Mòdul de suport a la docència per assignatures de programació

Roger Pera Martín

Resum— El propòsit d'aquest projecte és el d'ajudar a simplificar l'ensenyament de certs conceptes de programació que són difícils d'entendre per a alumnes de primer de carrera. Per aconseguir aquest propòsit és necessari l'ús d'un programa debugger, l'automatització de l'entrada de comandes d'aquest debugger en funció del programa a analitzar i, per últim, la visualització esquemàtica i didàctica dels resultats obtinguts. Tota aquesta simplificació serà realitzada amb una simple execució del projecte mitjançant un programa de control del DOS, que conté les diferents ordres per cridar de forma seqüencial a cada una de les parts del projecte.

Paraules clau— GDB, GCC, debugger, C++, software, memòria dinàmica, compilador, Scratch, línia d'ordres, DOS.

Abstract—
The purpose of this project is to help simplify the teaching of certain programming concepts that are difficult to understand for students at the first steps of their degree . To achieve this purpose it is necessary to use a debugger program, the automation of the debugger command entry aligning commands with the requirements depending on the program to be analyzed and, finally, a schematic and didactic display for the results. All this simplification will be made with a simple project execution through a DOS control program that contains different lines to call secuencially every one of the sections of the project.

Index Terms— GDB, GCC, debugger, C++, software, dinamic memory ,compiler, Scratch, order comand line, DOS.

1 Introducció

L'APRENENTAGE de la programació requereix passos d'abstracció que per un alumne de primers cursos poden ser difícils de realitzar. En particular, l'estudi de la memòria dinàmica i els punters requereixen un nivell de comprensió bastant alts que poden ser recolzats amb una visualització del procés, pas a pas.

Avui dia, en el treball a les aules s'està deixant de banda les classes magistrals per a incorporar les noves tecnologies i facilitar el treball dels alumnes a partir de casos pràctics en el qual es pugui veure la variabilitat que només amb les classes de teoria no es podria apreciar.

Per aquests dos motius sorgeix l'aplicació del mòdul de suport per a la docència per assignatures de programació, tant per ajudar als estudiants de primer curs a comprendre millor la programació a partir d'una eina de visualització del procés de programació amb la memòria dinàmica, com per a evolucionar en el món de la impartició d'assignatures de programació a partir de l'utilització d'eines d'ús més pràctic i no basar-nos només en la teoria.

La teoria impartida per explicar punters i memòria dinàmica necessita de dibuixos i representacions gràfiques que els alumnes posteriorment s'han de plantejar mentalment, alhora de desenvolupar el seu propi codi per a entendre que estan fent realment i com s'estan organitzant cadascun d'aquests. Aquesta ajuda mental els pot ajudar a organitzar millor o de manera més òptima el seu codi desenvolupat i, per tant, millorar el seu rendiment a l'hora de programar.

En aquest projecte oferim una eina que permet a l'alumne fer les proves necessàries per entendre, tant l'organització de la memòria dinàmica, com els punters en si i les seves variacions durant el codi. Aquesta ajuda es fa mitjançant una representació esquemàtica de com varia la memòria al llarg del codi. La presentació es fa el més semblant a l'explicada a teoria de primer curs d'aquest grau, és a dir, senzilla i intel·ligible. Així, l'alumne podrà utilitzar l'aplicació per fer-se la imatge mental de l'esquema explicat a teoria, però a temps real, tantes vegades com necessiti i amb tantes variants de codi com vulgui, ja que els codis seran els proposats pel mateix alumne.

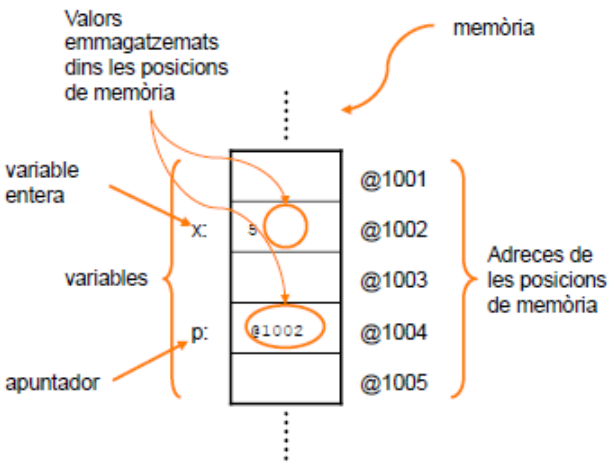

Figura 1. Esquema teòric memòria.

A la figura 1, podem veure el que seria l'esquema de la memòria explicada a teoria amb un punter que guarda l'adreça de la posició de memòria on apunta. A grans trets l'esquema mental que té un alumne sobre la memòria del programa seria un esquema molt similar o igual a aquest.

