ΠΑΝΕΠΙΣΤΗΜΙΟ ΠΕΙΡΑΙΩΣ

UNIVERSITY OF PIRAEUS

ΤΜΗΜΑ ΠΛΗΡΟΦΟΡΙΚΗΣ

ΠΜΣ ΠΛΗΡΟΦΟΡΙΚΗ

ΛΟΓΙΚΟΣ ΠΡΟΓΡΑΜΜΑΤΙΣΜΟΣ

ΕΡΓΑΣΙΑ ΕΞΑΜΗΝΟΥ

ΣΤΑΜΑΤΟΠΟΥΛΟΣ ΙΑΣΩΝ - ΜΠΠΛ20077

ΔΙΔΑΣΚΩΝ

ΚΑΘΗΓΗΤΗΣ ΘΕΜΙΣΤΟΚΛΗΣ ΠΑΝΑΓΙΩΤΟΠΟΥΛΟΣ

ΑΘΗΝΑ

Ακαδημαϊκο Ετος 2020-2021

Περιεχόμενα

1 Εισαγωγή

Στην παρούσα εργασία υλοποιείται πρόγραμμα επίλυσης λαβυρίνθου με τον αλγόριθμο αναζήτησης κατά βάθος *(Depth-First)* με χρήση της γλώσσας *Prolog*. Εκτός της σωστής επίλυσης, σκοπός είναι να παρουσιαστεί ένας τρόπος επικοινωνίας της *Prolog* με άλλα προγραμματιστικά οικοσυστήματα.

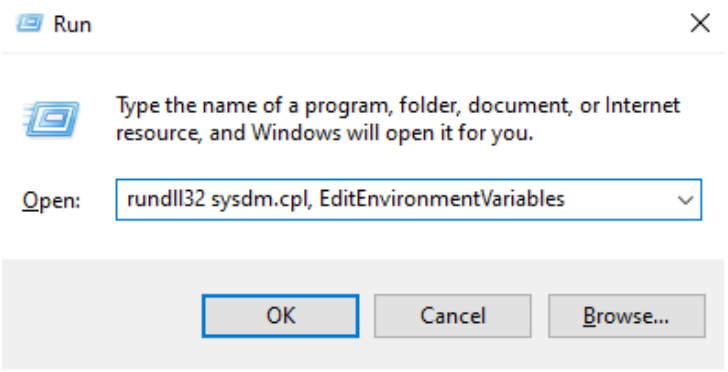
Πιο συγκεκριμένα υλοποιήθηκε γραφικό περιβάλλον σε κονσόλα για την απεικόνιση του λαβυρίνθου και τη διεπαφή με το χρήστη σε γλώσσα *C#* και βιβλιοθήκη σε γλώσσα *C++* που ενθυλακώνει την εκτέλεση της μηχανής της *Prolog*. Τέτοιου τύπου διεπαφές μεταξύ γλωσσών είναι πολύ σημαντικές γιατί μας επιτρέπουν να αντιμετωπίζουμε τα επιμέρους προβήματα μιας διαδικασίας επίλυσης στα εκάστοτε εξειδικευμένα περιβάλλοντα. Δηλαδή, ενώ η *Prolog* έχει εξαιρετικά εργαλεία για την επίλυση αναζήτησης κατά βάθος και μπορεί να γραφεί κώδικας για το συγκεκριμένο πρόβλημα μικρός σε έκταση και κατανοητός, από την άλλη είναι πολύ δύσκολο να γραφούν προγράμματα διεπαφής με χρήστες και γραφικής απεικόνισης. Τα τελευταία, όμως, αποτελούν πολύ εύκολη διαδικασία στον κόσμο της *C#*. Η απευθείας διεπαφή μεταξύ *Prolog* και *C#* παρουσιάζει δυσκολίες και δεν υπάρχουν αυτή τη στιγμή πολλές και έμπιστες βιβλιοθήκες σε *C#* που να διατηρούνται και να βρίσκονται σε συνεχή ανάπτυξη και αναβάθμιση. Εδώ, λοιπόν, χρησιμοποιείται η *C++* ως γέφυρα και αναλαμβάνει την επικοινωνία μεταξύ *Prolog* και *C#*, αφού υπάρχει τόσο επίσημη βιβλιοθήκη για την χρήση της μηχανής της *Prolog* μέσα από τη *C++* όσο και άψογα εργαλεία διεπαφής μεταξύ *C++* και *C#*. Παρατηρείται, δηλαδή, ότι το πρόβλημα αποδομήθηκε με βάση τη φύση του στις εκάστοτε εξειδικευμένες εκτελεστικές μονάδες.

Στα επόμενα κεφάλαια θα δοθούν οδηγίες εγκατάστασης των διάφορων εργαλείων ανάπτυξης, θα γίνει ανάλυση της στρατηγικής της ανάπτυξης, θα δοθούν λεπτομέρειες υλοποίησης όλων των σημαντικών κομματιών των προγραμμάτων και θα παρουσιαστούν παραδείγματα εκτέλεσης του τελικού προϊόντος.

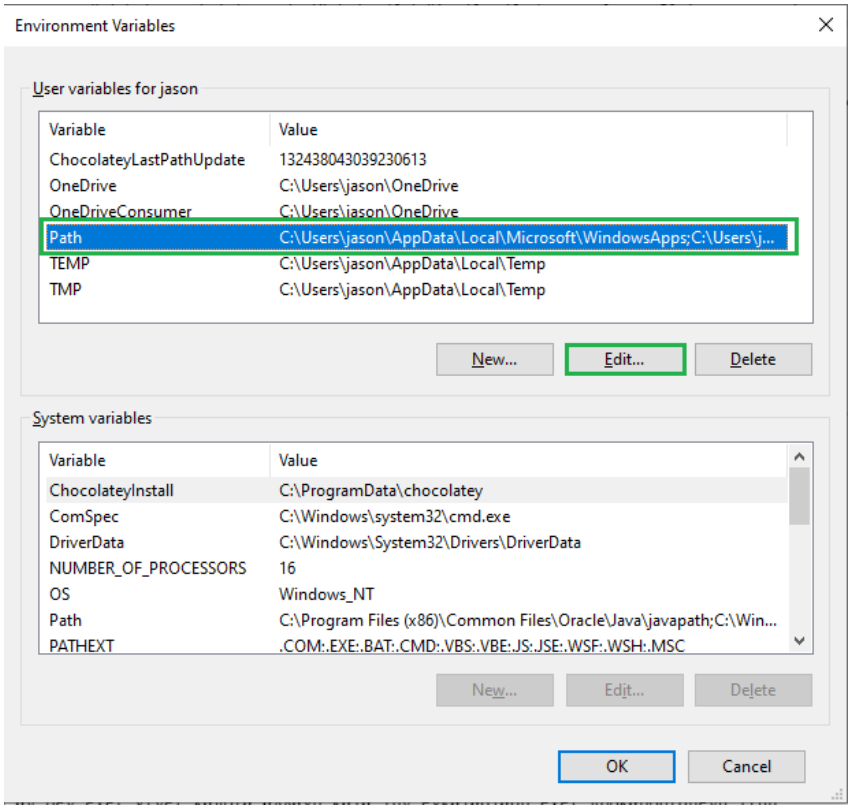
2 Εργαλεία Ανάπτυξης

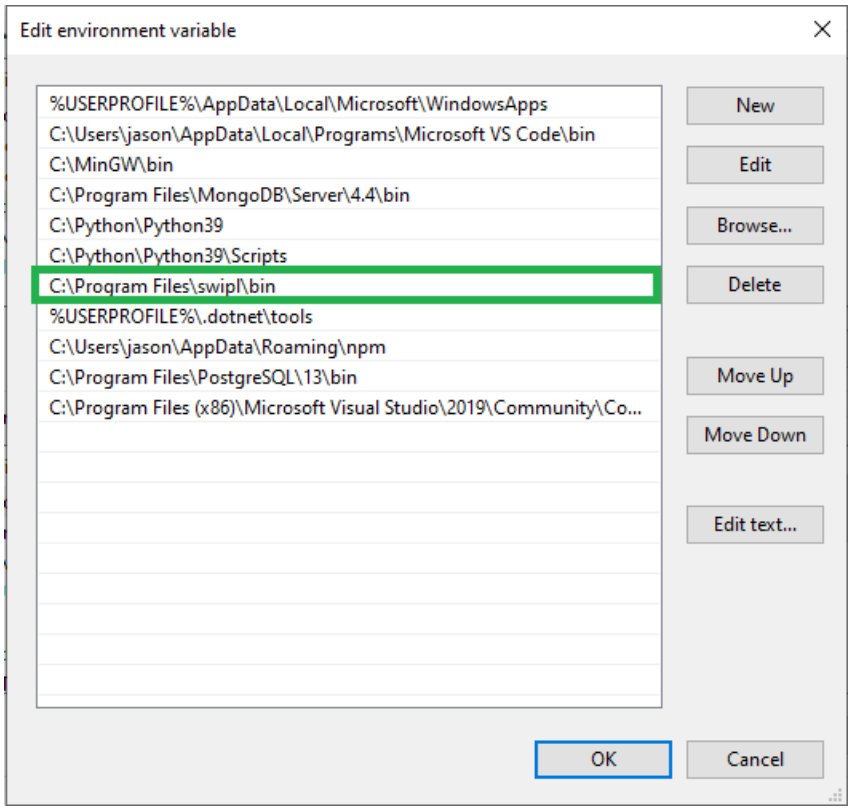
Αρχικά, είναι απαραίτητη η εγκατάσταση της *Prolog* στον υπολογιστή. Μπορεί κανείς να την κατεβάσει δωρεάν στο σύνδεσμο https://www.swi-prolog.org/download/stable. Μετά την εγκτάσταση θα πρέπει να προστεθεί η διαδρομή στην οποία βρίσκεται το εκτελέσιμο αρχείο του διερμηνέα της *Prolog* στις μεταβλητές περιβάλλοντος του συστήματος. Αυτό γίνεται ώστε να υπάρχει η δυνατότητα εκτέλεσης της *Prolog* από οποιδήποτε περιβάλλον χωρίς να χρειάζεται κάθε φορά να μεταφερόμαστε στην συγκεκριμένη διαδρομή. Ένας γρήγορος τρόπος είναι ο παρακάτω:

Χρησιμοποιώντας τη συντόμευση *(Windows Logo Key + R)* ανοίγουμε το πρόγραμμα *Run* των *Windows*. Στο κενό πεδίο γράφουμε *rundll32 sysdm.cpl, EditEnvironmentVariables* και πατάμε *OK*.

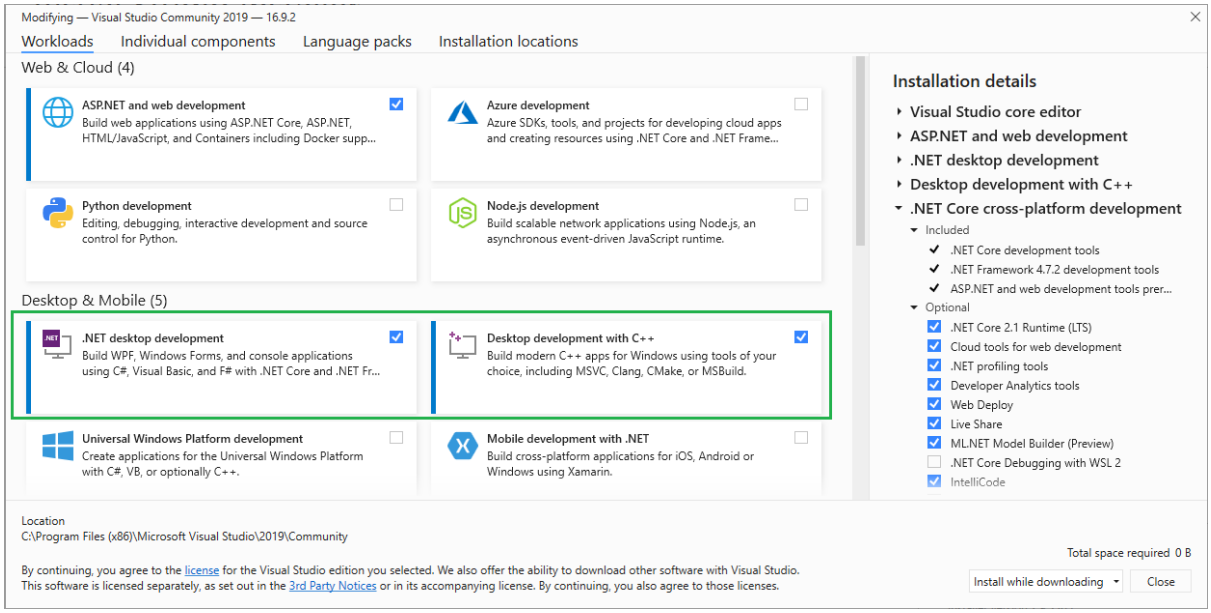


Έπειτα, εμφανίζεται ένα παράθυρο στο οποίο μπορούμε να επιλέξουμε κάποια μεταβλητή περιβάλλοντος και να της αλλάξουμε τιμή. Οπότε, επιλέγουμε τη μεταβλητή *Path* και την επιλογή *Edit* και προσθέτουμε τη διαδρομή του εκτελέσιμου αρχείου της *Prolog*. Η διαδρομή αυτή αν δεν έχει γίνει κάποια αλλαγή κατά την εγκατάσταση έχει προκαθορισμένη τιμή *C:\Program Files\swipl\bin*.


