UNIVERSIDAD
COMPLUTENSE
MADRID

Proyecto de Innovación y Mejora de la Calidad Docente

Convocatoria 2014

Proyecto nº 245

Desarrollo de una herramienta de depuración simbólica para las asignaturas de iniciación a la programación en las facultades de Informática y Estudios Estadísticos

Responsable del proyecto: Miguel Gómez-Zamalloa Gil

Facultad de Informática

Departamento de Sistemas Informáticos y Computación

1. Objetivos propuestos en la presentación del proyecto

La programación de computadores es una de las materias fundamentales en los primeros cursos de los grados de las Facultades de Informática, Estudios Estadísticos y Matemáticas. Las asignaturas de iniciación a la programación juegan un papel fundamental en la formación de los estudiantes de dichos grados, y suelen resultar muy complicadas para ellos. Si bien la mayoría de estudiantes al finalizar el curso son capaces de escribir programas para resolver problemas relativamente sencillos, cuando los problemas crecen en complejidad, los programas que escriben habitualmente no se comportan como deben para todos los casos, siendo por tanto parcialmente incorrectos. Escribir programas totalmente correctos requiere el uso de una serie de metodologías que en general resultan muy complejas para los estudiantes. La mayoría de ellos siguen una mecánica de prueba-error, es decir, escriben los programas sin pararse a pensar demasiado acerca de su corrección, y luego los prueban para ver si se comportan como esperan. Estas pruebas normalmente no son lo suficientemente exhaustivas y por lo tanto los estudiantes normalmente no son ni siquiera conscientes de las incorrecciones de sus programas.

En general, razonar acerca de la corrección de los programas es una tarea muy compleja. El "testing" de programas es la técnica más utilizada en este contexto. Consiste en ejecutar los programas con una serie de datos de prueba con el objetivo de comprobar que el resultado es el esperado. Al encontrar un resultado inesperado se inicia un costoso proceso de "depuración" para encontrar la causa del problema. Desgraciadamente, los mecanismos y herramientas que ayudan en el testing y depuración de programas son en general demasiado avanzados para poder ser utilizados por programadores inexpertos.

Este proyecto estudia y desarrolla una herramienta de "depuración simbólica" que permite estudiar el funcionamiento de programas informáticos sin ser ejecutados, a base de observar todas sus posibles ramas de ejecución (hasta un cierto nivel) así como los correspondientes pares entrada-salida. Dicha herramienta podría ayudar enormemente a los estudiantes de iniciación a la programación, y en general a programadores inexpertos, a la hora de razonar acerca de la corrección de sus programas.

La herramienta dispone de un interfaz de usuario basado en web, proporcionando varias ventajas adicionales: por una parte, permite el uso de la herramienta a través de internet, lo que fomentará el aprendizaje autónomo del alumno. El acceso a través de internet permite enriquecer el entorno virtual de aprendizaje del alumno, ya que se puede enlazar desde el campus virtual y el alumno puede utilizar la herramienta antes de enviar los ejercicios para su corrección como tareas del campus. En segundo lugar, el acceso a través de internet permite obtener datos estadísticos sobre el uso de la

herramienta, número de envíos, tipos de problemas encontrados por los alumnos, etc. Por último, el depurador simbólico estará disponible públicamente como sistema de acceso abierto (Open Access).

La herramienta puede proporcionar múltiples beneficios. Por un lado se disminuye el tiempo de prueba, pudiendo emplear dicho tiempo en implementar un código más estructurado y eficiente. Por otra parte, se fomenta en el alumno el aprendizaje autónomo en un entorno virtual, pues no es necesaria la asistencia del profesor después de una sesión introductoria. Y en tercer lugar, se facilitará enormemente la corrección de las prácticas por parte del profesor, ya que una vez validadas utilizando la herramienta, el profesor puede centrarse más en el diseño correcto del código sin necesidad siquiera de ejecutarla. Sin lugar a dudas, aprender a diseñar e implementar programas correctamente es un pilar fundamental para poder cursar con éxito el resto de asignaturas de los grados, donde se da por supuesto que dichos conocimientos se han adquirido en los primeros cursos.

Las asignaturas cuyos alumnos podrán beneficiarse de esta herramienta son las siguientes:

- Fundamentos de la Programación, obligatoria del primer curso de los grados en Ingeniería Informática, Ingeniería de Computadores e Ingeniería del Software de la Facultad de Informática.

- Programación I y Programación II, obligatorias del primer curso del grado en Estadística Aplicada de la Facultad de Estudios Estadísticos.

- Estructuras de Datos y Algoritmos, obligatoria del segundo curso de los grados en Ingeniería Informática, Ingeniería de Computadores e Ingeniería del Software de la Facultad de Informática.