2 Estat de l'art

Al mercat actual existeixen un gran nombre d'aplicacions amb l'objectiu de millorar l'aprenentatge a la programació [1]. Eines com "Scratch" [2], són molt útils per ajudar a desenvolupar aplicacions bàsiques en un llenguatge de més alt nivell. Aquesta eina, té com a objectiu la difusió de la programació i ho fa simplificant-la per que qualsevol persona pugui programar de manera senzilla. La simplificació és realitzada a partir de la programació per blocs, cada comparació, bucle, etc. és tractat com un sol bloc, que conté un conjunt de codi per realitzar una funció concreta. Per exemple, l'usuari de l'eina visualitza com a bloc el resultat del que passa si clica un botó concret controlat pel programa, és a dir, "si clica el botó (passa això) sinó (allò)", sent una manera senzilla d'explicar el if/else. Dintre d'aquests blocs, l'usuari pot afegir més blocs del mateix estil, en llenguatge natural, i així anar desenvolupant el seu propi "codi", tal com es pot apreciar a la figura 2.

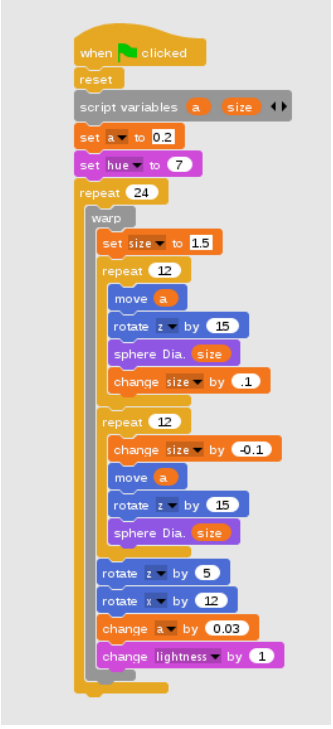

Figura 2. Programa realitzat amb Scratch.

Hi ha d'altres eines [3] com "PSeInt" que ajuda a comprendre el funcionament d'un programa bàsic però amb un pseudocodi propi, o eines que ajuden a aprendre programant, especialment jocs. Eines com "Processing" o "Gamekit", etc. abunden en el mercat, essent moltes d'elles de lliure ús.

L'inconvenient principal de totes aquestes aplicacions és que són molt senzilles, ja que estan especialment encaminades a l'ajuda a la iniciació de la programació, amb funcions molt bàsiques. Podem veure que utilitzen una metodologia massa simple i no arribaria a tractar, des de la perspectiva de l'usuari, temes com la memòria dinàmica o funcionament dels punters en el codi. És a dir, no és una eina vàlida per a l'anàlisi profund del funcionament d'un programa i, per tant, no ho és per a la impartició d'una assignatura de primer curs del grau en enginyeria informàtica.

Un altre inconvenient que ens ha obligat a buscar una altra solució és que, per un alumne d'iniciació, és molt important comprendre com es comporten les variables i els punters a cada pas d'un programa. Avui per avui, no he estat capaç de trobar cap eina que faciliti l'estudi de la memòria dinàmica de manera directa per tal de que els estudiants puguin visualitzar i comprendre els seus valors, tret dels mateixos programes de depuració lligats als compiladors. De totes maneres, els depuradors o "debuggers" estan més enfocats a la depuració d'errors en els programes que no pas a l'ús didàctic per tal d'entendre el funcionament del programa analitzat, malgrat que l'eina ho permet. En tot cas, busquem una eina que ens permeti fer un anàlisi del codi i tractar-lo de la manera que ens convingui per tal de fer comprensible el procés lògic que segueix el programa a analitzar. Al no trobar cap eina al mercat amb aquest enfoc, aquest projecte pretén esbossar un ús diferent dels debugguers mitjançant el desenvolupament de mòduls acoblats al cor principal del programa depurador, que ens doni com a resultat la informació ordenada de tal forma que permeti un seguiment didàctic del comportament del programa analitzat.

3 OBJECTIUS

L'objectiu principal d'aquest projecte és la implementació d'una eina que faciliti a l'estudiant l'enteniment de l'assignatura de programació, facilitant-li l'esquematització de parts crítiques com podrien ser l'organització de memòria i el funcionament dels punters. Per a tot això, s'han planejat diversos objectius més reduïts que acabaran satisfent l'objectiu principal del projecte una vegada s'hagin complert tots i cadascun d'aquests.

Aquest treball té un segon objectius que és el de demostrar que, fent servir una eina no pensada pel suport docent tal com és un programa depurador, es pot arribar, desenvolupant alguns mòduls nous, a aconseguir un resultat amb grans possibilitats didàctiques.