Environment Variables
User variables for jason
Variable
Value
ChocolateyLastPathUpdate
132438043039230613
OneDrive
C:\Users\jason\OneDrive
OneDriveConsumer
C:\Users\jason\OneDrive
Path
C:\Users\jason\AppData\Local\Microsoft\WindowsApps;C:\Users\j...
TEMP
C:\Users\jason\AppData\Local\Temp
TMP
C:\Users\jason\AppData\Local\Temp
New...
Edit...
Delete
System variables
Variable
Value
ChocolateyInstall
C:\ProgramData\chocolatey
ComSpec
C:\Windows\system32\cmd.exe
DriverData
C:\Windows\System32\Drivers\DriverData
NUMBER_OF_PROCESSORS
16
OS
Windows_NT
Path
C:\Program Files (x86)\Common Files\Oracle\Java\javapath;C:\Win...
PATHEXT
.COM;.EXE;.BAT;.CMD;.VBS;.VBE;.JS;.JSE;.WSF;.WSH;.MSC
New...
Edit...
Delete
OK
Cancel


Edit environment variable
%USERPROFILE%\AppData\Local\Microsoft\WindowsApps
C:\Users\jason\AppData\Local\Programs\Microsoft VS Code\bin
C:\MinGW\bin
C:\Program Files\MongoDB\Server\4.4\bin
C:\Python\Python39
C:\Python\Python39\Scripts
C:\Program Files\swipl\bin
%USERPROFILE%\.dotnet\tools
C:\Users\jason\AppData\Roaming\npm
C:\Program Files\PostgreSQL\13\bin
C:\Program Files (x86)\Microsoft Visual Studio\2019\Community\Co...
New
Edit
Browse...
Delete
Move Up
Move Down
Edit text...
OK
Cancel

Για την ανάπτυξη του κώδικα της *C#* και *C++* χρησιμοποιήθηκε το *Visual Studio*. Μπορείτε να προμηθευτείτε τη δωρεάν έκδοση (*Community*) στο σύνδεσμο https://visualstudio.microsoft.com/downloads/. Κατά την εγκατάσταση του *Visual Studio* θα πρέπει να γίνει επιλογή των πακέτων *.NET desktop development* και *Desktop development with C++* όπως φαίνεται παρακάτω:



Τέλος, προτείνεται η εγκτάσταση της τελευταίας έκδοσης του *.NET*. Κατά την εκπόνηση της παρούσας εργασίας είχε πρόσφατα δωθεί έκδοση της νέας *multi-platform* υλοποίησης *.NET 5*. Η ακριβής έκδοση που χρησιμοποιήθηκε είναι η 5.0.201. Μπορεί κανείς να επιλέξει όποια έκδοση του *.NET 5* επιθυμεί στο σύνδεσμο https://dotnet.microsoft.com/download/dotnet/5.0. Είναι καλύτερα να επιλέξουμε τη *multi-platform* έκδοση γιατί επιτρέπει να γραφτεί ένας κώδικας που θα τρέχει σε οποιδήποτε λειτουργικό. Για παράδειγμα, αν και στην παρούσα εργασία το τελικό προϊόν είναι για *Windows*, μπορούμε πολύ εύκολα με τον ίδιο κώδικα της *C#* να παράγουμε εκτελέσιμο για *linux*.

3 Επίλυση Αναζήτησης Κατά Βάθος Σε Prolog

Για τη διατύπωση του προβλήματος της επίλυσης του λαβυρίνθου θεωρήθηκαν οι εξής παραδοχές:

- Ο λαβύρινθος παριστάνεται με σημεία (X, Y) και είναι νοητά τοποθετημένος σε ένα καρτεσιανό επίπεδο ώστε το κάτω αριστερά σημείο να έχει συνταταγμένες $(0, 0)$.
- Ένα σημείο μπορεί να είναι η αρχή (είσοδος), το τέλος (έξοδος) ή τοίχος. Αν δεν δηλωθεί ένα σημείο ως ένα από αυτά, τότε θεωρείται ελεύθερο σημείο.
- Οι επιτρεπτές μεταβάσεις από σημείο σε σημείο είναι με κατεύθυνση αριστερά, δεξιά, πάνω ή κάτω.
- Οι διαστάσεις του λαβυρίνθου, η αρχή, το τέλος και τα σημεία που είναι τοίχος δίνονται στατικά ως *facts*. Στην πραγματικότητα τα *facts* που θα παρουσιαστούν στη συνέχεια χρησιμοποιούνται απλά σαν οδηγός για την παραγωγή του κώδικα δυναμικά από το εκτελέσιμο της *C#*. Άρα είναι στατικά στο πλαίσιο της *Prolog* ως *facts*, αλλά απόλυτα δυναμικά στο πλαίσιο του συνολικού προγράμματος.

Αν συμβολίσουμε με S την αρχή, G το τέλος, $*$ τα σημεία με τοίχο και 0 τα ελέθερα σημεία, ένα παράδειγμα λαβυρίνθου θα μπορούσε να είναι το παρακάτω:

S * * * * *
0 0 0 0 0 0
* 0 * 0 * 0
* 0 * 0 * 0
0 0 0 * 0 G
* * 0 0 0 *

Αυτός ο λαβύρινθος έχει διαστάσεις $6 * 6$. Οι αντίστοιχες συντεταγμένες τους με βάση τις παραδοχές που έγιναν παραπάνω θα είναι $S(0, 5)$, $G(5, 1)$ και τα σημεία που υπάρχει τοίχος είναι τα $(1, 5)$, $(2, 5)$, $(3, 5)$, $(4, 5)$, $(5, 5)$, $(0, 3)$, $(2, 3)$, $(4, 3)$, $(0, 2)$, $(2, 2)$, $(4, 2)$, $(3, 1)$, $(0, 0)$, $(1, 0)$, $(5, 0)$.

Έτσι λοιπόν μπορούμε να καταλήξουμε στα παρακάτω κατηγορήματα σε κώδικα *Prolog*:

```
boundX(6).
boundY(6).

start(0,5).
end(5,1).

wallPoint(1,5).
wallPoint(2,5).
wallPoint(3,5).
wallPoint(4,5).
wallPoint(5,5).
wallPoint(0,3).
```

```
13 wallPoint(2,3).
14 wallPoint(4,3).
15 wallPoint(0,2).
16 wallPoint(2,2).
17 wallPoint(4,2).
18 wallPoint(3,1).
19 wallPoint(0,0).
20 wallPoint(1,0).
21 wallPoint(5,0).
```

Αναπτύσσουμε στη συνέχεια τους κανόνες του προγράμματος. Αρχικά χρειαζόμαστε έναν κανόνα ο οποίος να ελέγχει αν ένα ζεύγος (X, Y) είναι εντός ορίων του λαβυρίνθου. Δηλαδή, αν είναι αυστηρά μικρότερα τα X και Y από τις αντίστοιχες διαστάσεις, καθώς και μεγαλύτερα ή ίσα με το 0.

```
isInBounds(X,Y) :- boundX(X_DIM),
                   boundY(Y_DIM),
                   X < X_DIM,
                   Y < Y_DIM,
                   X >= 0,
                   Y >= 0.
```

Έπειτα πρέπει να υπάρχει ένας κανόνας που να καθορίζει αν ένα σημείο είναι ελεύθερο προς διάβαση. Αυτό σημαίνει ότι είναι εντός ορίων και ότι δεν είναι η αρχή S ή τοίχος.

```
isFreePoint(X,Y) :- isInBounds(X,Y),
                    \+start(X,Y),
                    \+wallPoint(X,Y).
```

Στη συνέχεια πρέπει να αναπτύξουμε τους κανόνες που αφορούν την κίνηση από σημείο σε σημείο, δηλαδή που θα μετασχηματίζουν σωστά τις συντεταγμένες ανάλογα με την κατεύθυνση. Μπορούμε λοιπόν να υλοποιήσουμε έναν κανόνα για κάθε κατεύθυνση και έπειτα ένα γενικό κανόνα κίνησης που ενθυλακώνει την αναζήτηση σημείου προς όλες τις κατευθύνσεις.

```
goNorth(X,Y,X1,Y1) :- X1 is X, Y1 is Y + 1, isFreePoint(X1,Y1).
goSouth(X,Y,X1,Y1) :- X1 is X, Y1 is Y - 1, isFreePoint(X1,Y1).
goEast(X,Y,X1,Y1) :- X1 is X + 1, Y1 is Y, isFreePoint(X1,Y1).
goWest(X,Y,X1,Y1) :- X1 is X - 1, Y1 is Y, isFreePoint(X1,Y1).

goSearch(X,Y,X1,Y1) :- goNorth(X,Y,X1,Y1);
                       goSouth(X,Y,X1,Y1);
                       goEast(X,Y,X1,Y1);
                       goWest(X,.Y,X1,Y1).
```

Τέλος έρχεται ο αναδρομικός κανόνας κατά τον οποιο θα γίνεται αναζήτηση μεχρι να βρεθούν όλες οι πιθανές αλληλουχίες σημείων που καταλήγουν στο σημείο του τέλους G. Κάθε σημείο εξετάζεται αν είναι ελεύθερο και αν δεν έχει ήδη περπατηθεί. Αν αυτοί οι δύο ισχυρισμοί είναι αληθείς, τότε το σημείο προστίθεται στην ήδη υπάρχουσα διαδρομή και ο κανόνας ξανακαλείται αναδρομικά με νέες συντεταγμένες και νέα συνολική διαδρομή. Ως συνθήκη τερματισμού θεωρείται προφανώς η εύρεση του σημείου του τέλους. Θα παρατηρήσετε ότι υπάρχει άλλος ένας κανόνας που χρησιμεύει για τη διεπαφή με την $C++$. Πρόκειται για τον κανόνα *solve(Path)*. Είναι γενικά καλό όταν δημιουργούνται ειδικά σημεία διεπαφής να είναι όσο πιο απλά γίνονται στη χρήση. Εν προκειμένω, επιλέγουμε να υπάρχει μια *solve* με μόνο μια παράμετρο αφού είναι δυνατό, ώστε να διευκολύνουμε τη διαδικασία καλέσματος από τη $C++$.

```
solve(Path) :- start(X,Y), solve(X,Y,[],Path).
solve(X,Y,PathHistory,Path) :- end(X,Y), PathHistory = Path.
solve(X,Y,PathHistory, Path) :-
    goSearch(X,Y,X1,Y1),
    \+member([X1, Y1], PathHistory),
    append(PathHistory,[[X1,Y1]],NewPathHistory),
    solve(X1,Y1,NewPathHistory,Path).
```

Έτσι καταλήγουμε στο συνολικό κώδικα για την επίλυση του παραδείγματος:

```
1  boundX(6).
2  boundY(6).
3
4  start(0,5).
5  end(5,1).
6
7  wallPoint(1,5).
8  wallPoint(2,5).
9  wallPoint(3,5).
10 wallPoint(4,5).
11 wallPoint(5,5).
12 wallPoint(0,3).
13 wallPoint(2,3).
14 wallPoint(4,3).
15 wallPoint(0,2).
16 wallPoint(2,2).
17 wallPoint(4,2).
18 wallPoint(3,1).
19 wallPoint(0,0).
20 wallPoint(1,0).
21 wallPoint(5,0).
22
23 isInBounds(X,Y) :- boundX(X_DIM),
24                    boundY(Y_DIM),
```

```
                    X < X_DIM,
                    Y < Y_DIM,
                    X >= 0,
                    Y >= 0.

isFreePoint(X,Y) :- isInBounds(X,Y),
                    \+start(X,Y),
                    \+wallPoint(X,Y).

goNorth(X,Y,X1,Y1) :- X1 is X, Y1 is Y + 1, isFreePoint(X1,Y1).
goSouth(X,Y,X1,Y1) :- X1 is X, Y1 is Y - 1, isFreePoint(X1,Y1).
goEast(X,Y,X1,Y1) :- X1 is X + 1, Y1 is Y, isFreePoint(X1,Y1).
goWest(X,Y,X1,Y1) :- X1 is X - 1, Y1 is Y, isFreePoint(X1,Y1).

goSearch(X,Y,X1,Y1) :- goNorth(X,Y,X1,Y1);
                       goSouth(X,Y,X1,Y1);
                       goEast(X,Y,X1,Y1);
                       goWest(X,.Y,X1,Y1).

solve(Path) :- start(X,Y), solve(X,Y,[],Path).
solve(X,Y,PathHistory,Path) :- end(X,Y), PathHistory = Path.
solve(X,Y,PathHistory, Path) :-
    goSearch(X,Y,X1,Y1),
    \+member([X1, Y1], PathHistory),
    append(PathHistory,[[X1,Y1]],NewPathHistory),
    solve(X1,Y1,NewPathHistory,Path).
```

Ακολουθεί η εκτέλεση του παραπάνω παραδείγματος στην κονσόλα.

```
jason@HOME-SERVER  ~\source\repos\PathFinder  master ≡
                          [20:13]
❯ swipl.exe .\pathfinder.pl
Welcome to SWI-Prolog (threaded, 64 bits, version 8.2.2)
SWI-Prolog comes with ABSOLUTELY NO WARRANTY. This is free software.
Please run ?- license. for legal details.

For online help and background, visit https://www.swi-prolog.org
For built-in help, use ?- help(Topic). or ?- apropos(Word).

1 ?- set_prolog_flag(answer_write_options,[max_depth(0)]).
true.

2 ?- solve(Path).
Path = [[0,4],[1,4],[1,3],[1,2],[1,1],[2,1],[2,0],[3,0],[4,0],[4,1],[5,1]] ;
Path = [[0,4],[1,4],[2,4],[3,4],[4,4],[5,4],[5,3],[5,2],[5,1]] ;
false.

3 ?-
```

Παρατηρούμε ότι βρέθηκαν δύο δυνατές διαδρομές. Ας τις ελέγξουμε για να δούμε την εγκυρότητα του αποτελέσματος. Κάθε σημείο της διαδρομής θα αναπαρίσταται με μια παύλα.

1. $(0,4) \Rightarrow (1,4) \Rightarrow (1,3) \Rightarrow (1,2) \Rightarrow (1,1) \Rightarrow (2,1) \Rightarrow (2,0) \Rightarrow (3,0) \Rightarrow (4,0) \Rightarrow (4,1) \Rightarrow (5,1)$

S * * * * *
- - 0 0 0 0
* - * 0 * 0
* - * 0 * 0
0 - - * - G
* * - - - *

2. $(0,4) \Rightarrow (1,4) \Rightarrow (2,4) \Rightarrow (3,4) \Rightarrow (4,4) \Rightarrow (5,4) \Rightarrow (5,3) \Rightarrow (5,2) \Rightarrow (5,1)$

S * * * * *
- - - - - -
* 0 * 0 * -
* 0 * 0 * -
0 0 0 * 0 G
* * 0 0 0 *

Άρα το πρόγραμμα λειτουργεί σωστά δίνοντας και τις δύο λύσεις που περιμέναμε. Οπότε έχουμε έτοιμη τη λογική που θα λύσει το λαβύρινθο και μένει να γίνει η σύνδεση και επικοινωνία με τη *C++* και στη συνέχεια με τη *C#* ώστε αφενός να δίνονται με κάποιο τρόπο δυναμικά τα κατηγορήματα ύστερα από αλληλεπίδραση με το χρήστη, αφετέρου να παρουσιάζεται στο χρήστη η τελική λύση.