Como efecto colateral, pero no menos importante, hacemos notar que las asignaturas de programación en los primeros cursos tienen un número elevado de alumnos, y que las entregas de prácticas, implementadas correctamente, es indispensable para la superación de la asignatura. El número elevado de alumnos hace difícil poder dedicar el tiempo suficiente a cada uno de ellos, por lo que es de gran utilidad una herramienta que permita que el alumno sea más autónomo. Por otra parte, esta herramienta puede ayudar a que la resolución de dudas por parte del profesor se limite a comprobar exclusivamente la parte del código que nos lleva a la solución incorrecta. Esa parte de código se puede determinar con el uso de la herramienta, ya que además de generar casos de prueba, la herramienta mostrará la traza de ejecución para una solución determinada siempre que lo solicite el usuario. Así pues, cuando el alumno detecte que alguna solución es incorrecta, podrá seguir la traza de ejecución mostrada para encontrar el error de implementación.

2. Objetivos alcanzados

En este proyecto se ha diseñado e implementado una herramienta de depuración simbólica para ayudar a los estudiantes de iniciación a la programación a razonar acerca de la corrección de sus programas y a encontrar y corregir posibles fallos.

La herramienta se ha implementado para programas escritos en el lenguaje de programación C++, uno de los lenguajes más utilizados a nivel mundial y el lenguaje elegido para todas las asignaturas de programación del primer curso, y algunas del segundo curso, de los grados de las Facultades de Informática y Estudios Estadísticos. Cabe destacar que el motor de la herramienta es independiente del lenguaje de programación y por tanto podría ser reutilizado para poder tratar con otros lenguajes de programación, como por ejemplo Python, un lenguaje emergente y utilizado en la asignatura de iniciación a la programación en el grado de Matemáticas.

De cara a abordar dicho objetivo fundamental se han identificado los siguientes sub-objetivos

1) Diseño e implementación del motor de la herramienta: La técnica base que ha servido como motor fundamental de la herramienta es la "ejecución simbólica". A diferencia de la "ejecución concreta" en la cual un programa se ejecuta con unos valores concretos de entrada y da lugar a una salida concreta, en la ejecución simbólica la ejecución se realiza sin valores concretos de entrada, o lo que es lo mismo, con valores simbólicos. Este tipo de ejecución produce una exploración sistemática de todos los posibles caminos de ejecución de un programa, y ha sido y sigue siendo utilizada como técnica de base para verificadores y testeadores de programas.

2) Desarrollo de interfaces: Una vez se dispone del motor de la herramienta, el siguiente paso ha consistido en el desarrollo de los interfaces que hacen posible su uso de manera sencilla y amigable para estudiantes y/o programadores inexpertos. Se han desarrollado dos interfaces de usuario: (1) una interfaz gráfica en forma de aplicación, que permitirá a los usuarios que usen la herramienta habitualmente instalarla de forma local en sus equipos; y (2) una interfaz web, que permitirá el uso de la herramienta a través de internet sin necesidad de llevar a cabo instalación alguna. En la interfaz web se plantea permitir la identificación del usuario de la herramienta, para poder obtener información estadística sobre el número de accesos y resultados básicos de su utilización.

3) Evaluación experimental: Para verificar la aplicabilidad de la herramienta en los cursos de iniciación a la programación, ésta se ha estado probando de forma controlada por profesores de estas asignaturas, antiguos alumnos y algún alumno voluntario. Tras las primeras pruebas se concluye que aunque la herramienta es muy prometedora por lo mucho que podría ayudar a los alumnos, su uso no es aún lo suficientemente sencillo e intuitivo como para poder ser usada de manera autónoma por los estudiantes. Se requieren por tanto algunas mejoras fundamentalmente en las interfaces de usuario, en la presentación de los datos y en la documentación de ayuda.

3. Metodología empleada en el proyecto

Se han identificado una serie de tareas orientadas a conseguir los objetivos mencionados. Cada una de estas tareas ha tenido un responsable principal, que ha diseñado el plan para conseguir el objetivo de la tarea, delegando para ello en los demás miembros del proyecto de la manera que ha considerado oportuna. La metodología utilizada durante el proceso de diseño e implementación ha sido la habitual de los proyectos de desarrollo de aplicaciones (ver más detalles en el apartado 5).

En las fases finales del proyecto se ha evaluado la herramienta de depuración simbólica de forma restringida por profesores involucrados en las asignaturas mencionadas, así como por algún antiguo alumno. Se les ha explicado el uso de la herramienta en una sesión introductoria y se les ha proporcionado acceso para que la utilicen con algunos programas de ejemplo. Aunque en la propuesta del proyecto estaba planteado el uso real de la herramienta por los alumnos en una sesión de laboratorio, con el objetivo de hacer una prueba más realista en la que recoger las dudas y los comentarios de los alumnos, esto no ha sido posible debido a las restricciones de temario y a la falta de madurez en el desarrollo de las interfaces de la herramienta. Se preven por tanto aún modificaciones y mejoras para ajustar el uso de la herramienta a las características de un curso de iniciación a la programación. En todo caso, estas mejoras y extensiones serán de aplicación general para el sistema y su uso posterior como sistema de acceso abierto.

En el informe final, el responsable de cada tarea se ha encargado de la parte del informe correspondiente a su tarea, de nuevo posiblemente delegando a su conveniencia. El responsable principal del proyecto ha sido el encargado de juntar las distintas partes del informe y producir el informe final.