Com a conseqüència d'aquest segon objectiu, apareixen una sèrie d'objectius secundaris, lligats a la mateixa creació d'un mòdul de demostració de les possibilitats didàctiques del debugger. En aquest projecte he marcat els següents:

1. El més primordial és aconseguir els valors de les variables en cada etapa del programa que estem analitzant, ja que aquestes seran necessàries per a realitzar la visualització de la memòria dinàmica a mostrar a l'alumne.
2. Treballar amb el programa depurador sense haver d'entrar les comandes necessàries a mà, és a dir, que el debugger executi totes les comandes de cop sense haver d'introduir-les una per una.
3. Fer una visualització de la memòria dinàmica de manera esquemàtica i intel·ligible, una vegada s'hagin obtingut els valors del debugger.
4. Unir tots els mòduls per aconseguir que tot es materialitzi en una sola execució. És a dir, muntar un fitxer que executi tot el projecte seguint els passos que calgui.

4 PARTS DEL PROJECTE

En aquesta secció es detallarà el funcionament i l'enllaç de cadascuna de les parts del projecte amb les altres per tal de poder entendre el projecte com un conjunt únic.

Primer de tot cal dir que com a programa depurador hem utilitzat el debugger GDB en la seva versió 7. Ha estat l'escollit perquè, en primer lloc, és un depurador "open source", i malgrat que no forma part d'aquest projecte, una de les possibilitats per explotar el depurador de forma més didàctica és la de modificar directament aquesta eina per tal de que ens doni els informes o visions que s'adaptin a les nostres necessitats, sense problemes de royalties. A més, el GDB suporta la depuració de programes escrits en diferents llenguatges de programació, tals com C, C++, Java, Fortran, ADA etc. , el que el fa molt versàtil. Així, l'ús del GDB implica que els programes a analitzar han de ser compilats amb el compilador GCC, que també és una eina "open source" i suporta pràcticament els mateixos llenguatges de programació que el GDB, amb mòduls diferents per a cada llenguatge, i són distribuïts per la "Free Software Fundation" sota llicència general pública (GPL).

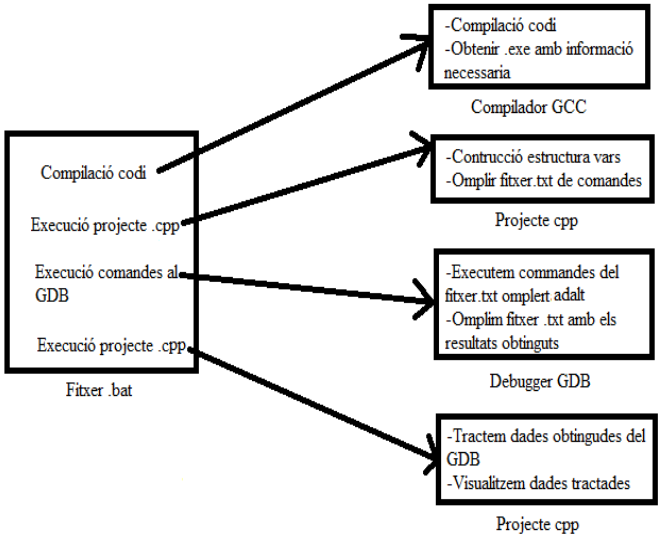

Figura 3. Esquema de l'execució del projecte [4].

Com es pot apreciar a la figura 3 de manera esquemàtica el programa de control anomenat "fitxer .bat", a l'esquerra, crida a totes i cadascuna de les parts del projecte. El que passa internament a cada part està explicat dins de les caixes de la dreta de l'esquema. Primerament el programa de control, crida al compilador GCC per tal de que compili el codi realitzat per l'alumne per tal d'obtenir l'executable necessari per treballar amb el debugger. Posteriorment, executa el codi del nostre projecte en C++ per omplir l'estructura on guardem les variables que existeixen en el codi i omple el fitxer de comandes "comandes.txt" que seran executades en el debugger GDB. Una vegada tenim el fitxer de comandes omplert i l'executable del codi de l'alumne, crida al debugger per que executi les comandes del fitxer sobre l'executable del programa de l'alumne, i tota la informació obtinguda pel debugger és bolcada sobre un fitxer de text "GeneracióGDB.txt". El programa de control torna a executar el nostre projecte en C++ per segona vegada per tal de tractar el fitxer de text "GeneracioGDB.txt", omplert pel debugger. Finalment el mateix programa C++ ens visualitza la memòria en cada pas de l'execució del codi.

Aquesta figura 4 s'explica de manera més detallada a les següents seccions.

4.1 Compilació del codi

Per poder tractar un codi sobre el debugger GDB és totalment necessari que, amb anterioritat, aquest s'hagi compilat i per tant, haguem obtingut el seu executable.

Aquest executable ha de ser creat amb el compilador

GCC, ja que aquest aporta un conjunt d'opcions que permeten guardar certes dades específiques en funció dels paràmetres de compilació que se l'hi passin. En el nostre cas, s'ha compilat el codi amb extensió .C amb l'opció –g del compilador [5], per poder obtenir tota la informació necessària que el debugger GDB exigeix per recollir les dades pertinents, que després extraurem. L'opció –g, en cridar el compilador, fa que l'executable guardi informació necessària per al depurador per tal de poder mostrar dades relatives al codi font, noms de variables i funcions, referències creuades, número de línia de cada instrucció al codi font, etc.