4 Διεπαφή C++ με Prolog

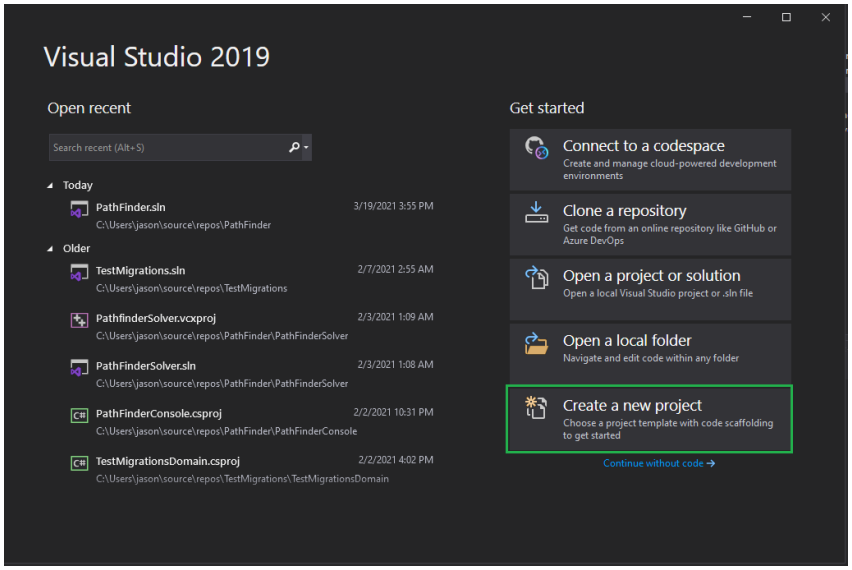
Κατά την εγκατάσταση της *swi-prolog*, εγκαθίστανται επίσης τα επίσημα εργαλεία που χρειαζόμαστε για τη διεπαφή με τη *C++*. Αυτά είναι στην ουσία οι βιβλιοθήκες που ενθυλακώνουν τις κλήσεις προς το διερμηνέα της *Prolog* καθώς και τα αντίστοιχα αρχεία κεφαλλίδας *(headers)* στα οποία βρίσκονται οι δηλώσεις συναρτήσεων, τύπων, μακροεντολών κλπ. Σχετικά με τα εργαλεία διεπαφής της *C++* και της *swi-prolog* υπάρχει το επίσημο εγχειρίδιο ελεύθερα στο διαδίκτυο.

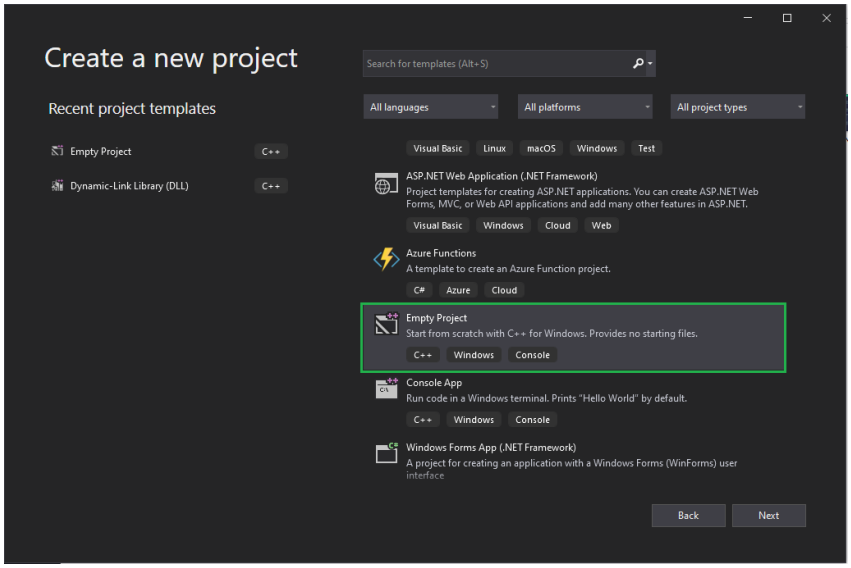
Για το παρόν πρόβλημα χρησιμοποιήθηκε η κεφαλλίδα *SWI-Prolog.h* που βρίσκεται στη διαδρομή *C:\Program Files\swipl\include* και η αντίστοιχη βιβλιοθήκη *libswipl.dll* που βρίσκεται στη διαδρομή *C:\Program Files\swipl\bin*. Για την ακρίβεια, όπως θα δούμε και στη συνέχεια χρησιμοποιήθηκε η στατική βιβλιοθήκη *libswipl.dll.a* που με τη σειρά της καλεί τη δυναμική βιβλιοθήκη *libswipl.dll*, γεγονός που δεν έχει κάποια σημασία για τον προγραμματιστή της *C++* και απλά σημειώνεται αναφορικά.

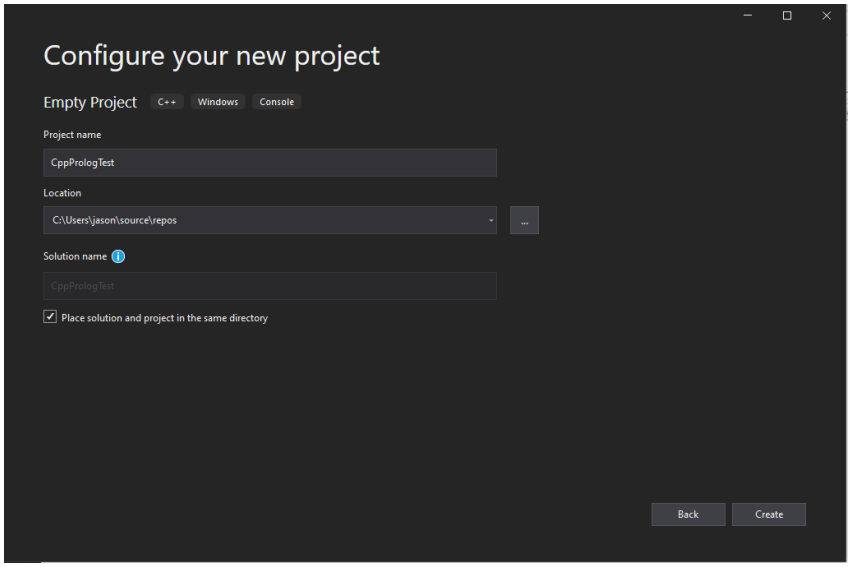
Άρα, συγκεντρωτικά, καλούμαστε να υλοποιήσουμε μια βιβλιοθήκη σε *C++*, η οποία να περιλαμβάνει κλήσεις στο διερμηνέα της *Prolog* και να εκθέσει μια συνάρτηση προς τη *C#* που θα δέχεται με κάποιο τρόπο τα δεδομένα του χρήστη και θα επιστρέφει τις τελικές λύσεις. Πιο συγκεκριμένα, θα δέχεται ως παράμετρο τη διαδρομή ενός αρχείου *Prolog* που έχει δημιουργηθεί δυναμικά με βάση τις επιλογές του χρήστη και θα το στέλνει στο διερμηνέα της *Prolog*. Έπειτα θα επιστρέφει τις λίστες με τα σημεία σε μορφή συμβολοσειράς.

Τα βήματα για τη δημιουργία μιας απλής βιβλιοθήκης σε *C++* με τη βοήθεια του *Visual Studio* έχουν ως εξής:

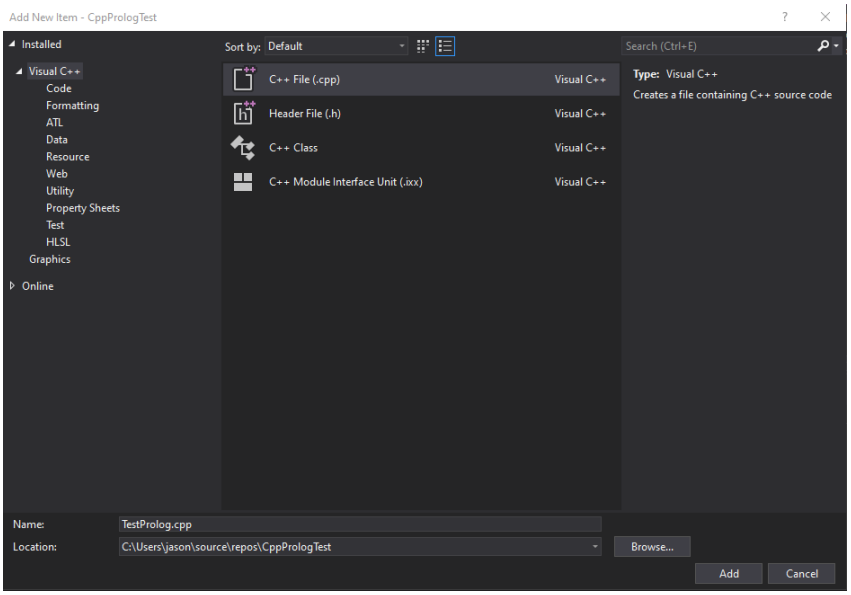
Επιλέγουμε νέο *Project* και στη συνέχεια ένα άδειο *Project C++*. Έπειτα δίνουμε ένα όνομα στο *Project*.

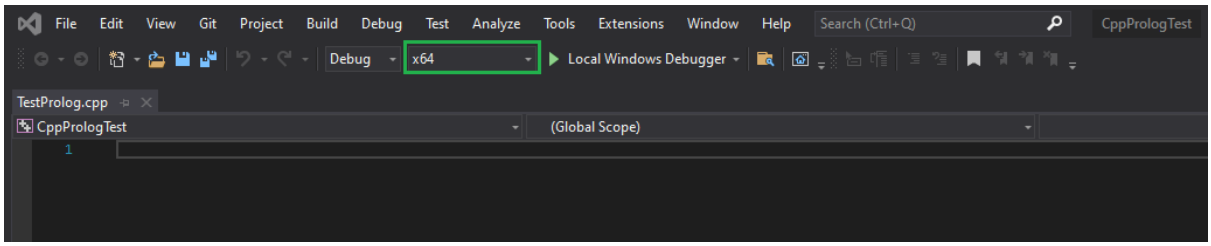




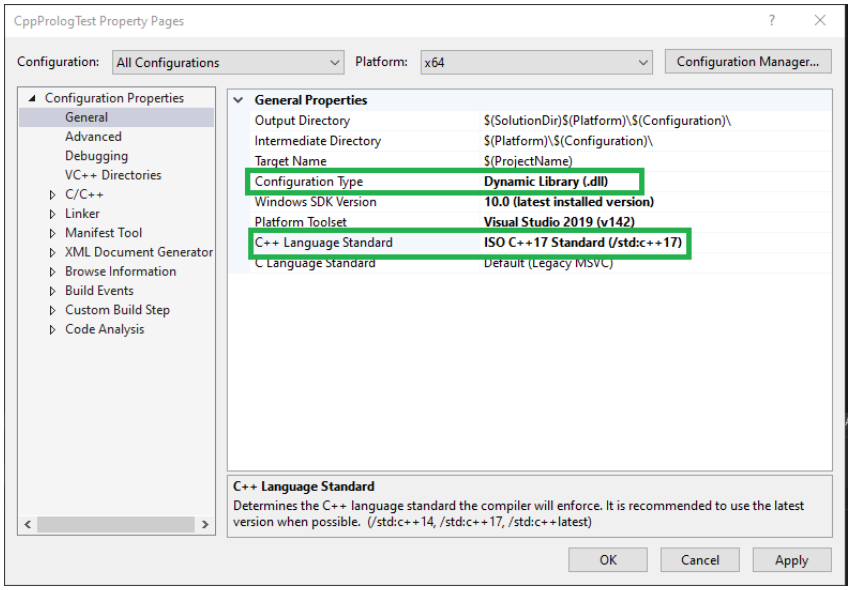


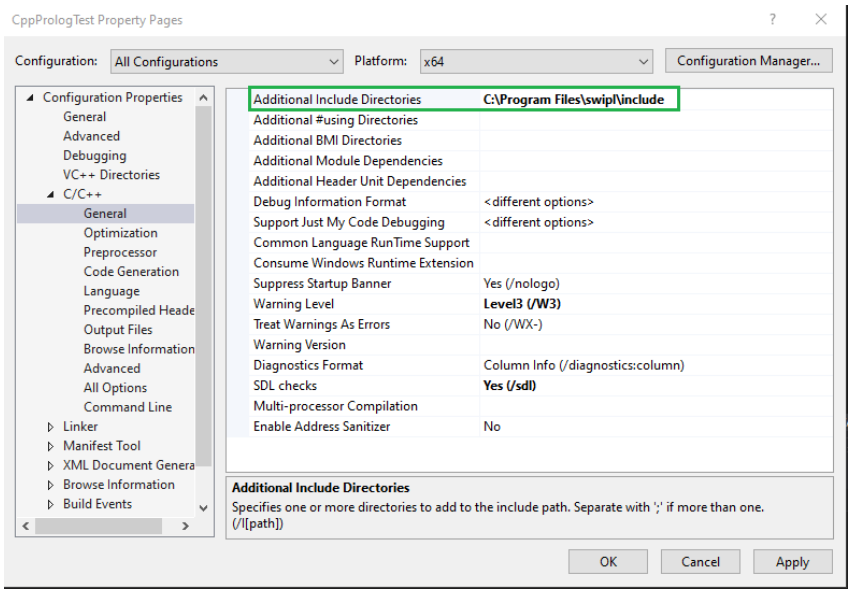
Εν συνεχεία, φτιάχνουμε ένα νέο αρχείο *C++*, για παράδειγμα *TestProlog.cpp*, είτε με δεξί κλίκ πάνω στο *Project* και *Add* ⇒ *New Item*, είτε πατώντας *Ctrl+Shift+A* και ελέγχουμε ώστε το *Project* είναι ρυθμισμένο να παράγει 64-bit προγράμματα.