4. Recursos humanos

Todos los profesores integrantes del proyecto tienen amplia experiencia en la impartición de asignaturas de las distintas titulaciones en Informática relacionadas con la programación. El siguiente grupo de trabajo se ha compuesto de tal manera que todos los aspectos a ser estudiados en el proyecto, directa o indirectamente, queden cubiertos por los conocimientos de sus miembros:

- Miguel Gómez-Zamalloa Gil: Lleva 9 años impartiendo asignaturas relacionadas con la programación y con las estructuras de datos y algoritmos. Desde su origen con los nuevos grados de la Facultad de Informática es profesor de la asignatura "Fundamentos de la Programación" del primer curso de todos los grados. Su investigación se ha centrado en el testing formal basado en la ejecución simbólica (tecnología de base de la herramienta objeto de este proyecto).

- Purificación Arenas Sánchez: Tiene una dilatada experiencia docente de casi 20 años. En particular, ha sido profesora responsable de "Laboratorio de Programación 2" y "Programación Orientada a Objetos", ambas utilizando el lenguaje C++, y la asignatura sucesora de estas dos, "Tecnología de la Programación".

- Elvira Albert Albiol: Lleva 15 años impartiendo clases relacionadas con la programación. Ha sido profesora responsable durante 5 años de la asignatura "Laboratorio de Programación 3", una de las asignaturas predecesoras de "Tecnología de la Programación". Además co-lidera el grupo de investigación COSTA.

- Jesús Correas Fernández: Tiene 11 años de experiencia docente. Es profesor responsable de "Programación 1" y "Programación 2" en la Fac. de Estudios Estadísticos, y ha sido profesor de "Laboratorio de Programación 2" en Informática, todas ellas utilizando el lenguaje C++.

- Samir Genaim: Tiene una amplia experiencia impartiendo este tipo de asignaturas en diferentes universidades. Ha sido profesor responsable de "Tecnología de la Programación", "Laboratorio de Programación 2" e "Introducción a la Programación". Además, es el principal desarrollador dentro del grupo de investigación COSTA.

- Germán Puebla Sánchez: Tiene una amplia experiencia impartiendo este tipo de asignaturas en la Facultad de Informática de la UPM durante más de 15 años. Además, co-lidera el grupo de investigación COSTA y es uno de los investigadores españoles más citados en el área de lenguajes de programación.

- Guillermo Román Díez: Ha sido profesor responsable de asignaturas de programación y además ha realizado su actividad profesional en la empresa privada en el desarrollo y gestión de proyectos software en diversas empresas. Es experto en programación Java y en la creación de páginas y servidores web.

- Raquel Peces, Carlos Gabriel Giraldo y Clara Antolín son estudiantes de último curso de la antigua Ingeniería Informática, todos ellos con un expediente académico más que notable. A parte de haber sido alumnos de asignaturas de iniciación a la programación con C++ recientemente, tienen conocimientos y experiencias de uso en la mayoría de tecnologías que se emplearán en el proyecto.

5. Desarrollo de las actividades

Se han llevado a cabo las siguientes actividades:

1) Definición de requisitos y limitaciones: El lenguaje de programación C++ es demasiado complejo como para plantearse tratar todas sus funcionalidades. Además en los cursos de iniciación a la programación solo se considera un subconjunto suficiente para iniciarse en la programación. Se ha definido el subconjunto de funcionalidades de C++ a ser consideradas, el cual incluye: declaraciones de variables, instrucciones de asignación, expresiones aritméticas y booleanas clásicas, instrucciones de control “if”, “while” y “for”, operaciones de entrada/salida simple, y llamadas a funciones. Solo se ha considerado el tipo de los números enteros “int”.

2) Diseño de la herramienta: En esta actividad se ha diseñado la arquitectura básica de la herramienta así como la de cada uno de sus componentes básicos. En particular se han desarrollado los siguientes componentes básicos: (I) Un parseador de programas C++ (acorde al subconjunto de C++ definido en la tarea 1); (II) Un ejecutor simbólico de programas C++ (acorde al subconjunto de C++ definido en la tarea 1); (III) Una interfaz gráfica de usuario de tipo aplicación, implementada en el lenguaje de programación Java; (IV) Una interfaz web.

3) Implementación del motor de la herramienta: Éste consta de dos componentes fundamentales: un parseador de programas C++ y un ejecutor simbólico. Se distinguen por tanto las siguientes dos actividades:

3a) Desarrollo de un parseador de programas C++: Los programas C++ deben ser analizados e interpretados con el objetivo de obtener una representación en el nivel de abstracción conveniente (típicamente en forma de "árbol sintáctico abstracto"). En esta actividad se ha estudiado el uso de compiladores de C++ existentes de código abierto. Finalmente se ha decidido utilizar el compilador "clang". La estructura que emplea CLANG para representar el AST de un código se basa principalmente en la interacción de dos clases base muy flexibles a partir de las cuales se construyen todas las demás: Decl (Declarations), que engloba toda las declaraciones (funciones, variables, templates, etc.), y, Stmt (Statement) que abarca las instrucciones. La mayoría de las clases que se derivan de ellas se explican por sí mismas, como por ejemplo: BinaryOperator (operador binario), FunctionDecl (declaración de función), etc.