4.2 Construcció de l'estructura "variables"

La segona part desenvolupada del treball és un projecte realitzat amb C++ que tracta el codi realitzat per l'alumne com un text [6] per tal d'aconseguir totes les variables que es troben definides en ell. Totes les variables es guarden en una estructura que conté un conjunt de característiques segons el tipus, mida, etc. Concretament, a la figura 4, podem veure que els atributs de l'estructura creada són: Tipo de Dato, Tamaño de Dato, Nombre Variable, Valor Variable, Tamaño Array i si és apuntador o no.

El projecte llegeix el codi paraula per paraula per tal de trobar les paraules clau d'inicialització (int, float, short,...). Cadascuna d'aquestes paraules tenen assignades una grandària de dades [7], en nombre de bytes que, juntament amb el mateix nom del tipus de dades, són guardades en els atributs "Tipo Dato" i "Tamaño Dato" de l'estructura.

Una vegada trobem una d'aquestes paraules llegim la següent i la tractem de la manera convenient omplint els camps de l'estructura segons el tipus de variable que estem tractant. En el cas de que sigui un array també guardarem un "Tamaño Array" com un nombre major a 1 segons la grandària d'aquest i si ens trobem davant d'un punter li ficarem a "true" l'atribut "EsApuntador" de l'estructura mostrada a la figura 4.

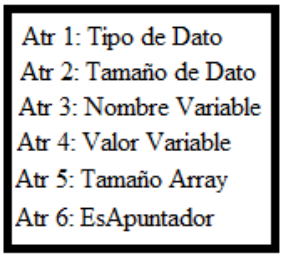

Figura 4. Estructura en la que es guarden les variables.

La necessitat de l'atribut "Tamaño Array" és deguda a que a l'hora de la visualització puguem saber quants espais de memòria reserva aquest array, per exemple, si tenim un array → array[3] , aquest reservarà 3 posicions de memòria, tots de la mateixa grandària, basant-nos en el tipus de dades pel qual està format i la grandària d'aquestes dades. Pel que fa a l'atribut "esApuntador", és necessari saber si ho és, ja que la comanda que nosaltres utilitzem al debugger GDB de "info locals", que s'explicarà més endavant, només ens mostraria l'adreça de l'apuntador, fet que no ens interessa, ja que seria un valor sense cap sentit per a la persona que estigues veien l'esquema de la memòria que visualitzem. A tot això, aquest atribut és utilitzat per afegir una part de codi al fitxer de comandes, executat pel GDB , referent al punter per tal de que rebem el valor de l'adreça de memòria on senyala l'apuntador i no la mateixa adreça. Per tant, podríem dir que és necessari en el cas dels apuntadors per tal de poder mostrar un valor intel·ligible per l'usuari que utilitza l'eina.

És necessari guardar totes les variables per poder veure com van canviant els valors de les mateixes en memòria per a cadascun dels passos que realitza el codi.

4.3 Ompliment del fitxer.txt de comandes

El debugger GDB té una característica que ens serà molt valuosa en aquest projecte. Aquesta característica és la possibilitat de cridar al debugger passant-li un fitxer text [8] amb totes les línies de comandaments que aplicaríem, un per un, com si estiguéssim depurant un programa de forma directa. Al passar el fitxer de comandes, el debugger es comporta executant una instrucció darrera l'altre de forma automàtica. Això ens permet la possibilitat de preprogramar aquelles comandes que volem que siguin executades pel debugger.

En el mateix projecte C++ anterior, on hem buscat les variables que utilitza el programa a analitzar, omplim un fitxer amb extensió text que contindrà les comandes necessàries per a poder obtenir del GDB els valors de dites variables a cada línia d'execució del programa.

El debugger GDB té un gran nombre de possibles comandes a realitzar per a obtenir dades del fitxer executat. Entre aquestes comandes hi ha la possibilitat de realitzar bucles i també preguntes condicionals [9] de forma que l'automatització de l'obtenció de dades sigui més òptima. És a dir, podem programar la manera en què el debugger es comportarà en funció de valors, condicions, bucles, etc..

En el nostre cas aprofitem aquesta possibilitat per a fer un bucle que vagi avançant pas per pas en el codi i així saber tots els valors de les variables en cada pas i conseqüentment podrem veure com van canviant aquestes en la continuïtat del codi.

Tot i que el debugger disposa d'una gran varietat de possibles comandes per executar, nosaltres només utilitzarem un nombre reduït d'aquestes [10], les quals seran explicades breument juntament amb la funcionalitat que els hi donem nosaltres en el nostre fitxer de comandes a executar pel GDB:

- Break: la seva versió reduïda seria "b". Aquesta comanda el que fa és crear un "break point" a la

línia o funció que nosaltres decidim. En el nostre projecte el que farem serà utilitzar la versió del "break function" i aquest estarà al principi de la funció "main" per tal de començar a analitzar les variables amb el debugger al principi del codi.