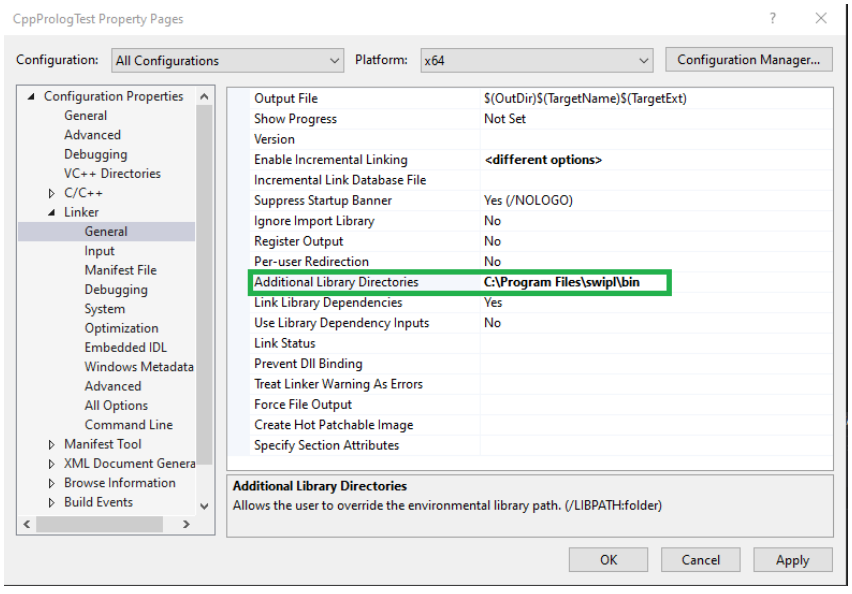


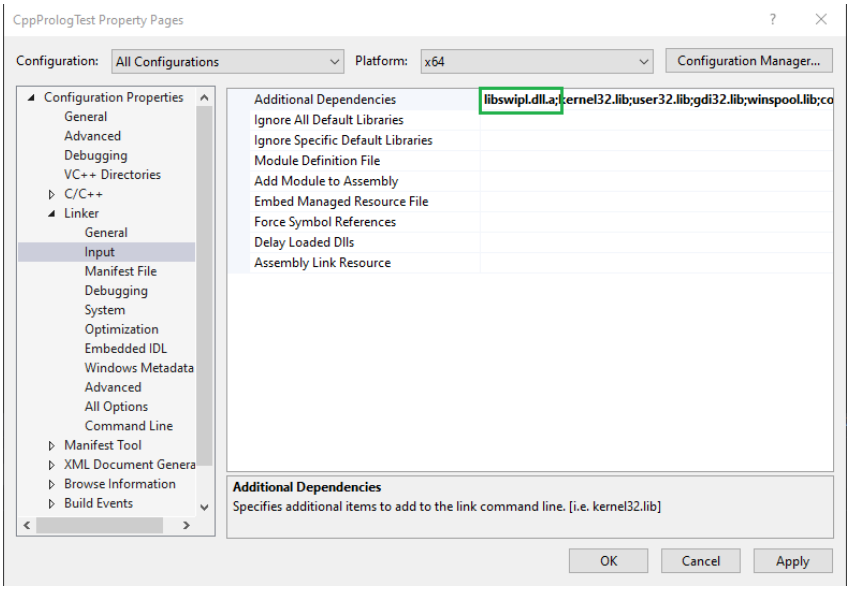


Έπειτα πρέπει να μεταβούμε στις ιδιότητες του *Project* με δεξί κλικ στο *Project* και επιλέγοντας *Properties*. Εκεί, στην καρτέλα *Configuration Properties* ⇒ *General* επιλέγουμε ως *Configuration Type* το *Dynamic Library(.dll)* και στην επιλογή *C++ Language Standard* μπορούμε να δηλώσουμε την έκδοση της *C++* που επιθυμούμε. Μετά, στην καρτέλα *C/C++* ⇒ *General* θα προσθέσουμε στο πεδίο *Additional Include Directories* το μονοπάτι στο οποίο βρίσκεται το αρχείο κεφαλλίδας που μας ενδιαφέρει *(SWI-Prolog.h)*, δηλαδή το *C:\Program Files\swipl\include*. Έπειτα θα πρέπει να ρυθμίσουμε τον *Linker*. Στην καρτέλα *Linker* ⇒ *General* θα προσθέσουμε στο πεδίο *additional Library Directories* το μονοπάτι που βρίσκεται η βιβλιοθήκη που θα χρησιμοποιήσουμε *(libswipl.dll.a)*, δηλαδή το *C:\Program Files\swipl\bin*. Τέλος, στην καρτέλα *Linker* ⇒ *Input* θα προσθέσουμε στο πεδίο *Additional Dependencies* το όνομα της βιβλιοθήκης, δηλαδή το *libswipl.dll.a*.


CppPrologTest Property Pages
Configuration: All Configurations
Platform: x64
Configuration Manager...
Configuration Properties
General
Advanced
Debugging
VC++ Directories
C/C++
Linker
Manifest Tool
XML Document Generator
Browse Information
Build Events
Custom Build Step
Code Analysis
General Properties
Output Directory $(SolutionDir)$(Platform)\$(Configuration)\
Intermediate Directory $(Platform)\$(Configuration)\
Target Name $(ProjectName)
Configuration Type Dynamic Library (.dll)
Windows SDK Version 10.0 (latest installed version)
Platform Toolset Visual Studio 2019 (v142)
C++ Language Standard ISO C++17 Standard (/std:c++17)
C Language Standard Default (Legacy MSVC)
C++ Language Standard
Determines the C++ language standard the compiler will enforce. It is recommended to use the latest version when possible. (/std:c++14, /std:c++17, /std:c++latest)
OK
Cancel
Apply


CppPrologTest Property Pages
Configuration: All Configurations
Platform: x64
Configuration Manager...
Configuration Properties
General
Advanced
Debugging
VC++ Directories
C/C++
General
Optimization
Preprocessor
Code Generation
Language
Precompiled Heade
Output Files
Browse Information
Advanced
All Options
Command Line
Linker
Manifest Tool
XML Document Genera
Browse Information
Build Events
Additional Include Directories C:\Program Files\swipl\include
Additional #using Directories
Additional BMI Directories
Additional Module Dependencies
Additional Header Unit Dependencies
Debug Information Format <different options>
Support Just My Code Debugging <different options>
Common Language RunTime Support
Consume Windows Runtime Extension
Suppress Startup Banner Yes (/nologo)
Warning Level Level3 (/W3)
Treat Warnings As Errors No (/WX-)
Warning Version
Diagnostics Format Column Info (/diagnostics:column)
SDL checks Yes (/sdl)
Multi-processor Compilation
Enable Address Sanitizer No
Additional Include Directories
Specifies one or more directories to add to the include path. Separate with ';' if more than one. (/I[path])
OK
Cancel
Apply


CppPrologTest Property Pages
Configuration: All Configurations
Platform: x64
Configuration Manager...
Configuration Properties
General
Advanced
Debugging
VC++ Directories
C/C++
Linker
General
Input
Manifest File
Debugging
System
Optimization
Embedded IDL
Windows Metadata
Advanced
All Options
Command Line
Manifest Tool
XML Document Genera
Browse Information
Build Events
Output File $(OutDir)$(TargetName)$(TargetExt)
Show Progress Not Set
Version
Enable Incremental Linking <different options>
Incremental Link Database File
Suppress Startup Banner Yes (/NOLOGO)
Ignore Import Library No
Register Output No
Per-user Redirection No
Additional Library Directories C:\Program Files\swipl\bin
Link Library Dependencies Yes
Use Library Dependency Inputs No
Link Status
Prevent Dll Binding
Treat Linker Warning As Errors
Force File Output
Create Hot Patchable Image
Specify Section Attributes
Additional Library Directories
Allows the user to override the environmental library path. (/LIBPATH:folder)
OK
Cancel
Apply



Εφόσον οι ρυθμίσεις του *Project* είναι έτοιμες, ας δούμε μερικά απλά παραδείγματα χρήσης της *Prolog* μέσα από τη C++. Στα παραδείγματα θα χρησιμοποιηθούν ακριβώς οι ίδιες ρυθμίσεις με τη διαφορα ότι αντί για βιβλιοθήκη επιλέχθηκε εκτελέσιμο κονσόλας ώστε να δείχνουμε εύκολα τα αποτελέσματα των παραδειγμάτων. Τα παραδείγματα αυτά θα βοηθήσουν στην κατανόηση του τελικού κώδικα της *C++*. Στην παρούσα εργασία θα γίνει μια μικρή αναφορά στα εργαλεία διεπαφής, καθώς ανάλυση θα γίνει μόνο σε αυτά που χρειαζόμαστε στο παρόν πρόβλημα. Αν κάποιος επιθυμεί να μάθει περισσότερες πληροφορίες, λειτουργείες και λεπτομέρειες μπορεί όπως προαναφέρθηκε να επισκεφτεί το εγχειρίδιο της *swi-prolog* που είναι δωρεάν και διαθέσιμο στο διαδίκτυο.

Ας αναπτύξουμε λοιπόν ένα απλό πρόγραμμα κονσόλας το οποίο θα παίρνει ένα αρχείο κώδικα *Prolog* που θα περιέχει ένα απλό κατηγόρημα και στη συνέχεια θα καλούμε το κατηγόρημα και θα τυπώνουμε το αποτέλεσμα. Για παράδειγμα έστω ότι έχουμε στο αρχείο της *Prolog* ένα απλό κατηγόρημα που προσθέτει δύο αριθμούς και το όνομα του αρχείου είναι *mylib.pl*.

```
myAdd(X,Y,Result) :- Result is X + Y.
```

Κατά αρχάς, θα πρέπει να ξεκινήσουμε το runtime της *Prolog*. Αυτό γίνεται με χρήση της συνάρτησης *PL_initialise* που κατά μία έννοια είναι το αντίστοιχο της εκτέλεσης του διερμηνέα της *Prolog*, αλλά προγραμματιστικά. Χρειαζόμαστε δύο παραμέτρους για αυτή τη συνάρτηση. Η δεύτερη είναι ένας πίνακας συμβολοσειρών που αναφέρεται στις παραμέτρους *(Flags)* με τα οποία θα ξεκινήσει ο διερμηνέας της *Prolog* και η πρώτη ο αριθμός των παραμέτρων που επιλέξαμε. Η πρώτη συμβολοσειρά αυτού του πίνακα είναι ιδιαίτερα σημαντική αφού αφορά την εύρεση του runtime της *Prolog*. Αν το αφήσουμε κενό, πηγαίνει προκαθορισμένα και ψάχνει αν είναι φορτωμένη η βιβλιοθήκη *libswipl.dll* στην παρούσα εκτέλεση. Εμείς εκ των προτέρων γνωρίζουμε ότι έχουμε φορτώσει τη συγκεκριμένη βιβλιοθήκη, άρα θα το αφήσουμε κενό. Σαν δεύτερη παράμετρο θα προσθέσουμε το *Flag -q*

που σημαίνει ότι θέλουμε ο διερμηνέας της *Prolog* να ακολουθήσει «ήσυχη» εκτέλεση. Ο παρακάτω κώδικας δείχνει ένα παράδειγμα έναρξης του διερμηνέα της *Prolog* από τη *(C++)*:

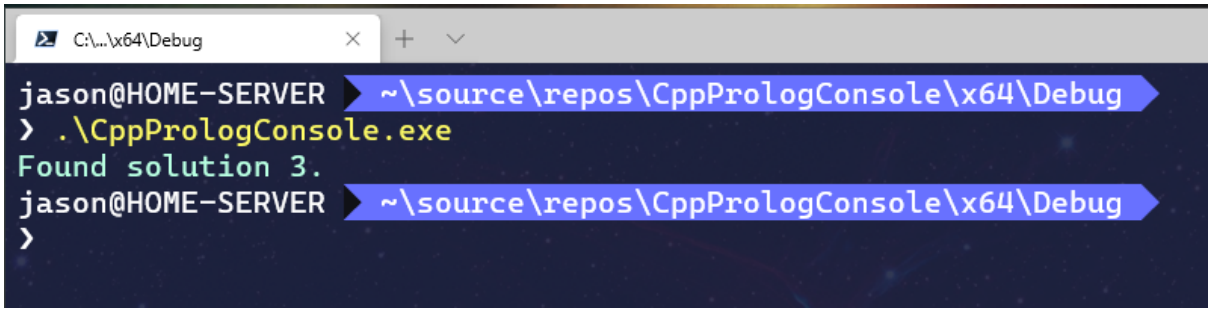
```
#include <iostream>
#include <SWI-Prolog.h>

int main()
{
  char* plav[2];
  plav[0] = (char*)"";
  plav[1] = (char*)("-q");

  if (!PL_initialise(2, plav))
  {
    printf("Could not communicate with prolog runtime");
  }
}
```

Στη συνέχεια θα δημιουργήσουμε ένα pointer *(predicate_t)* που θα δείχνει στο κατηγόρημα *consult*, το οποιο και θα χρησιμοποιήσουμε για να φορτώσουμε το αρχείο μας *mylib.pl*. Ως παράμετρο στο *consult* θα περάσουμε έναν ειδικό pointer πάνω στο αρχείο με τη χρήση του τύπου *term_t* που είναι μια αφαιρετική γεικότερη προσέγγιση των όρων της *Prolog*. Τέλος, καλείται η *Prolog* να τρέξει την *consult* με τις παραμέτρους που ορίσαμε.

```
#include <iostream>
#include <SWI-Prolog.h>

int main()
{
  char* plav[2];
  plav[0] = (char*)"";
  plav[1] = (char*)("-q");

  if (!PL_initialise(2, plav))
  {
    printf("Could not communicate with prolog runtime");
  }

  predicate_t p_consult = PL_predicate("consult", 1, "database");
  term_t t_file = PL_new_term_ref();
  PL_put_string_chars(t_file, "mylib.pl");
  PL_call_predicate(NULL, 0, p_consult, t_file);

}
```