Como resultado, el parseador genera un fichero en formato XML con el árbol sintáctico abstracto del programa. Véase el siguiente ejemplo:

```
<function name="llamadaFactorial" type="int" line="79">
        <params>
                <param type="int" name="a"/>
        </params>
```

```
        <body>
            <return line="80">
            <callFunction name="factorial" type="int">
            <arg>
                <variable name="a"/>
            </arg>
            </callFunction>
            </return>
        </body>
</function>
```

3b) Motor de ejecución simbólica en Prolog: El árbol sintáctico abstracto generado es posteriormente leído desde un programa escrito en el lenguaje Prolog. El motor de ejecución simbólica, escrito en Prolog, dará como resultado una descripción de la ejecución de cada una de las ramas consideradas de acuerdo a los parámetros establecidos, que será devuelta de nuevo en forma de fichero XML. Véase el siguiente ejemplo:

```
<caso>
    <variable name="ret" value="-1"/>
    <variable name="a" value="-5"/>
    <data>
        <traza> 76 77 83</traza>
    <cin/>
    <cout/>
    </data>
</caso>
```

4) Pruebas y depuración: Debido a la complejidad de la herramienta ha sido necesario llevar a cabo un riguroso proceso de pruebas y depuración con el objetivo de encontrar y solucionar posibles errores de de implementación.

5) Interfaz gráfica: Se ha desarrollado una interfaz gráfica de usuario en forma de aplicación. Como lenguaje de implementación de dicha interfaz se ha escogido el lenguaje Java, y en particular los paquetes AWT y SWING. La interfaz lee el fichero XML generado por el motor de ejecución simbólica e interactua con el usuario.

6) Interfaz web: Se ha desarrollado una interfaz web que permite a los usuarios probar y ejecutar la herramienta desde internet sin necesidad de instalar software alguno. Al igual que en la tarea anterior, la interfaz web lee el fichero XML generado por el motor de ejecución simbólica e interactua con el usuario.

7) Evaluación experimental: Se ha probado la herramienta de forma controlada por profesores de estas asignaturas, antiguos alumnos y algún alumno voluntario. Tras las

primeras pruebas se concluye que aunque la herramienta es muy prometedora por lo mucho que podría ayudar a los alumnos, su uso no es aún lo suficientemente sencillo e intuitivo como para poder ser usada de manera autónoma por los estudiantes. Se requieren por tanto algunas mejoras fundamentalmente en las interfaces de usuario, en la presentación de los datos y en la documentación de ayuda.probará con un grupo reducido de alumnos para evaluar su idoneidad en el contexto de un curso de iniciación a la programación.

6. Anexos. Manual de usuario

Apéndice A

Manual de usuario

A.1. Requisitos previos

Como ya hemos indicado en las secciones de la memoria, nuestro proyecto actualmente sólo funciona en sistemas operativos LINUX, por tanto en lo siguiente supondremos que se trabaja bajo este sistema operativo.

Por otro lado para poder ejecutar el sistema que hemos desarrollado será necesario cumplir una serie de requisitos software previos. En concreto deberemos tener instalados los siguientes componentes:

- **"Java Runtime Environment"** o JRE de 32 bits, versión 1.6 o superior.
  Para ello desde la propia página recomiendan, en lo relativo a requisitos hardware, contar con un Pentium 2 a 266 MHz o un procesador más rápido con al menos 128 MB de RAM física.
  Enlace de descarga: `http://www.java.com/es/download/`

- **"SWI-Prolog"**. Para poder ejecutar el intérprete simbólico es necesario tener instalada una versión establde de SWI-Prolog.
  Enlace de descarga: `http://www.swi-prolog.org/download/stable`

- **"Clang"**. Al comienzo del proyecto la última versión disponible era la 3.5, por lo que esta fue la que se utilizó.
  Para poder compilar nuestra herramienta de generación del AST2XML, cuyos pasos detallados de instalación comentamos en la próxima sección, es necesario tener instalado Clang en nuestro sistema. Puesto que la instalación y requisitos ya están detallados en la propia web de Clang no vamos a detenernos a comentarlos uno a uno.

- **"Requisitos LLVM System"**:
  Enlace de requisitos: `http://llvm.org/docs/GettingStarted.html#requirements`
- **"Python"**. Para compilar Clang es necesario tener instalado Python en nuestro sistema.
  Enlace de descarga: `https://www.python.org/download`
- **"Compilacion de Clang"**
  Enlace con los pasos detallados `http://clang.llvm.org/get_started.html`

A.2. Instalación herramienta ast2xml

Es necesario compilar la herramienta ast2xml en nuestro ordenador para poder utilizarla, puesto que es diferente para cada equipo. En esta sección damos por sentado que el usuario ya ha instalado Clang en su ordenador, a continuación detallamos los pasos para su compilación en nuestro sistema para luego poder utilizarla.