- Run: la seva versió reduïda seria "r". La comanda serà utilitzada per a començar l'execució del programa des del principi.

- Step: la seva versió reduïda seria "s". Per tal de poder continuar l'execució del debugger a partir del punt de ruptura main, existeixen diverses opcions. En el nostre projecte hem utilitzat la comanda step, ja que aquesta ens porta al següent pas d'execució sense la necessitat d'haver de crear un punt de ruptura en cada línia on vulguem consultar les variables del programa. La comanda "continue" seria una comanda molt semblant, però aquesta sí que necessitaria un punt de ruptura en cada línia per que el debugger parés i li poguéssim demanar les variables a mostrar.

- Info locals: Aquesta comanda el que fa és imprimir el valor de les variables locals del frame en el que ens trobem, justament el que necessitem en cada pas del programa. Tot i això, hi ha una variant que es podria utilitzar, aquesta seria la comanda "print nomVariable" que ens imprimiria el valor de la variable abans d'executar les instruccions de la línia en la qual ens trobem. L'inconvenient que té aquesta comanda és que li hauríem d'indicar totes les variables que volem que ens mostri en cada moment i per aquest motiu no la utilitzem, ja que no seria la manera més òptima que existeix pel nostre cas. Utilitzar la comanda "info locals" ens permet obtenir totes les variables locals i els seus valors sense la necessitat de tenir que demanar-les una per una. La comanda "print variable" ha sigut utilitzada en el cas dels punters, ja que la comanda "info locals" només ens dóna l'adreça a la que apunta el punter i no el valor que conté dita adreça.

- While: les comandes "info locals" i "step" s'executen dins d'un bucle, ja que nosaltres volem anar avançant per l'execució i obtenint els valors de les variables a cada pas, fins que el programa de l'usuari acaba.

- Quit: la seva versió reduïda "q". És la instrucció que utilitzem per sortir del debugger GDB.

- Error: En cas de que surti qualsevol error que bloquegi l'execució del debugger farem que surti del GDB. El tractament dels errors s'hi ha realitzat a partir d'una variable que fa aquest control.

```
1  b main
2  set $_exitcode = -1
3  r
4  search return
5  while main
6  s
7  if $_exitcode != -1
8  quit
9  end
10 info locals
11 end
12 q
13
```

Figura 5. Exemple de codi de Comandes del GDB.

La figura 5 mostra el codi de comandes per a omplir el fitxer Comandes.txt en cas de que el codi no contingui cap punter, ja que en cas d'existir apuntadors el codi seguiria un esquema gairebé idèntic però s'hauria d'ampliar amb unes línies per tractar de forma individualitzada a cadascun dels punters.

Com ja he dit amb anterioritat pel cas dels apuntadors no es pot utilitzar la comanda "info local", si volem obtenir el valor contingut a l'adreça que apunta, en comptes de la mateixa adreça que és el que retorna aquesta comanda. Per a obtenir el valor contingut a l'adreça de memòria indicada per l'apuntador s'ha d'utilitzar la comanda "print nomVariable", però aquesta, no es pot posar sense realitzar una comprovació de si l'adreça esta inicialitzada o no abans de demanar-la . Això s'ha de fer ja que no podem demanar al debugger GDB que mostri el valor d'una adreça que no existeix perquè no ha estat prèviament definida. Per exemple, quan definim el primer punter, aquest es guarda l'adreça 0X0. Per comprovar que està inicialitzada, en el fitxer de comandes comprovem que l'adreça de l'apuntador primer és 0X0 per poder demanar-li al debugger que ens mostri el valor de l'adreça a la qual senyala aquest primer apuntador.

4.4 Execució de les comandes al GDB

Una vegada compilat el codi amb el compilador GCC i amb l'opció –g, cridarem al debugger GDB, l'hi demanem que s'executi en "batch" per tal de no mostrar l'execució en pantalla, l'hi donem el nom del fitxer de text que hem creat i que conté les comandes a executar, i l'hi donem el nom del programa de l'alumne que volem que analitzi. Un cop executat el GDB obtindrem unes dades de sortida que seran bolcades directament a un altre fitxer de text [11], anomenat "GeneracioGDB.txt", que conté el resultat de les instruccions que el GDB ha executat.

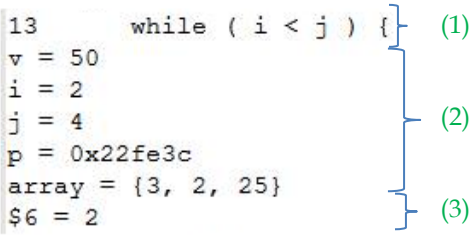

Figura 6. Exemple dadesde sortida del GDB.