Έπειτα θα πρέπει να καλέσουμε από το αρχείο που μόλις φορτώθηκε το κατηγόρημα *myAdd* που υλοποιήθηκε παραπάνω. Αυτό θα γίνει δημιουργώντας παλι ένα pointer *predicate_t*, αυτή τη φορά δείχνοντας στο κατηγόρημα *myAdd*. Αξίζει εδώ να σημειωθεί ότι για τη δημιουργία αυτών των δεικτών εκτός από το όνομα του κατηγορήματος είναι απαραίτητος και ο αριθμός των παραμέτρων που δέχονται *(arrity)* καθως και το *module* στο οποίο βρίσκονται. Για να περάσουμε τις τρεις παραμέτρους που χρειάζεται η *myAdd* θα χρησιμοποιήσουμε και πάλι τον τύπο *term_t* που θα ενθυλακώσει και τις τρεις δίνοντάς μας πρόσβαση σε κάθε μία με χρήση αριθμητικής δεικτών. Όπως φαίνεται και παρακάτω επιλέγουμε ώς πρώτη παράμετρο τον αριθμό 1, ως δεύτερη τον αριθμό 2 και δηλώνουμε ότι η τρίτη παράμετρος είναι μεταβλητή, ακριβώς με το ίδιο δηλαδή σκεπτικό που υλοποιήθηκε η *myAdd*. Στη συνέχεια φτιάχνουμε έναν ειδικό pointer που δείχνει στο ερώτημα που θα στείλουμε στην *Prolog* και ζητάμε το αποτέλεσμα. Φυσικά η *Prolog* αποκρίνεται μόνο λογικά με «αλήθεια» ή «ψέμα». Άρα ελέγχουμε πρώτα το αποτέλεσμα και αν είναι αληθές χρησιμοποιούμε την κατάλληλη συνάρτηση για να εξάγουμε το αποτέλεσμα της πρόσθεσης από την τρίτη παράμετρο. Συνολικά ο κώδικας και η εκτέλεση παρουσιάζονται παρακάτω:

```
#include <iostream>
#include <SWI-Prolog.h>

int main()
{
  char* plav[2];
  plav[0] = (char*)"";
  plav[1] = (char*)("-q");

  if (!PL_initialise(2, plav)) {
    printf("Could not communicate with prolog runtime");
  }

  predicate_t p_consult = PL_predicate("consult", 1, "database");
  term_t t_file = PL_new_term_ref();
  PL_put_string_chars(t_file, "mylib.pl");
  PL_call_predicate(NULL, 0, p_consult, t_file);

  predicate_t p_myAdd = PL_predicate("myAdd", 3, "user");
  term_t t = PL_new_term_refs(3);
  PL_put_integer(t, 1);
  PL_put_integer(t + 1, 2);
  PL_put_variable(t + 2);
  qid_t query = PL_open_query(NULL, PL_Q_NORMAL, p_myAdd, t);

  int result = PL_next_solution(query);
  if(result) {
    int x;
    PL_get_integer(t + 2, &x);
    printf("Found solution %d.\n", x);
  }
  PL_close_query(query);
}
```



Πολύ σημαντικό, ειδικά για το παρόν πρόβλημα, είναι ο χειρισμός των λιστών της *Prolog* μέσα από τη *C++* καθώς το κατηγόρημα το οποίο υλοποιήθηκε στην προηγούμενη ενότητα για την επιστροφή της λύσης του λαβυρίνθου, γυρνάει το αποτέλεσμα στην παράμετρο *Path* η οποία είναι μια λίστα από λίστες σημείων, δηλαδή λίστα λιστών ακεραίων.

Η δημιουργία και η προσπέλαση μιας λίστας ακολουθεί κατά πολύ τη λογική της *Prolog* ακόμα και μέσα από τη *C++*. Σε κάθε λειτουργία αποδομούμε τη λίστα στο πρώτο της στοιχείο *Head* και την υπόλοιπη *Tail*. Έστω για παράδειγμα ότι το αρχείο *mylib.pl* περιέχει την υλοποίηση της *bubblesort* η οποία δέχεται 2 παραμέτρους. Η πρώτη είναι μια τυχαία λίστα ακεραίων και στη δεύτερη μας επιστρέφει αυτή τη λίστα ταξινομημένη.

```
swap([X,Y|List],[Y,X|List]) :- X > Y.
swap([Z|List],[Z|List1]) :- swap(List,List1).

bubblesort(InputList,SortedList) :-
    swap(InputList,List) , ! ,
    bubblesort(List,SortedList).
bubblesort(SortedList,SortedList).
```

Παρακάτω παρουσιάζεται κώδικας σε *C++* που καλέι την *bubblesort* ακολουθώντας παρόμοια διαδικασία με αυτή που αναφέρθηκε. Η σημαντική διαφορά που πρέπει να τονιστεί είναι ότι και κατά τη δημιουργία της λίστας που θα στείλουμε ως είσοδο στην *Prolog*, αλλά και κατά την προσπέλαση της τελικής ταξινομημένης λίστας που τυπώνεται στην κονσόλα, το σημείο αναφοράς είναι πάντα το πρώτο στοιχείο της λίστας.

```
#include <iostream>
#include <SWI-Prolog.h>

int main()
{
  char* plav[2];
  plav[0] = (char*)"";
  plav[1] = (char*)("-q");

  if (!PL_initialise(2, plav))
```

```
  {
    printf("Could not communicate with prolog runtime");
  }

  predicate_t p_consult = PL_predicate("consult", 1, "database");
  term_t t_file = PL_new_term_ref();
  PL_put_string_chars(t_file, "mylib.pl");
  PL_call_predicate(NULL, 0, p_consult, t_file);

  int arr[10] = { 2, 4, 1, 3, 9, 7, 5, 6, 10, 8 };

  predicate_t p_sort = PL_predicate("bubblesort", 2, "user");
  term_t t = PL_new_term_refs(2);
  term_t input_list = t;
  term_t output_list = t + 1;

  term_t head = PL_new_term_ref();
  PL_put_nil(input_list);

  for (int i = 0; i < 10; i++)
  {
    PL_put_integer(head, arr[i]);
    PL_cons_list(input_list, head, input_list);
  }

  qid_t query = PL_open_query(nullptr, PL_Q_NORMAL, p_sort, t);

  int result = PL_next_solution(query);

  if (result) {
    term_t tail = PL_copy_term_ref(output_list);
    head = PL_new_term_ref();
    int x;
    while (PL_get_list(tail, head, tail)) {
      PL_get_integer(head, &x);
      printf("%d ", x);
    }
  }
  PL_close_query(query);
}
```

```
C:\...\x64\Debug
jason@HOME-SERVER ~\source\repos\CppPrologConsole\x64\Debug
❯ .\CppPrologConsole.exe
1 2 3 4 5 6 7 8 9 10
jason@HOME-SERVER ~\source\repos\CppPrologConsole\x64\Debug
❯
```

Όσον αφορά το παρόν πρόβλημα, η λογική είναι αρκετά παρόμοια με τα παραπάνω παραδείγματα. Θα πρέπει όμως να δωθεί μεγάλη προσοχή στο γεγονός ότι εδώ η λίστα που επιστρέφεται με το αποτέλεσμα περιέχει και η ίδια λίστες από δύο αριθμούς η κάθε μια. Επίσης, μπορούμε να έχουμε παραπάνω απο μία λύσεις, άρα αντί να πάρουμε κατευθείαν το πρώτο αποτέλεσμα του ερωτήματος που θα σταλει στην *Prolog* θα πρέπει να προσπαθήσουμε να προσπελάσουμε όλες τις λύσεις. Επομένως, όπως θα φανεί και στον κώδικα, χρειαζόμαστε ένα κομμάτι κώδικα με μια επαναληπτική διαδικασία για κάθε λύση και δυο εμφωλευμένες επαναληπτικές διαδικασίες που αντιστοιχούν στην προσπέλαση της διπλά εμφωλευμένης λίστας. Έπειτα θα πρέπει να προσέξουμε πολύ τους τύπους δεδομένων των παραμέτρων συνάρτησης που θα εκθέσουμε στη *C#*, ώστε να είναι εύκολα συμβατοί. Ο πιο απλός τρόπος είναι οι απλοί τύποι, όπως αριθμοί και συμβολοσειρές.

Πιο συγκεκριμένα η συνάρτηση που υλοποιήθηκε ονομάστηκε *Solve* και επιστρέφει μια λογική τιμή. Είναι *true* αν η *PL_initialise* εκτελέστηκε επιτυχώς και αντίστοιχα *false* αν δεν μπόρεσε να ξεκινήσει η μηχανή της *Prolog*. Άρα αυτή η τιμή έχει απλά μια επικυρωτική σημασία σχετικά με το αν υπήρξε ή όχι επικοινωνία με την *Prolog*. Αν ο καταναλωτής της συνάρτησης έχει μια τέτοια πληροφορία είναι εύκολο σε περίπτωση αποτυχίας να τυπώσει στο χρήστη κάτι σχετικό, όπως για παράδειγμα ότι απέτυχε η επικοινωνία με την *Prolog* και να δοκιμάσει να ελέγξει αν στο *Path* υπάρχει σωστά το μονοπάτι για το εκτελέσιμο της *Prolog*. Η συνάρτηση δέχεται δύο παραμέτρους. Η πρώτη είναι το μονοπάτι στο αρχείο του κώδικα *Prolog* που δημιουργήθηκε δυναμικά και το δεύτερο μια συμβολοσειρά που σκοπό έχει να επιστρέψει το τελικό αποτέλεσμα.

Πολύ σημαντικό θέμα είναι η μορφή που θα έχει η λύση στην παράμετρο τύπου συμβολοσειράς. Εδώ καλούμαστε να υλοποιήσουμε έναν απλό τρόπο να κωδικοποιηθούν οι λύσεις που κάθε μία είναι μια λίστα από λίστες. Ο τρόπος που επιλέχθηκε είναι η συμβολοσειρά να περιέχει συγκεκριμένους χαρακτήρες σήμανσης για να ξεχωρίζουμε τα σημεία, την κατέθυνση και τις διαφορετικές λύσεις. Πιο αναλυτικά, κάθε σημείο θα σημαίνεται εντός παρενθέσεων και με κόμμα. Ανάμεσα σε κάθε σημείο κοινής λύσης θα υπάρχει ο χαρακτήρας $>$ που δείχνει την κατέθυνση και κάθε λύση θα χωρίζεται με την άλλη με τον χαρακτήρα $|$. Θα μπορούσε φυσικά να χρησιμοποιηθεί κάποια βιβλιοθήκη με βοηθητικές συναρτήσης για *serialization* σε κάποια από τις δημοφιλείς *markup* γλώσσες όπως *XML* ή *Json*. Επιλέγουμε μια πιο αφελή προσέγγιση για λόγους απλότητας αλλά και ευκρίνειας αφού η κωδικοποίηση που επιλέχτηκε είναι πολύ ξεκάθαρη. Αν για παράδειγμα χρησιμοποιήσουμε την υλοποίηση που φαίνεται στον κώδικα παρακάτω για τη λύση του λαβυρίνθου της τρίτης ενότητας της παρούσας εργασίας θα καταλήξουμε με την εξής συμβολοσειρά:

(0,4)>(1,4)>(1,3)>(1,2)>(1,1)>(2,1)>(2,0)>(3,0)>(4,0)>(4,1)>(5,1)|(0,4)>(1,4)>
(2,4)>(3,4)>(4,4)>(5,4)>(5,3)>(5,2)>(5,1)

Παρακάτω παρουσιάζεται ο τελικός κώδικας της βιβλιοθήκης σε *C++* που περιλαμβάνει την συνάρτηση η οποία θα καταναλωθεί από το εκτελέσιμο πρόγραμμα κονσόλας της *C#*:

```cpp
#include <SWI-Prolog.h>
#include <vector>
#include <string>
#include <numeric>

#define PATHFINDER_SOLVER _declspec(dllexport)
#define MAX_BUFFER_SIZE 1024

extern "C" {
  PATHFINDER_SOLVER bool Solve(const char* prologFilePath, char* solutions)
  {
    char* plav[2];
    plav[0] = (char*)"";
    plav[1] = (char*)("-q");

    if (!PL_initialise(2, plav))
    {
      return false;
    }

    std::vector<std::string> vSolutions;

    predicate_t p_consult = PL_predicate("consult", 1, "database");
    term_t t = PL_new_term_ref();
    PL_put_string_chars(t, prologFilePath);
    PL_call_predicate(nullptr, 0, p_consult, t);

    predicate_t p_solve = PL_predicate("solve", 1, "user");
    term_t list = PL_new_term_ref();
    PL_put_variable(list);

    qid_t query = PL_open_query(nullptr, PL_Q_NORMAL, p_solve, list);

    int coordinate;
    while (PL_next_solution(query))
    {
      term_t tail = PL_copy_term_ref(list);
      term_t head = PL_new_term_ref();
      while (PL_get_list(tail, head, tail))
      {
        term_t nestedHead = PL_new_term_ref();
        term_t nestedTail = PL_copy_term_ref(head);

        bool isXCoordinate = true;
        while (PL_get_list(nestedTail, nestedHead, nestedTail))
        {
          if (isXCoordinate)
          {
            PL_get_integer(nestedHead, &coordinate);
            vSolutions.push_back("(" + std::to_string(coordinate) + ",");
            isXCoordinate = false;
          }
          else
          {
            PL_get_integer(nestedHead, &coordinate);
```

```
            vSolutions.push_back(std::to_string(coordinate) + ")>");
            isXCoordinate = true;
          }
        }
      }

      vSolutions.push_back("|");
    }

    if (!vSolutions.empty())
    {
      std::string sSolutions = std::accumulate(vSolutions.begin(),
    vSolutions.end(), std::string(""));

      for (std::string::size_type i = 0; i < sSolutions.size(); i++) {
        solutions[i] = sSolutions[i];
      }
      for (std::string::size_type i = sSolutions.size(); i <
    MAX_BUFFER_SIZE; i++)
      {
        solutions[i] = '\0';
      }
    }

    PL_close_query(query);

    return true;
  }
}
```

5 Πρόγραμμα διεπαφής με το χρήστη σε C#

Η πρώτη ενέργεια πριν τον προγραμματισμό είναι να αναλυθεί και να είναι ξεκάθαρη η λειτουργικότητα που θέλουμε να υλοποιήσουμε. Ως προς το πρόγραμμα διεπαφής με το χρήστη θεωρήθηκαν οι παρακάτω απαιτήσεις:

- Ο χρήστης θα μπορεί να επιλέξει τις διαστάσεις του λαβυρίνθου, τα σημεία αρχής, τέλους και τα σημεία που υπάρχει τοίχος. Εν ολίγοις, θα μπορεί να χτίσει το δικό του λαβύρινθο.
- Θα πρέπει να δίνεται από το πρόγραμμα σύντομη και ακριβής επεξήγηση των δεδομένων που πρέπει να εισαχθούν σε κάθε βήμα και γενικότερα να υπάρχει μια ξεκάθαρη βηματική πορεία κατά την εκτέλεση.
- Πρέπει το πρόγραμμα να διασφαλίζει την διόρθωση των λαθών του χρήστη, δίνοντας του την επιλογή να ξαναεισάγει κάποιο δεδομένο σε περίπτωση λάθους ή να τερματίσει το πρόγραμμα.
- Θα παρέχει κατάλληλη σχηματική απεικόνιση του λαβυρίνθου τόσο κατά τη διάρκεια εισαγωγής των δεδομένων όσο και για την τελική παρουσίαση των αποτελέσμάτων.