- Dirigirse a nuestro directorio LLVM.
  cd (directorio llvm)/tools/clang/tools
- Clonar el repositorio con nuestra herramienta en una carpeta del directorio.
  git clone `https://github.com/Si1314/AST2XML.git` **ast2xmltool**
- Copiar el Makefile.
- Ir al directorio de compilación.
  cd (directorio de compilación)/tools/clang/tools
- Crear un directorio específico para la herramienta (con el mismo nombre).
  mkdir ast2xmltool
- Copiar en él el archivo Makefile.
- Ir dentro del directorio.
  cd ast2xmltool
- Ejecutar el comando "make"
  La herramienta estará compilada y enlazada si no hay ningún problema

- Volver al directorio de compilación.
  cd (directorio de compilación)/Debug + Asserts/build

  Aquí debería encontrarse la herramienta, así como el resto de herramientas proporcionadas por el paquete de clang.

A.3. Instalación interfaz de usuario

Para poder utilizar la interfaz de usuario que hemos desarrollado será necesario descargarla desde nuestro repositorio. La interfaz se encuentra en un archivo comprimido con todo el directorio de archivos necesario, incluidos los archivos del intérprete simbólico.

Para empezar a utilizarlo será necesario descomprimir el archivo comprimido en el directorio que desee el usuario y ejecutar la aplicación.

A.4. Tutorial de uso

Esta sección la dedicaremos a explicar como usar la herramienta SymC++ a través de su interfaz gráfica.

En primer lugar, cuando la ejecutamos por primera vez, nos dará un aviso para seleccionar donde tenemos instalada nuestra herramienta ast2xml, la cual es necesaria para la ejecución del programa, por tanto se nos mostrará la siguiente imagen como aviso para seleccinar la herramienta (Figura A.1) y seleccionando "aceptar" se mostrará la ventana para navegar en busca de ast2xml (Figura A.2).

Una vez que se abre la interfaz completa se muestra como un editor de texto con las funciones propias (abrir, copiar, guardar, deshacer, etc...). La vista normal de la interfaz se muestra en la figura A.3.

A partir de aquí, podemos tanto escribir nuestra función en el editor de texto, como cargar un archivo C++ y seleccionar la función que queremos testear. En la figura A.4 se muestra una captura con el código ya cargado.

Para lanzar la herramienta de ejecución simbólica, es necesario que el texto a escribir esté guardado. Por tanto, si al seleccionar "Run" desde el menú, o pinchar en la tecla de "play" desde la barra de herramientas el texto no está guardado pueden ocurrir dos situaciones:

- Si el texto procede de un archivo se guardará automáticamente en su archivo origen.

- Si hemos redacatado el texto desde el editor, se abrirá una ventana para guardar el archivo con el nombre y en el directorio que deseemos

Una vez guardado el archivo C++ se mostrará una ventana con los valores de entrada necesarios para que nuestro intérprete simbólico sepa de que función de todo el archivo queremos obtener los resultados, y el rango de valores y la profundidad de ejecución de los bucles. En la figura A.5 se muestra la ventana para introducir los valores de entrada.

A continuación la interfaz cargará los resultados generados en la "Results table" la cual se muestra en la figura A.6, quedando la vista principal como se muestra en la figura A.7.

Para poder ver los detalles del resultado de la ejecución del intérprete debemos selecccionar el resultado del que queremos ver los detalles haciendo "doble click" sobre él, y se cargará la traza de ejecución en su zona de resultados en "Execution tracking" como se puede observar en la figura A.8 al mismo tiempo que se resaltará el texto de las líneas por las que ha pasado en la zona del editor como se puede ver en la figura A.9.

La interfaz tiene dos zonas de resultados más, que también se rellenaran al hacer "doble click" si los resultados las usan, son las zonas de la "Input console" y "Output console" que se pueden observar en la figura A.10. Estas "consolas" simulan la consola de entrada propia del compilador, cuando el usuario introduce un número y el número que generaría por la consola de salida.

Por otro lado, se pueden las distintas opciones que se nos presentan desde el menú. En las figuras A.11, A.12, A.13 y A.14 podemos ver todas las opciones que detallamos a continuación.

- **File:** En este submenú (figura A.11) se encuentran todas las opciones de archivos: abrir un nuevo archivo, guardar, salir de la aplicación, etc...

- **Edit:** En este submenú (figura A.12) se encuentran todas las opciones de edición de texto: copiar, cortar y pegar, hacer y deshacer, etc...

- **Run:** En este submenú (figura A.13) se encuentran las opciones de ejecución de la herramienta, la primera opción "Change params to run" permite configurar los valores de ejecucion de la función (rango de valores enteros, nombre función y profundidad para recorrer los bucles). La opción "run project" procede a la ejecución por parte de nuestro intérprete de la función seleccionada.

- **Debug:** En este submenú (figura A.14) se encuentran las opciones de depuración, por un lado, la opción de ver y cambiar donde se encuentra

instalada la herramienta ast2xml. Por otro lado, la opción de poder ver los XMLs generados tanto por la herramienta ast2xml como por el intérprete de prolog (figura A.15).