Tal com es pot observar en l'exemple de la figura 6, el que el debugger ens retorna a la sortida és:

1- Ens mostra de forma conjunta el número de línia i el codi font contingut a dita línia del programa de l'alumne
2- El conjunt de variables locals que existeixen amb el seu valor, amb el format: "nomVarible = valor", junt amb l'adreça dels punters i els valors continguts dins dels arrays. Aquests valors es corresponen als valors de les variables abans d'executar la línia que apareix en l'apartat (1).
3- I finalment, ens mostra el valor contingut dins de l'adreça de cada punter, també abans de l'execució de la línia de l'apartat (1), però amb una singularitat especial. Cada vegada que mostra un valor l'assigna a un nom de variable del tipus $num. L'assignació del nom és seqüencial i l'haurem de tractar a posteriori per tal de saber a quin punter fa referència.

4.5 Tractament de les dades obtingudes del GDB

Tal com he comentat prèviament, el GDB ens haurà creat un fitxer anomenat "GeneracioGDB.txt". Aquest fitxer ha de ser tractat per organitzar la informació continguda i donar-l'hi un sentit més comprensible.

Si ens miréssim el contingut del fitxer veuríem que hem obtingut els valors de les variables, arrays, adreces de memòria dels punters i valors continguts a dites adreces en cada pas de l'execució del codi. Recordem que els valors obtinguts corresponen als valors previs a l'execució de la línia i, per tant, haurem de tractar el fitxer per mostrar els valors després de l'execució de cada línia. També hem de tractar l'assignació dels valors continguts a les adreces de memòria de cada punter per saber a quin punter corresponen. Tots aquests tractaments els realitzem amb el programa C++, que ubica tota la informació de forma estructurada per tal de ser visualitzada. El programa ens mostra a cada línia de codi els valors obtinguts. La idea de fer-ho en cada pas de l'execució és deguda a que el que volem arribar a veure és la variació de la memòria interna del codi analitzat. Es podrien analitzar més canvis produïts a cada línia de codi, però en aquest cas ens centrarem només a les variables, arrays i punters. El depurador ens dóna una àmplia gama de possibilitats respecte als diferents tipus d'informació que ens pot subministrar, des de la anàlisis de la memòria en cada moment, valors de variables d'entorn, bolcats de memòria, etc., el nostre objectiu és només mostrar algunes de les possibilitats que ens dóna el GDB.

Un aspecte que s'ha de tenir en compte a l'hora de tractar aquest fitxer és que el debugger GDB, en executar i tancar el codi, afegeix un text al final i al principi del fitxer de text on fem el bolcat que és totalment irrellevant per a nosaltres i que s'ha d'obviar. Per fer-ho buscarem el principi i el final del tractament de les variables i s'utilitzaran com a punts de referència per a realitzar correctament el tractament del fitxer.

4.6 Visualització de les dades tractades

A mesura que anem tractant les variables en cada pas de l'execució del codi i guardant el seu valor a la nostra estructura, aquestes es van visualitzant per pantalla de forma que l'alumne pot veure la variació en el valor d'aquestes i, així, anar entenent el que realment està passant internament en el codi.

Aquesta visualització es realitza el més intel·ligiblement possible per facilitar a la persona que executa el programa d'anàlisis, el no haver d'entendre una gran quantitat de conceptes o valors sense sentit, ja que cal recordar que aquest treball ha sigut creat per ajudar als estudiants de primer curs que encara no tenen un ampli coneixement del funcionament de la memòria interna de l'ordinador.

4.7 Unificació de les parts

Com ja he explicat amb anterioritat, la idea principal d'aquest projecte és aconseguir que la persona que està tractant amb l'aplicació pugui entendre millor el funcionament de la part interna del codi que ha realitzat. Per facilitar-l'hi la tasca, en lloc d'executar els diferents passos necessaris per a realitzar l'anàlisi, m'he posat com a objectiu el que es pugui fer amb nomes una sola acció per part de l'estudiant.

Per tal d'aconseguir-ho, tots els diferents processos s'han agrupat dins d'un programa de control DOS d'extensió .bat que realitza totes les crides de les parts que es farien a la línia d'ordres [12]. D'aquesta manera l'usuari només haurà de modificar el seu codi, guardar-lo, executar el .bat i veure com varia la memòria dinàmica del codi introduït per ell. Es poden veure totes les crides que realitza aquest fitxer a la figura 2.

5 METODOLOGIA

Per tal d'aconseguir els objectius marcats al projecte s'ha desenvolupat seguint la metodologia incremental [13]. Aquesta metodologia consisteix a desenvolupar una arquitectura inicial del projecte i, a partir d'aquí, anar afegint funcionalitats implementades i testejades, al sistema inicial.