Όπως προαναφέρθηκε, για τα παραπάνω υλοποιείται πρόγραμμα κονσόλας σε γλώσσα *C#*. Η γλώσσα *C#*, αν και πολυπαραδειγματική, είναι εν πρώτοις αντικειμενοστρεφής, χαρακτηριστικό που θα μας βοηθήσει ιδιαίτερα στο σχεδιασμό του συγκεκριμένου προβλήματος. Το πρόβλημα αποδομείται με βάση τις οντότητες που δρουν σε αυτο και η τελική σύνθεσή της δράσης αυτής θα παράγει τα ζητούμενα αποτελέσματα. Παρακάτω, λοιπόν, θα παρουσιαστούν και θα αναλυθούν όλες οι κλάσεις που σχεδιάστηκαν και τελικά το κύριο πρόγραμμα στο οποίο γίνεται κατάλληλη σύνθεση της λειτουργίας των αντικειμένων.

- *struct Point:* Είναι προφανής η ανάγκη χρήσης της έννοιας ενός σημείου πάνω στο επίπεδο. Το *struct Point*, λοιπόν, παριστάνει ένα ζεύγος ακεραίων (X, Y) που κάθε ένα αντιστοιχεί σε σημείο στο επίπεδο του λαβυρίνθου. Καλό είναι ο χρήστης να δίνει τα σημεία στην κονσόλα σε μια γνωστή μορφή όπως η κλασική μαθηματική αναπαράσταση εντός παρενθέσεων και με κόμμα. Οπότε, υλοποιείται στα πλαίσια αυτού του *struct* μια μέθοδος *TryParse* η οποία έχει ως σκοπό να μετατρέπει μια συμβολοσειρά δύο ακεραίων X και Y που έχουν τη μορφή (X, Y) σε *struct Point*. Αυτό επιτυγχάνεται πολύ εύκολα με τη χρήση της κανονικής έκφρασης $\hat{}\backslash(\backslash d,\backslash d\backslash)\$$. Αντίστοιχα, υλοποιήθηκε και η αντίστροφη και πολύ χρήσιμη λειτουργία της μετατροπής ενός *struct Point* σε συμβολοσειρά με μορφή (X, Y). Αυτό έγινε με override της built-in μεθόδου *ToString*. Τέλος, υλοποιήθηκαν όλες οι κατάλληλες λειτουργίες και τελεστές που αφορούν την ισότητα ή την ανισότητα δύο σημείων.

```
1  using System.Text.RegularExpressions;
2  using System;
3
4  namespace PathFinderConsole
5  {
6      struct Point : IEquatable<Point>
7      {
8          public int X { get; set; }
```

```
9         public int Y { get; set; }
10
11        public static bool TryParse(string strPoint, out Point point)
12        {
13            point = new Point();
14            var result = Regex.Match(strPoint, @"^\(\d,\d\)$");
15
16            if (result.Success)
17            {
18                point.X = Convert.ToInt32(strPoint.TrimStart('(').Split(',')[0]);
19                point.Y = Convert.ToInt32(strPoint.TrimEnd(')').Split(',')[1]);
20                return true;
21            }
22
23            return false;
24        }
25
26        public bool Equals(Point other)
27        {
28            if (X == other.X && Y == other.Y)
29                return true;
30
31            return false;
32        }
33
34        public override string ToString()
35        {
36            return $"({X},{Y})";
37        }
38
39        public override bool Equals(object obj)
40        {
41            if (TryParse(obj.ToString(), out Point point))
42                if (Equals(point))
43                    return true;
44
45            return false;
46        }
47
48        public static bool operator ==(Point point1, Point point2)
49        {
50            if (point1.Equals(point2))
51                return true;
52
53            return false;
54        }
55
56        public static bool operator !=(Point point1, Point point2)
```

```
        {
            if (point1.Equals(point2))
                return false;

            return true;
        }

        public override int GetHashCode()
        {
            return ToString().GetHashCode();
        }
    }
}
```

- *class PrologCodeGenerator:* Ευθύνη της κλάσης *PrologCodeGenerator* είναι να δημιουργήσει δυναμικά με βάση τις επιλογές του χρήστη τον κώδικα της *Prolog*. Πιο συγκεκριμένα αυτή η κλάση δημιουργεί δυναμικά ένα αρχείο με το όνομα *pathfinder.pl* το οποίο έχει τη μορφή του παραδείγματος της τρίτης ενότητας με τη διαφορά ότι τα *facts* είναι δυναμικά. Οι παράμετροι που δέχεται ένα αντικείμενο τύπου *PrologCodeGenerator* κατά την κατασκευή του είναι κατά συνέπεια οι διαστάσεις του λαβυρίνθου, η αρχή και το τέλος, καθώς και τα σημεία που υπάρχει τοίχος. Η διαδικασία παραγωγής του κώδικα συντελείται στη μέθοδο *Generate* η οποία αφού ετοιμάσει τον κώδικα, τον γράφει στο αρχείο και γυρνά ολόκληρο το μονοπάτι προς το αρχείο ώστε να σταλεί ως παράμετρος στη συνάρτηση της *C++*.

```
using System.Collections.Generic;
using System.Text;
using System.IO;
using System;

namespace PathFinderConsole
{
    class PrologCodeGenerator
    {
        private readonly int _boundX;
        private readonly int _boundY;
        private readonly Point _startPoint;
        private readonly Point _endPoint;
        private readonly List<Point> _wallPoints;

        public PrologCodeGenerator(int boundX, int boundY, Point startPoint, Point endPoint, List<Po
        {
            _boundX = boundX;
            _boundY = boundY;
            _startPoint = startPoint;
            _endPoint = endPoint;
            _wallPoints = wallPoints;
```

```
23          }
24
25          public string Generate()
26          {
27              StringBuilder sb = new StringBuilder();
28
29              // Make Facts from User input
30              sb.AppendLine($"boundX({_boundX}).");
31              sb.AppendLine($"boundY({_boundY}).");
32              sb.AppendLine();
33              sb.AppendLine($"start({_startPoint.X},{_startPoint.Y}).");
34              sb.AppendLine($"end({_endPoint.X},{_endPoint.Y}).");
35              sb.AppendLine();
36              foreach (var point in _wallPoints)
37              {
38                  sb.AppendLine($"wallPoint({point.X},{point.Y}).");
39              }
40              sb.AppendLine();
41
42              // Make Rules
43              sb.AppendLine("isInBounds(X,Y) :- boundX(X_DIM),\n" +
44                            "                   boundY(Y_DIM),\n" +
45                            "                   X < X_DIM,\n" +
46                            "                   Y < Y_DIM,\n" +
47                            "                   X >= 0,\n" +
48                            "                   Y >= 0.\n");
49              sb.AppendLine();
50              sb.AppendLine("isFreePoint(X,Y) :- isInBounds(X,Y),\n" +
51                            "                   \\+start(X,Y),\n" +
52                            "                   \\+wallPoint(X,Y).");
53              sb.AppendLine();
54              sb.AppendLine("goNorth(X,Y,X1,Y1) :- X1 is X,\n" +
55                            "                      Y1 is Y + 1,\n" +
56                            "                      isFreePoint(X1,Y1).");
57              sb.AppendLine("goSouth(X,Y,X1,Y1) :- X1 is X,\n" +
58                            "                      Y1 is Y - 1,\n" +
59                            "                      isFreePoint(X1,Y1).");
60              sb.AppendLine("goEast(X,Y,X1,Y1) :- X1 is X + 1,\n" +
61                            "                     Y1 is Y,\n" +
62                            "                     isFreePoint(X1,Y1).");
63              sb.AppendLine("goWest(X,Y,X1,Y1) :- X1 is X - 1,\n" +
64                            "                     Y1 is Y,\n" +
65                            "                     isFreePoint(X1,Y1).");
66              sb.AppendLine();
67              sb.AppendLine("goSearch(X,Y,X1,Y1) :- goNorth(X,Y,X1,Y1);\n" +
68                            "                       goSouth(X,Y,X1,Y1);\n" +
69                            "                       goEast(X,Y,X1,Y1);\n" +
70                            "                       goWest(X,Y,X1,Y1).");
```

```
            sb.AppendLine();
            sb.AppendLine("solve(Path) :- start(X,Y),\n" +
                          "               solve(X,Y,[],Path).");
            sb.AppendLine("solve(X,Y,PathHistory,Path) :- end(X,Y),\n" +
                          "                               PathHistory = Path.");
            sb.AppendLine("solve(X,Y,PathHistory, Path) :-");
            sb.AppendLine("    goSearch(X,Y,X1,Y1),");
            sb.AppendLine("    \\+member([X1, Y1], PathHistory),");
            sb.AppendLine("    append(PathHistory,[[X1,Y1]],NewPathHistory),");
            sb.AppendLine("    solve(X1,Y1,NewPathHistory,Path).");

            using (FileStream fs = File.Create("pathfinder.pl"))
            {
                byte[] bytes = new UTF8Encoding(true).GetBytes(sb.ToString());
                fs.Write(bytes, 0, bytes.Length);
            }

            return Path.Combine(AppContext.BaseDirectory, "pathfinder.pl");
        }
    }
}
```

- *class SolutionParser:* Η κλάση αυτή ευθύνεται για την με μετατροπή της λύσης που θα επιστραφεί από τη συνάρτηση της *C++* σε λίστες από σημεία, δηλαδή σε λίστες απο *Point*. Όπως αναφέρθηκε στην προηγούμενη ενότητα η *C++* θα επιστρέψει τις λύσεις σε μια συμβολοσειρά κωδικοποιημένη με συγκεκριμένο τρόπο. Κάθε σημείο θα είναι εντός παρενθέσεων και με κόμμα, κάθε σημείο χωρίζεται με το επόμενο με το χαρακτήρα > και κάθε λύση χωρίζεται με τις υπόλοιπες με χρήση του χαρακτήρα |. Η συγκεκριμένη κλάση μέσω της μεθόδου *GetSolutions* διαχωρίζει τη συμβολοσειρά που περιέχει τη λύση με βάση αυτούς τους χαρακτήρες και επιστρέφει μια λίστα από λίστες απο σημεία, γιατί προφανώς κάθε λύση είναι μια αλληλουχία σημείων.

```
using System.Collections.Generic;
using System;

namespace PathFinderConsole
{
    class SolutionsParser
    {
        private readonly string _solutions;

        public SolutionsParser(string solutions)
        {
            _solutions = solutions;
        }

        public List<List<Point>> GetSolutions()
```

```
        {
            var res = new List<List<Point>>();

            foreach (var solution in _solutions.Split(new string[] { "|" },
                                        StringSplitOptions.RemoveEmptyEntries))
            {
                var resItem = new List<Point>();

                foreach (var strPoint in solution.Split('>'))
                {
                    if (Point.TryParse(strPoint, out Point point))
                    {
                        resItem.Add(point);
                    }
                }

                res.Add(resItem);
            }

            return res;
        }
    }
}
```

