثل النفي المنطقي (!)أو النفي الرياضي .(-)	Parser::parseUnary()
تحليل العناصر الأولية: تحلل الوحدات الأساسية للتعبير مثل الأرقام، السلاسل النصية، المعرفات، أو التعابير المحصورة بين أقواس.	Parser::parsePrimary()

3.الدوال في مرحلة توليد الكود (Compiler)

توجد هذه الدوال في ملف ArabicCompiler/Compiler/src/Compiler.cpp، وهي مسؤولة عن توليد الكود الوسيط وكود الهدف.

الوصف الوظيفي	الدالة
البناء: تهيئة المترجم بتصفير عدادات العلامات (labelCounter)والمتغيرات المؤقتة (tempVarCounter).	Compiler::Compiler()
توليد علامة: تولد علامة فريدة)مثل L0, L1) تستخدم في تعليمات القفز في الكود الوسيط.	Compiler::generateLabel()
توليد متغير مؤقت: تولد اسماً لمتغير مؤقت)مثل t0, t1) يستخدم لتخزين نتائج العمليات الحسابية الوسيطة.	Compiler::generateTempVar()
إصدار تعليمة: تضيف تعليمة كود وسيط جديدة إلى قائمة التعليمات، مع تحديد نوع التعليمة ومعاملاتها الثلاثة.	Compiler::emit(InstructionType type, ...)
علامة السلسلة النصية: تدير جدول السلاسل النصية، وتخصص علامة فريدة لكل سلسلة نصية)مثل str_0) لاستخدامها في تعليمات الطباعة أو التخزين.	Compiler::getStringLabel(const std::string &literal)
نقطة الدخول: تبدأ عملية الترجمة الشاملة، وتستدعي compileProgram()، وتضيف تعليمة HALT في النهاية، وتتعامل مع الأخطاء.	Compiler::compile(std::unique_ptr<ProgramNode> program)
ترجمة البرنامج: تمر على عقد AST الخاصة بالتعريفات أولاً ثم الجمل، وتستدعي compileStatement() لكل منها.	Compiler::compileProgram(ProgramNode *program)
ترجمة الجمل: دالة "الموزع "الرئيسية التي تحدد نوع عقدة AST وتستدعي دالة الترجمة المتخصصة المناسبة)مثل compileIf, compileWhile).	Compiler::compileStatement(ASTNode *statement)
ترجمة تعريف المتغير: تسجل المتغير في جدول الرموز وتحدد نوعه .إذا كانت هناك قيمة ابتدائية، تولد تعليمات لتعيين هذه القيمة.	Compiler::compileVariableDeclaration(...)
ترجمة تعريف الثابت: تسجل الثابت في جدول الرموز وتولد تعليمات لتعيين قيمته.	Compiler::compileConstantDeclaration(...)
ترجمة التعيين: تولد تعليمات لحساب قيمة الجانب الأيمن، ثم تعليمة STORE لتخزين النتيجة في المتغير الهدف.	Compiler::compileAssignment(...)
ترجمة الطباعة: تولد تعليمة PRINT لطباعة سلسلة نصية أو قيمة تعبير أو متغير.	Compiler::compilePrint(...)
ترجمة القراءة: تولد تعليمة READ لقراءة قيمة من المستخدم وتخزينها في متغير محدد.	Compiler::compileRead(...)

الوصف الوظيفي	الدالة
ترجمة الشرط: تولد تعليمات JUMP شرطية وغير شرطية لتمثيل منطق اذا...فان...وإلا باستخدام العلامات (L0, L1).	Compiler::compileIf(...)
ترجمة حلقة)**طالما**:(تولد تعليمات JUMP لتمثيل حلقة التكرار، حيث يتم فحص الشرط في كل دورة.	Compiler::compileWhile(...)
ترجمة حلقة)**كرر...حتى**:(تولد تعليمات JUMP لتمثيل حلقة التكرار التي يتم فيها فحص الشرط في نهاية الدورة.	Compiler::compileRepeat(...)
ترجمة حلقة)**كرر...الى**:(تولد تعليمات JUMP و ADD و SUB لتمثيل حلقة التكرار المحددة.	Compiler::compileFor(...)
ترجمة التعبير: دالة متكررة تولد تعليمات الكود الوسيط للعمليات الحسابية والمنطقية المختلفة.	Compiler::compileExpression(...)
توليد الكود الوسيط: تكتب قائمة التعليمات الوسيطة وجدول الرموز إلى ملف نصي.	Compiler::generateIntermediateCode(const std::string &filename)
توليد كود C: تكتب الكود المكافئ بلغة C إلى ملف، بما في ذلك تعريف المتغيرات المؤقتة والتعليمات.	Compiler::generateCCode(const std::string &filename)
توليد كود التجميع: تكتب الكود المكافئ بلغة التجميع MIPS إلى ملف، مع استخدام سجلات MIPS لتمثيل العمليات.	Compiler::generateAssembly(const std::string &filename)

التوثيق الكامل

المكون	الوصف
الكلمات المفتاحية	برنامج ,اذا ,فان ,وإلا ,اطبع ,اقرأ ,طالما ,كرر ,حتى ,نهاية ,الى ,اضف ,متغير ,ثابت ,إجراء ,صحيح , حقيقي ,منطقي ,خيط ,نوع ,قائمة ,سجل
أنواع البيانات المدعومة	صحيح (Integer), حقيقي (Real/Double), خيط (String), منطقي.(Boolean)
هياكل البيانات المدعومة	قائمة (Array), سجل.(Record)
هياكل التحكم	جملة اذا...فان...وإلا ,حلقة طالما...كرر ,حلقة كرر...حتى ,حلقة كرر...الى.(For Loop)
الترميز	يدعم المترجم الترميز العربي في الكلمات المفتاحية وأسماء المتغيرات والسلاسل النصية.

5.الخلاصة

يمثل مشروع المترجم العربي (Arabic Compiler)إنجازاً تقنياً مهماً في مجال دعم اللغات الطبيعية في البرمجة .البنية المعمارية المعتمدة على C++وCMakeتضمن الكفاءة، بينما يوفر الفصل الواضح بين مراحل التحليل اللغوي والنحوي وتوليد الكود مرونة كبيرة في التوسع والتطوير المستقبلي .إن قدرته على توليد كود وسيط وكود Cوكود تجميعي يجعله أداة تعليمية وتطبيقية قوية لفهم مبادئ عمل المترجمات.