En el cas d'aquest projecte el que s'ha realitzat ha sigut un esquelet inicial del codi i a partir d'aquí, s'ha anat refinant i millorant a partir de noves funcionalitats. El progrés d'aquest projecte era totalment seqüencial, és a dir, el projecte no podia avançar si les parts anteriors no estaven realitzades. Tot i que la realització de l'esquelet inicial va durar més temps del previst, es van poder plantejar correctament les següents etapes incrementals que desenvoluparien les parts conseqüents del projecte, aconseguint així seguir de manera idònia amb la metodologia proposada.

Un avantatge que té aquesta metodologia [14], és que una vegada desenvolupada l'arquitectura inicial, que suposa una gran quantitat de temps, es poden generar parts de software de manera ràpida i molt flexible, podent afegir millores de forma continuada. Tot i això, l'inconvenient que té aquesta metodologia és que si ocorre un error a l'arquitectura inicial a fases tardanes del desenvolupament es pot retardar de manera molt considerable la consecució de resultats útils.

6 RESULTATS

Com a resultat final d'aquest projecte, s'ha aconseguit visualitzar la memòria dinàmica de manera esquemàtica i fàcil d'entendre per l'alumne.

```c
#include <stdio.h>
int main(void)
{
    int v = 50;
    int i=1;
    int j=4;
    int *p=&i;

    int array[3];
    array[0]=3;
    array[1]=2;
    array[2]=array[0]+array[1];
    int *w=&j;

    v=v+*w;

    while ( i < j ) {
       i += 1;
     }
     i=1;
 printf("Resultado: %i\n", v);

 return 0;
}
```

Figura 7. Codi utilitzat per a la visualització de resultats.

A la figura 7 es mostra un codi senzill d'exemple, com el que pot desenvolupar qualsevol alumne de primer de carrera, i que prova de manera simple la majoria de conceptes que s'han volgut tractar en el desenvolupament del projecte. És a dir, tractament d'arrays amb operacions simples, utilització de punters, bucles, variables, etc. Tot aquests conceptes modifiquen els seus valors durant l'execució del codi per tal de poder veure el comportament de la memòria dinàmica.

L'estructura de la visualització de les dades és en format taula. A sobre de la taula s'imprimeix la línia d'execució en la qual ens trobem per la memòria mostrada.
Aquesta taula té dos separadors verticals, és a dir, 3 quadrants en els quals tenim:

1- Primerament, el nom de la variable.
2- Al segon quadrant trobem el valor que té la variable, que en el cas dels punters és el valor de l'adreça a la qual apunten.
3- I en l'últim quadrant, el nombre de bytes que ocupa la variable, en el cas dels arrays és la grandària d'un dels elements d'aquest.

Quan als separadors horitzontals, el que separen és cadascuna de les variables que es troben en memòria en aquest instant de l'execució.

```
4                int v = 50;

| v | 50 |4Bytes |
5                int i=1;

| v | 50 |4Bytes |
| i | 1 |4Bytes |
6                int j=4;

| v | 50 |4Bytes |
| i | 1 |4Bytes |
| j | 4 |4Bytes |
| p | 0x0 |4Bytes |
7                int *p=&i;

| v | 50 |4Bytes |
| i | 1 |4Bytes |
| j | 4 |4Bytes |
| p | 1 |4Bytes |
| array[0] | 1 |4Bytes |
| array[1] | 0 |4Bytes |
| array[2] | 4200185 |4Bytes |
10               array[0]=3;

| v | 50 |4Bytes |
| i | 1 |4Bytes |
| j | 4 |4Bytes |
| p | 1 |4Bytes |
| array[0] | 3 |4Bytes |
| array[1] | 0 |4Bytes |
| array[2] | 4200185 |4Bytes |
11               array[1]=2;

| v | 50 |4Bytes |
| i | 1 |4Bytes |
| j | 4 |4Bytes |
| p | 1 |4Bytes |
| array[0] | 3 |4Bytes |
| array[1] | 2 |4Bytes |
| array[2] | 4200185 |4Bytes |
12               array[2]=array[0]+array[1];

| v | 50 |4Bytes |
| i | 1 |4Bytes |
| j | 4 |4Bytes |
| p | 1 |4Bytes |
| array[0] | 3 |4Bytes |
| array[1] | 2 |4Bytes |
| array[2] | 5 |4Bytes |
| w | 1 |4Bytes |
```

Figura 8. Mostrem la memòria al principi del codi.

A la figura 8, podem veure com la memòria inicialment només té el valor de "v" i a posteriori es van afegint a mesura que va sortint en el codi. També podem veure que a l'execució de la línia 10 del codi els dos últims valors de l'array estan inicialitzats en un valor per defecte que ens retorna el debugger, però a mesura que es van inicialitzant els seus valor es van actualitzant fins i tot en el cas de fer una operació amb els anteriors elements d'aquest.