- *class ConsoleUtils*: Όπως προϊδεάζει και το όνομα, αυτή η κλάση παρέχει όλες τις βοηθητικές λειτουργίες που θα χρειαστούμε σε σχέση με την κονσόλα. Έχει μηχανισμούς σε μορφή στατικών μεθόδων για την σωστή εξαγωγή και επικύρωση δεδομένων με βάση τις εισαγωγές του χρήστη καθώς και την ευθύνη για σχηματική απεικόνιση του λαβυρίνθου. Πιο συγκεκριμένα, έχει μεθόδους για την εξαγωγή θετικών ακεραίων (διαστάσεις λαβυρύνθου) και εξαγωγή σημείων είτε μεμονομένα για την αρχή και το τέλος, είτε μαζικά για την εισαγωγή των σημείων τοίχου. Σε κάθε πιθανό λάθος του χρήστη επιστρέφεται μήνυμα σχετικό με τη φύση του σφάλματος και δίνεται η ευκαιρία επανεισαγωγής των αντίστοιχων δεδομένων.

```
using System;
using System.Collections.Generic;
using System.Text;

namespace PathFinderConsole
{
    static class ConsoleUtils
    {

        public static bool TryGetIntegerWithFilter(string message,
                                                   Predicate<int> filter,
                                                   string filterErrorMessage,
                                                   out int number)
```

```
14          {
15              while (true)
16              {
17                  Console.Write(message);
18                  var strNum = Console.ReadLine();
19
20                  if (!int.TryParse(strNum, out number))
21                  {
22                      Console.WriteLine("This is not a valid integer");
23                      if (TryAgain())
24                          continue;
25                      else
26                          return false;
27                  }
28
29                  if (!filter(number))
30                  {
31                      Console.WriteLine(filterErrorMessage);
32                      if (TryAgain())
33                          continue;
34                      else
35                          return false;
36                  }
37
38                  break;
39              }
40
41              return true;
42          }
43
44          public static bool TryGetPointInRange(string message, Range xRange,
45                                                Range yRange, out Point point)
46          {
47              while (true)
48              {
49                  Console.Write(message);
50                  var strPoint = Console.ReadLine();
51
52                  if (!Point.TryParse(strPoint, out point))
53                  {
54                      Console.WriteLine("This is not a valid point");
55                      if (TryAgain())
56                          continue;
57                      else
58                          return false;
59                  }
60
61                  if (point.X < xRange.Start.Value ||
```

```
                    point.X > xRange.End.Value   ||
                    point.Y < yRange.Start.Value ||
                    point.Y > yRange.End.Value)
                {
                    Console.WriteLine("Input point is out of range");
                    if (TryAgain())
                        continue;
                    else
                        return false;
                }

                break;
            }

            return true;
        }

        public static bool TryGetPointInRangeAndNotInList(string message,
                                                          Range xRange,
                                                          Range yRange,
                                                          List<Point> existedPoints,
                                                          out Point point)
        {
            while (true)
            {
                Console.Write(message);
                var strPoint = Console.ReadLine();

                if (!Point.TryParse(strPoint, out point))
                {
                    Console.WriteLine("This is not a valid point");
                    if (TryAgain())
                        continue;
                    else
                        return false;
                }

                if (point.X < xRange.Start.Value ||
                    point.X > xRange.End.Value   ||
                    point.Y < yRange.Start.Value ||
                    point.Y > yRange.End.Value)
                {
                    Console.WriteLine("Input point is out of range");
                    if (TryAgain())
                        continue;
                    else
                        return false;
                }
```

```
110
111                 var alreadyExists = false;
112                 foreach (var p in existedPoints)
113                     if (p.Equals(point))
114                     {
115                         alreadyExists = true;
116                         break;
117                     }
118
119                 if (alreadyExists)
120                 {
121                     Console.WriteLine("This point has already been declared");
122                     if (TryAgain())
123                         continue;
124                     else
125                         return false;
126                 }
127
128                 break;
129             }
130
131             return true;
132         }
133
134         public static bool TryGetPointsInRangeAndNotInList(string message,
135                                                            string stopSequence,
136                                                            Range xRange,
137                                                            Range yRange,
138                                                            List<Point> existedPoints,
139                                                            out List<Point> points)
140         {
141             points = new List<Point>();
142
143             while (true)
144             {
145                 Console.Write(message);
146                 var strPoint = Console.ReadLine();
147
148                 if (strPoint.ToLower() == stopSequence)
149                     break;
150
151                 Point point;
152
153                 if (!Point.TryParse(strPoint, out point))
154                 {
155                     Console.WriteLine("This is not a valid point");
156                     if (TryAgain())
157                         continue;
```

```
                    else
                        return false;
                }

                if (point.X < xRange.Start.Value ||
                    point.X > xRange.End.Value   ||
                    point.Y < yRange.Start.Value ||
                    point.Y > yRange.End.Value)
                {
                    Console.WriteLine("Input point is out of range");
                    if (TryAgain())
                        continue;
                    else
                        return false;
                }

                var alreadyExists = false;
                foreach (var p in existedPoints)
                    if (p.Equals(point))
                    {
                        alreadyExists = true;
                        break;
                    }

                if (alreadyExists)
                {
                    Console.WriteLine("This point has already been declared");
                    if (TryAgain())
                        continue;
                    else
                        return false;
                }

                points.Add(point);
            }

            return true;
        }

        public static string CreateGrid(int xdim, int ydim, Point startPoint,
                                         Point goalPoint, List<Point> wallPoints,
                                         List<Point> solutionPoints)
        {
            var gridBuilder = new StringBuilder();
            var offset = string.Empty.PadLeft(10);

            for (int y = ydim - 1; y >= 0; y--)
            {
```

```
                gridBuilder.Append(offset);

                for (int x = 0; x < xdim; x++)
                {
                    var currentPoint = new Point { X = x, Y = y };

                    if (startPoint.X == x && startPoint.Y == y)
                        gridBuilder.Append("S ");
                    else if (goalPoint.X == x && goalPoint.Y == y)
                        gridBuilder.Append("G ");
                    else if (wallPoints.Contains(currentPoint))
                        gridBuilder.Append("* ");
                    else if (solutionPoints.Contains(currentPoint))
                        gridBuilder.Append("- ");
                    else
                        gridBuilder.Append("0 ");
                }

                gridBuilder.AppendLine();
            }

            return gridBuilder.ToString();
        }


        private static bool TryAgain()
        {
            while (true)
            {
                Console.Write("Try Again? (Y/y or N/n) :");
                string answer = Console.ReadLine();

                if (answer.ToLower() == "y")
                    return true;
                else if (answer.ToLower() == "n")
                    return false;
                else
                    Console.WriteLine("This is not a valid answer");
            }
        }

    }
}
```

- *class Program:* Πρόκειται για την κεντρική κλάση σε ένα πρόγραμμα κονσόλας σε *C#* και περιέχει την ειδική μέθοδο *Main* για την επικοινωνία με το τερματικό του λειτουργικού συστήματος. Εδώ γίνεται η σύνδεση όλων των οντοτήτων που προαναφέρθηκαν για την σωστή με βάση τις απαιτήσεις διεπαφή με το χρήστη.

```
using System.Runtime.InteropServices;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System;

namespace PathFinderConsole
{
    class Program
    {
#if DEBUG
        [DllImport(@"..\..\..\..\x64\Debug\PathfinderSolver.dll",
        CallingConvention = CallingConvention.Cdecl)]
#else
        [DllImport("PathfinderSolver.dll",
        CallingConvention = CallingConvention.Cdecl)]
#endif
        public static extern bool Solve(string prologFilePath,
                                        StringBuilder solutions);

        static void Main()
        {
            List<Point> currentPointsInGrid = new List<Point>();
            List<Point> wallPoints = new List<Point>();
            List<Point> solutionPoints = new List<Point>();
            string grid = string.Empty;

            Console.WriteLine("Set maze dimensions:");

            if (!ConsoleUtils.TryGetIntegerWithFilter("x-dimension:",
                                                      xdim => xdim > 0,
                                                "x-dimension cannot be 0 or negative",
                                                      out int XDim))
                return;

            if (!ConsoleUtils.TryGetIntegerWithFilter("y-dimension:",
                                                      ydim => ydim > 0,
                                                "y-dimension cannot be 0 or negative",
                                                      out int YDim))
                return;

            var xRange = ..(XDim - 1);
            var yRange = ..(YDim - 1);
```

```
44
45          Console.WriteLine("Set Start and Goal coordinates in the form (x,y):");
46
47          if (!ConsoleUtils.TryGetPointInRange("Start:", xRange, yRange,
48                                               out Point startPoint))
49              return;
50          currentPointsInGrid.Add(startPoint);
51
52          if (!ConsoleUtils.TryGetPointInRangeAndNotInList("Goal:", xRange, yRange,
53                                                           currentPointsInGrid,
54                                                           out Point goalPoint))
55              return;
56          currentPointsInGrid.Add(goalPoint);
57
58          grid = ConsoleUtils.CreateGrid(XDim, YDim, startPoint,
59                                         goalPoint, wallPoints, solutionPoints);
60          Console.WriteLine($"\nGrid:\n{grid}");
61
62          Console.WriteLine("Choose the position of a wall(*) or type 'X' " +
63                            "to finalize the grid:");
64
65          if (!ConsoleUtils.TryGetPointsInRangeAndNotInList("Wall Point Coordinates:",
66                                                            "x", xRange, yRange,
67                                                            currentPointsInGrid,
68                                                            out wallPoints))
69              return;
70          currentPointsInGrid.AddRange(wallPoints);
71
72          Console.Clear();
73          grid = ConsoleUtils.CreateGrid(XDim, YDim, startPoint, goalPoint,
74                                         wallPoints, solutionPoints);
75          Console.WriteLine($"\nGrid:\n{grid}");
76
77          Console.Write("Continue Solving? (Press Y) : ");
78          string continueStr = Console.ReadLine();
79
80          if (continueStr.ToLower() != "y")
81              return;
82
83          Console.Clear();
84
85          var codeGen = new PrologCodeGenerator(XDim, YDim, startPoint,
86                                                goalPoint, wallPoints);
87
88          var prologFilePath = codeGen.Generate();
89          var solutionsBuilder = new StringBuilder(1024);
90          var solutionsString = string.Empty;
91
```

```
92              try
93              {
94                  if (Solve(prologFilePath, solutionsBuilder))
95                  {
96                      solutionsString = solutionsBuilder.ToString();
97                  }
98                  else
99                  {
100                     Console.WriteLine("Prolog Engine could not be initialized");
101                     return;
102                 }
103             }
104             catch (Exception e)
105             {
106                 Console.WriteLine(e.Message);
107                 return;
108             }
109
110
111             if (string.IsNullOrWhiteSpace(solutionsString))
112             {
113                 Console.WriteLine("No solutions found");
114                 return;
115             }
116
117             var parser = new SolutionsParser(solutionsString);
118             var solutions = parser.GetSolutions();
119
120             for (int i = 0; i < solutions.Count; i++)
121             {
122                 solutionPoints = solutions[i];
123
124                 Console.WriteLine($"Solution No: {i + 1}  Points Count: " +
125                                   "{solutions[i].Count}\n");
126                 solutionPoints.ForEach(p => Console.Write(p));
127                 Console.WriteLine();
128
129                 grid = ConsoleUtils.CreateGrid(XDim, YDim, startPoint, goalPoint,
130                                                wallPoints, solutionPoints);
131                 Console.WriteLine($"\nGrid:\n{grid}");
132
133                 if (i != solutions.Count - 1)
134                 {
135                     Console.Write("\nNext Solution:");
136                     Console.ReadKey();
137                     Console.Clear();
138                 }
139             }
```

```
140
141             Console.Write("Show shortest solutions? (Y): ");
142             string showShortestStr = Console.ReadLine();
143             if (showShortestStr != "Y" && showShortestStr != "y")
144             {
145                 return;
146             }
147
148             Console.WriteLine();
149
150             int shortestLength = solutions.Min(x => x.Count);
151             var shortestSolutions = solutions.Where(s => s.Count == shortestLength)
152                                              .ToList();
153
154             for (int i = 0; i < shortestSolutions.Count; i++)
155             {
156                 solutionPoints = shortestSolutions[i];
157
158                 Console.WriteLine($"Shortest Solution No: {i + 1}  Points Count: " +
159                                   "{shortestLength}\n");
160                 solutionPoints.ForEach(p => Console.Write(p));
161                 Console.WriteLine();
162
163                 grid = ConsoleUtils.CreateGrid(XDim, YDim, startPoint, goalPoint,
164                                                wallPoints, solutionPoints);
165                 Console.WriteLine($"\nGrid:\n{grid}");
166
167                 if (i != shortestSolutions.Count - 1)
168                 {
169                     Console.Write("\nNext Shortest Solution:");
170                     Console.ReadKey();
171                     Console.Clear();
172                 }
173             }
174
175             Console.ReadKey();
176         }
177     }
178 }
```

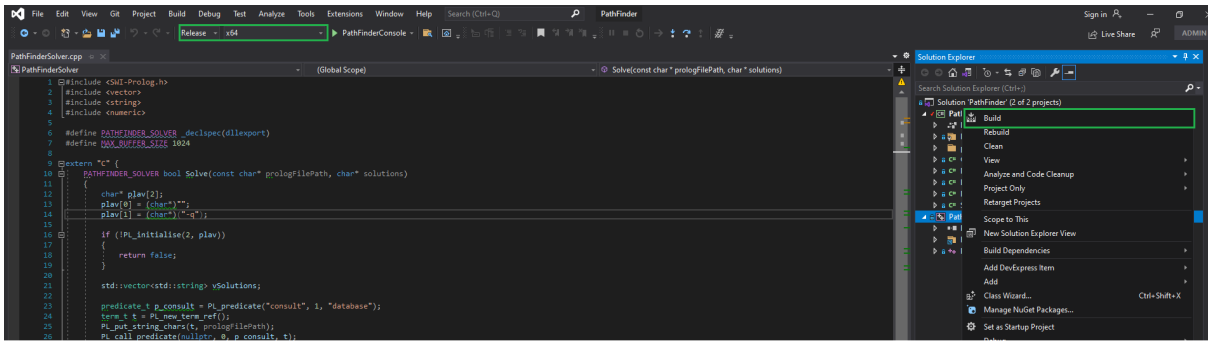
Τέλος, αξίζει να τονιστεί πώς πραγματοποιείται η συνεργασία με τη βιβλιοθήκη της *C++*. Όπως σημειώθηκε στο προηγούμενο κεφάλαιο, η συνάρτηση της *C++* που επιστρέφει τις λύσεις ονομάζεται *Solve* και γυρνάει μια λογική μεταβλητή που σχετίζεται με το αν η επικοινωνία με την *Prolog* ήταν επιτυχής. Δέχεται ώς παράμετρο μια συμβολοσειρά που έχει ως τιμή ολόκληρο το μονοπάτι του αρχείου κώδικα της *Prolog* που δημιουργήθηκε δυναμικά και άλλη μια μεταβλητή τύπου συμβολοσειράς ώστε να επιστραφεί η λύση. Η συνάρτηση ορίζεται ως εξωτερική με τη βοήθεια του ειδικού προσδιοριστικού *extern* στη δηλωσή της. Επίσης είναι πολύ σημαντικό να δωθεί σωστά το μονοπάτι στο οποίο βρίσκεται η βιβλιοθήκη που περιέχει την υλοποίηση της *Solve*. Αυτό όπως φαίνεται και παραπάνω δίνεται ως σχετικό μονοπάτι με βάση τη θέση που έχει το εκτελέσιμο της *C#* κατά την εκτέλεση. Επίσης θα παρατηρήσετε ότι υπάρχουν δύο ορισμοί της συνάρτησης μέσα σε μια preprocessor δομή ελέγχου. Αυτό γίνεται γιατί το μονοπάτι θέλουμε να αλλάζει σε σχέση με το σε τι τύπου έκδοση γίνεται το *compiling*. Αν βρισκόμαστε σε περιβάλλον ανάπτυξης (*Debug*) είναι επιθυμητό να η *C#* να βρίσκει τη βιβλιοθήκη της *C++* στο μονοπάτι που εξάγεται από το αντίστοιχο *Project*. Από την άλλη, ετοιμάζοντας το τελικό προϊόν (*Release*) είναι προτιμότερο να προετοιμάσουμε τη *C#* να καταναλώσει την εξωτερική βιβλιοθήκη από το ίδιο ακριβώς μονοπάτι στο οποίο βρίσκεται και το δικό της παραγόμενο εκτελέσιμο. Στην ουσία θέλουμε το τελικό πρόγραμμα να περιεχεί προφανώς όλες τι εξωτερικές εξαρτήσεις του σε κοινό σημείο. Μια άλλη σημαντική λεπτομέρεια είναι η επιλογή του τύπου για την δεύτερη παράμετρο της συνάρτησης που θα πρέπει να γεμίσει με τις λύσεις. Η *C++* και η *C#* ακολουθούν πολύ διαφορετικά μοντέλα γαι τις συμβολοσειρές σε σχέση με τη διαχείριση της μνήμης. Η διαφορά έγκειται στο γεγονός ότι στη *C#* τα *string* είναι μια κλάση που στην πραγματικότητα δεν αλλάζει ποτέ μετά από την κατσκευή της, είναι αυτό που λέμε συνήθως στην ορολογία *immutable*. Αυτό σημαίνει ότι όποτε στον κώδικα αλλάζουμε ένα αντικείμενο τύπου *string*, εσωτερικά δεν αλλάζει στην πραγματικότητα, αλλά κάθε φορά κατασκευάζεται ένα νέο αντικείμενο. Αυτή η λειτουργία έρχεται σε αντίθεση όμως με το σκοπό μας, δηλαδή να σταλεί στη *C++* ένα *string* που θα γεμίσει με τη συμβολοσειρά των λύσεων. Αν, λοιπόν, επιλέξουμε την κλάση *string* το πρόγραμμα θα αποτύχει αφού η *C++* θα προσπαθήσει να αλλάξει προστατευμένη μνήμη. Η κατάλληλη κλάση για αυτό το σκοπό είναι η *StringBuilder* αφού επιτρέπει αλλαγές στα αντικείμενα του τύπου της.