Capturas de pantalla

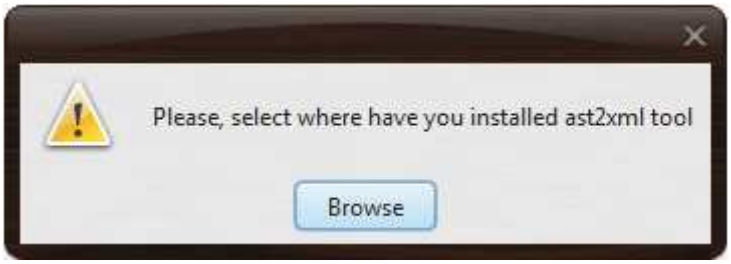

Figura A.1: Figura con el mensaje de aviso para seleccionar la herramienta ast2xml.

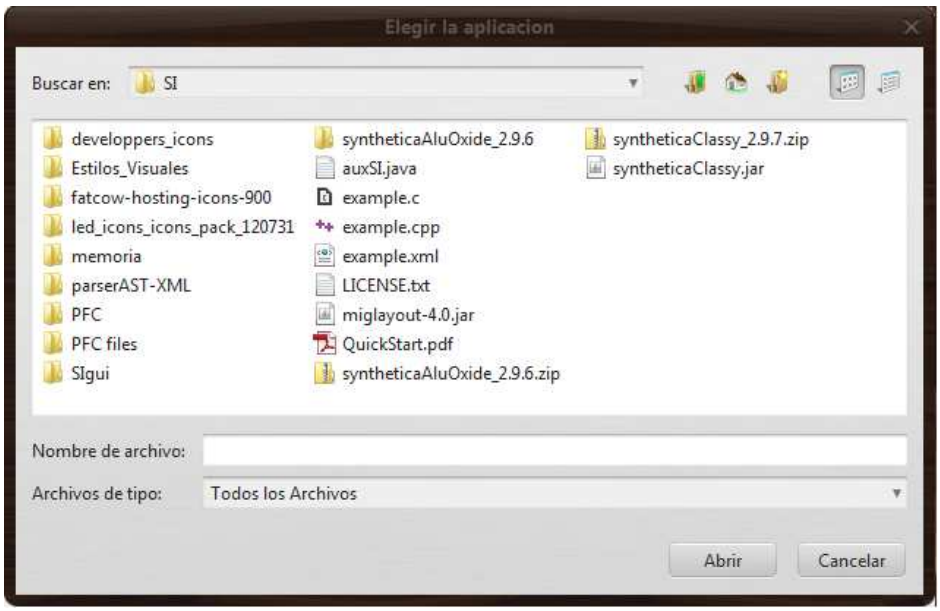

Figura A.2: Figura con el directorio de archivos para buscar la herramienta ast2xml.

Figura A.3: Figura con el directorio de archivos para buscar la herramienta ast2xml.

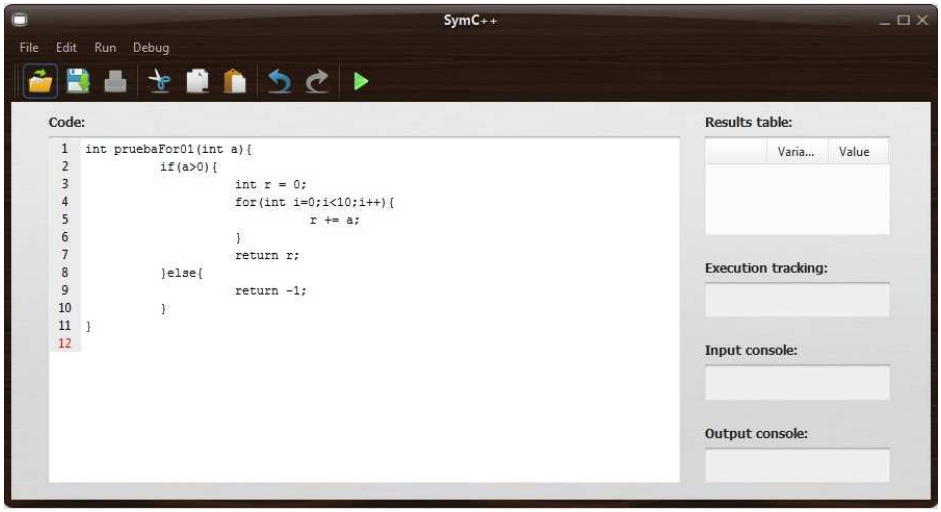

Figura A.4: Figura con una función escrita en el editor.