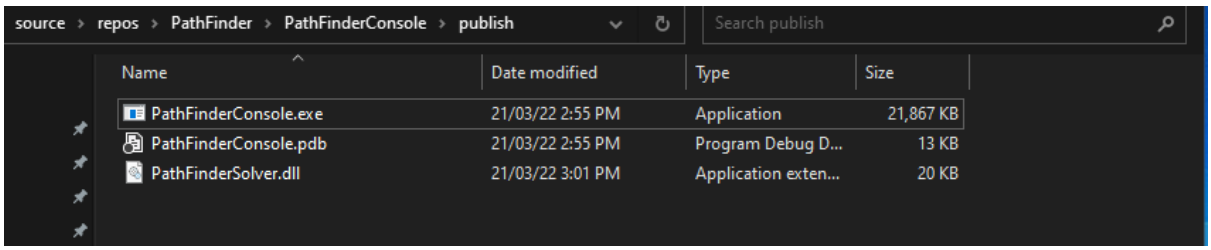
6 Κατασκεύη και εκτέλεση του τελικού προγράμματος

Η διαδικασία της δημιουργίας του τελικού προγράμματος είναι αρκετά εύκολη. Για την περίπτωση του προγράμματος της *C#* μπορούμε να χρησιμοποιήσουμε το βασικό εργαλείο κονσόλας του *.NET* που είναι το *dotnet.exe*. Επιλέγουμε τα κατάλληλα *flags* ώστε να παράγουμε ένα μοναδικό 64-bit εκτελέσιμο το οποίο θα ενθυλακώνει όλες τις εξαρτήσεις του *.NET Framework* ώστε να μπορεί δυνητικά να εκτελεστεί και σε υπολογιστή που δεν υπάρχει τοπική εγκατάσταση του *.NET*.

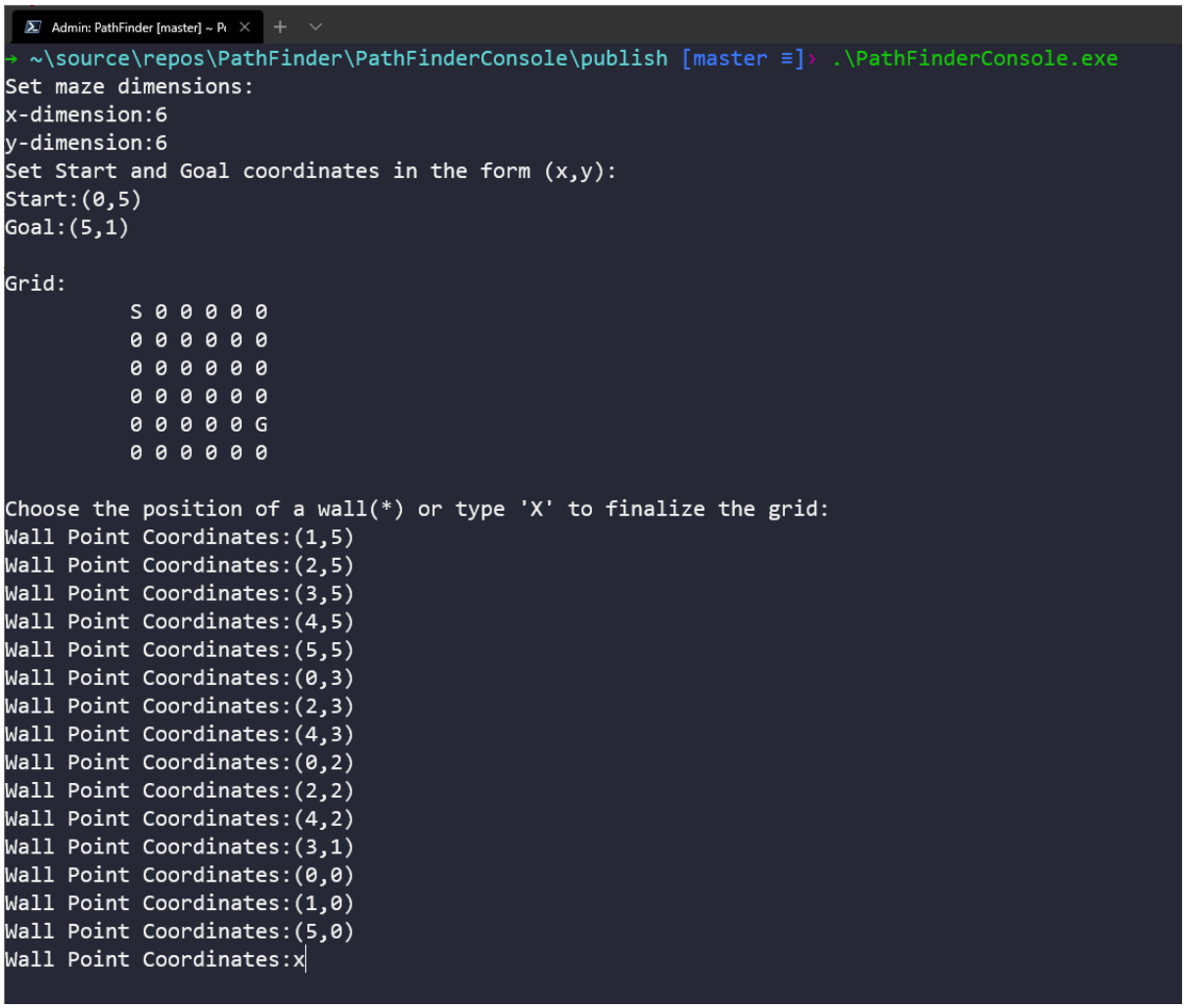
```
Admin: PathFinder [master] ~ Pa
→ ~\source\repos\PathFinder\PathFinderConsole [master ≡]> dotnet  publish -r win-x64 -c Release /p:PublishSingleFile=true --self-contained true
Microsoft (R) Build Engine version 16.9.0+57a23d249 for .NET
Copyright (C) Microsoft Corporation. All rights reserved.

  Determining projects to restore...
  All projects are up-to-date for restore.
  PathFinderConsole -> C:\Users\DeVM\source\repos\PathFinder\PathFinderConsole\bin\Release\net5.0\win-x64\PathFinderConsole.dll
  PathFinderConsole -> C:\Users\DeVM\source\repos\PathFinder\PathFinderConsole\bin\Release\net5.0\win-x64\publish\
→ ~\source\repos\PathFinder\PathFinderConsole [master ≡]>
```

Έπειτα με τη βοήθεια του *Visual Studio* θα πραγματοποιήσουμε επίσης ένα *build* με ρυθμίσεις τύπου *Release* στο *Project* της *C++* και δεν ξεχνάμε να αντιγράξουμε την παραγόμενη βιβλιοθήκη στο ίδιο μονοπάτι που βρίσκεται το εκτελέσιμο της *C#*.



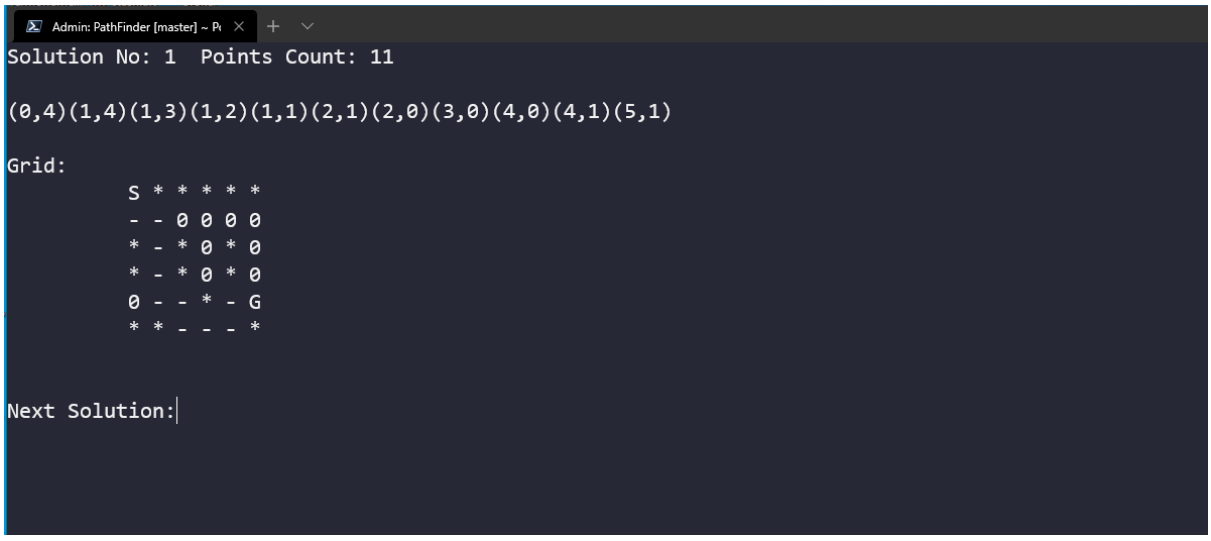


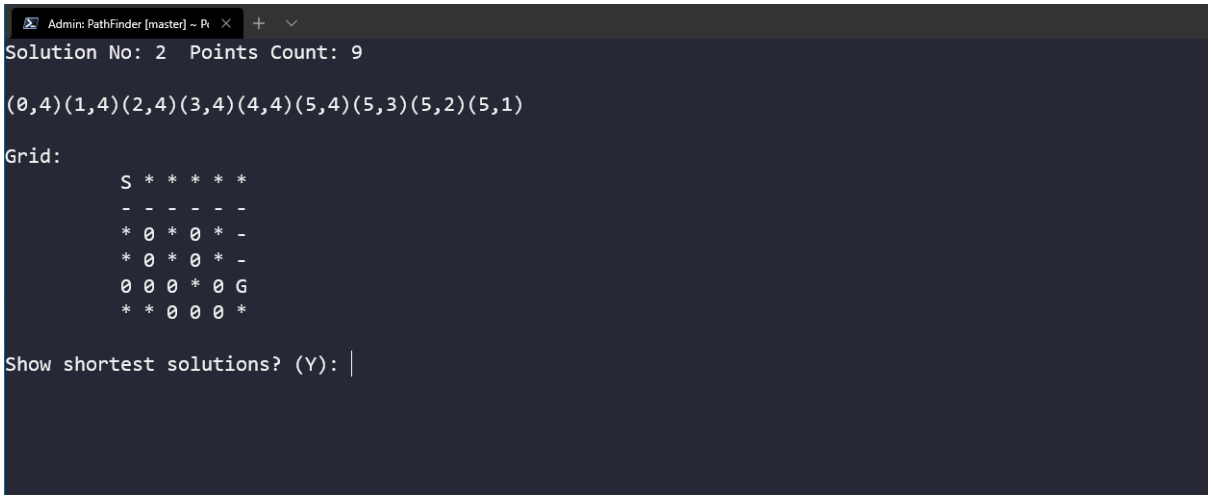
Παρακάτω ακολουθούν εικόνες με την εκτέλεση του τελικού προγράμματος. Αρχικα ο χρήστης πρέπει να εισάγει τα δεδομένα με τα οποία θα κατασκευαστεί ο λαβύρινθος.



Αφού ο χρήστης ολοκληρώσει την κατασκευή, η κονσόλα θα καθαρίσει και θα εμφανιστεί ο τελικός λαβύρινθος.

Αφού ο χρήστης επιλέξει να συνεχίσει με την επίλυση του συγκεκριμένου λαβυρίνθου, η κονσόλα εμφανίζει τα σημεία της κάθε λύσης και την αντίστοιχη διαδρομή μεσα στον λαβύρινθο. Για κάθε λύση που παρουσιάζεται η κονσόλα καθαρίζεται και ο χρήστης επιλέγει πότε επιθυμεί να δει την επόμενη λύση πατώντας το *Enter*.





Τέλος, αν την επιθυμεί, δίνουμε στον χρήστη τη δυνατότητα να δει την (ή τις) συντομότερες διαδρομές.