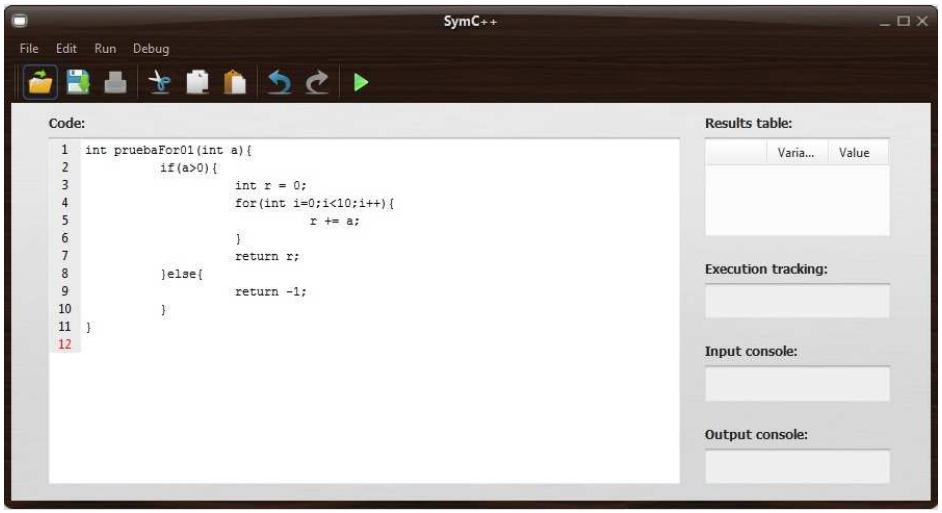

Figura A.5: Figura con la ventana para introducir los valores de entrada del intérprete.

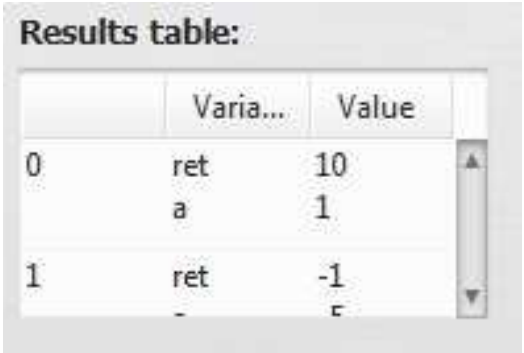

Figura A.6: Figura con la vista de la tabla de resultados.

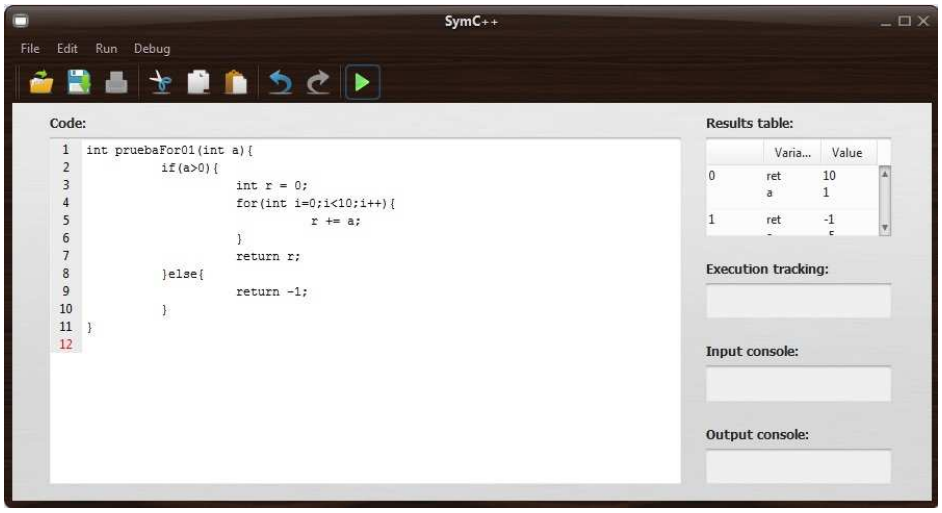

Figura A.7: Figura con la ventana principal de la interfaz con los resultados.

Figura A.8: Figura con la vista de la traza de ejecución.

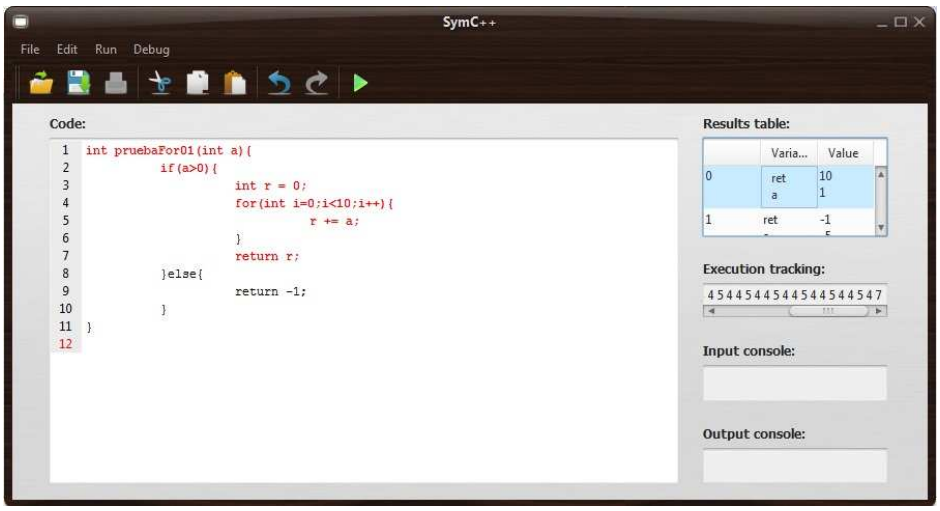

Figura A.9: Figura con la ventana principal de la interfaz con los resultados y la traza resaltada.

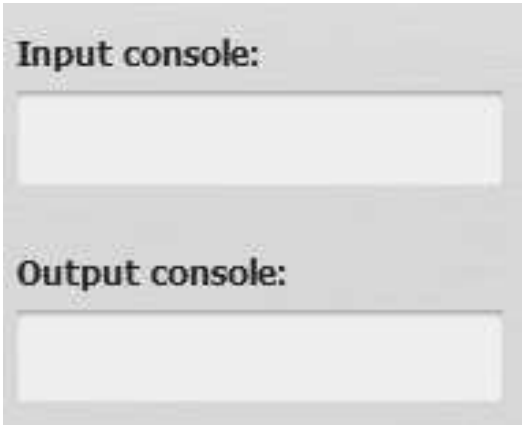

Figura A.10: Figura con la vista de las consolas de entrada y salida.

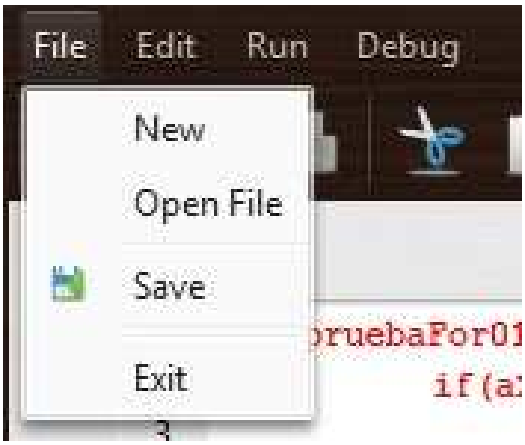

Figura A.11: Figura con la vista del submenu “file”.

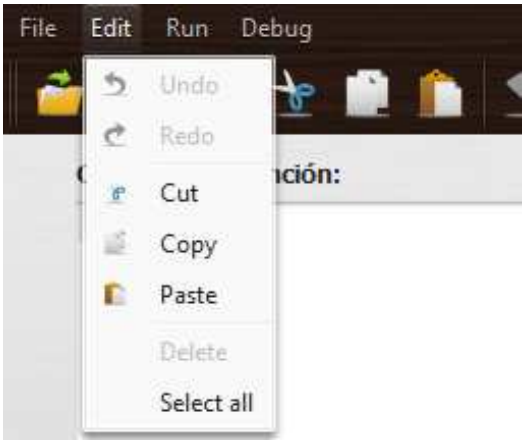

Figura A.12: Figura con la vista del submenu “edit”.

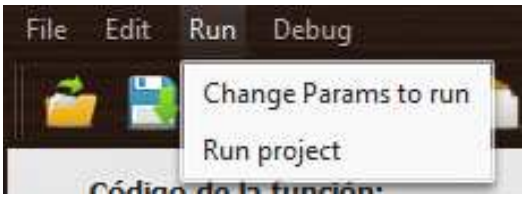

Figura A.13: Figura con la vistal submenu “run”.

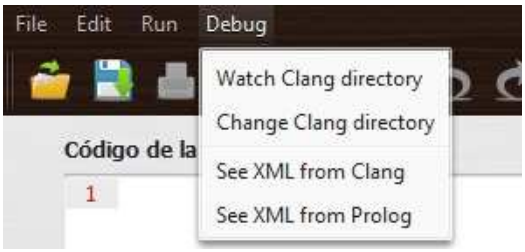

Figura A.14: Figura con la vista del submenu “debug”.

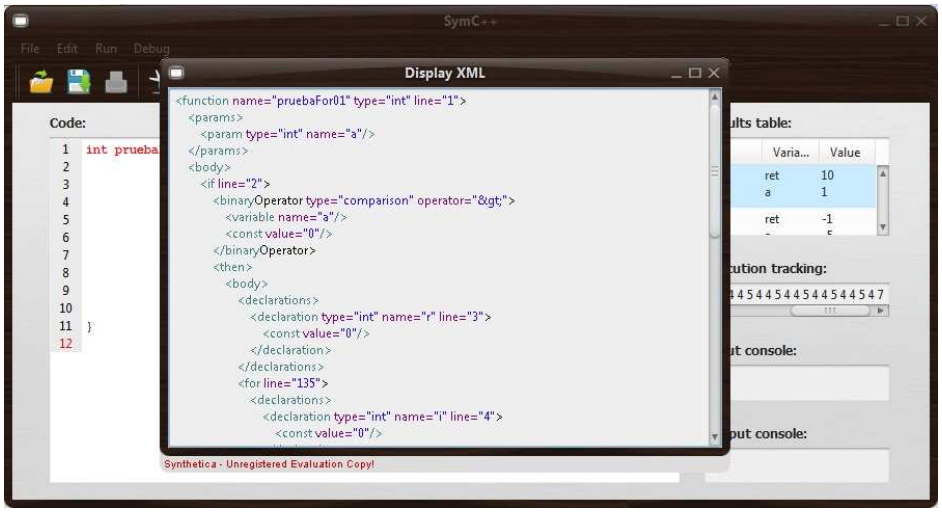

Figura A.15: Figura con la vista del XML generado.