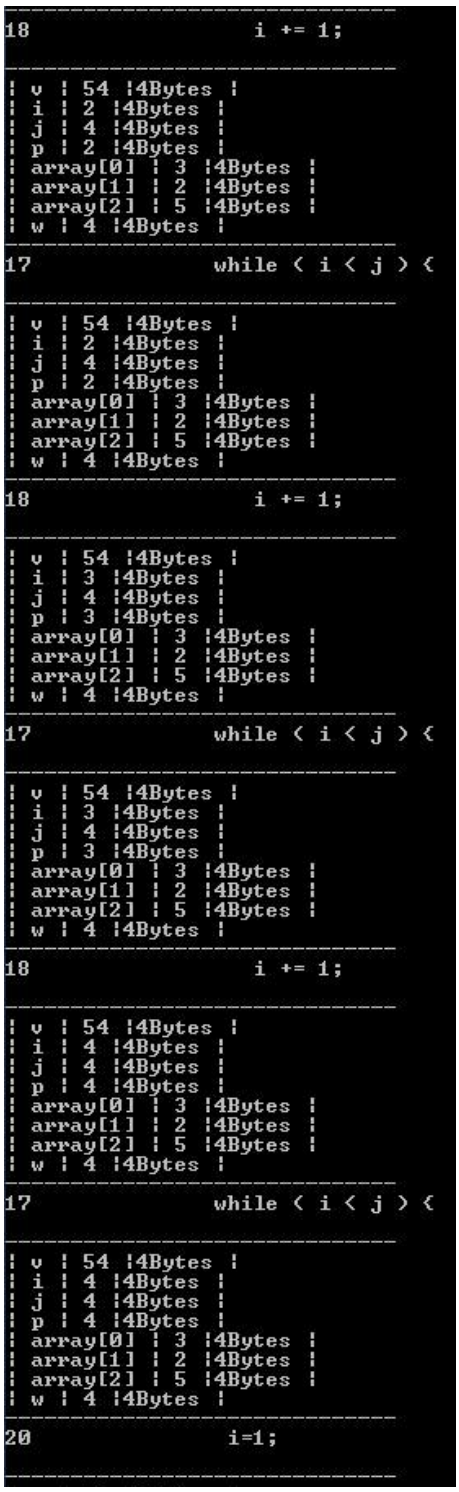

Figura 9. Mostrem la memòria a la part mitja del codi

En el cas de la figura 9 he agafat la variació de la memòria durant el "while" per tal de comprovar el funcionament correcte al treballar sobre un bucle. Podem comprovar que el bucle s'executa i les línies del programa es van repetint, mentre les variables van canviant, fins que s'arriba al final del bucle i el programa continua.

Figura 10. Mostrem la memòria final del codi.

A la figura 10 es mostra el resultat final de la memòria després haver passat per totes les línies del codi d'exemple executat.

7 CONCLUSIÓ

En aquest projecte hem vist com utilitzar un depurador com a base per extreure informació de l'execució d'un programa i tractar la informació resultant per a utilitzar-la de forma didàctica. El projecte només pretén ser un punt de partida, ja que la informació extreta es pot anar ampliant amb petites modificacions del codi par afegir noves comandes pel debugger escollit. L'avantatge d'utilitzar el GDB és triple, per una banda és software lliure, per un altre té una gran quantitat de comandes que es poden fer servir, i per últim ens permet treballar contra diferents llenguatges de programació d'ús habitual. El resultat és una base ampliable amb nous mòduls, que per si ja ens dóna un resultat atractiu que compleix amb tots els objectius primaris i secundaris que ens hem marcat a l'inici. Addicionalment, a mesura que he anat treballant amb el GDB he pogut comprovar que, la informació que és capaç de reportar, pot ser utilitzada d'infinitat de maneres diferents amb múltiples propòsits. Jo m'he centrat en el propòsit didàctic per alumnes de primer curs de programació, però la informació disponible ens permetria d'ampliar el projecte per crear mòduls per alumnes avançats que vulguin analitzar de forma complexa l'evolució de la memòria dinàmica durant l'execució d'un programa, no només basant-nos en els valors més bàsics. Podríem analitzar variables d'entorn, control d'errors, dumps del programa, etc. És més, podríem també analitzar el funcionament en programes anidats dins d'altres programes, i veure el funcionament de la memòria tant al programa pare com al programa fill, de forma conjunta. Com es pot veure, les possibilitats són enormes.

AGRAÏMENTS

Agraeixo a la meva tutora Aura Hernández Sabaté per haver dirigit i supervisat el meu projecte. Donar les gràcies per la seva implicació alhora de la cerca d'eines per a la correcta resolució d'aquest, i pel temps dedicat a proporcionar-me qualsevol ajut.

BIBLIOGRAFIA

[1] Exemples de programes realitzats amb scratch
http://edutec.citilab.eu/

[2] Projectes realitzats amb scratch
https://scratch.mit.edu/

[3] Altres eines que ajuden a aprendre a programar
http://www.techsupportalert.com/content/best-free-ways-learn-programming.htm
http://www.portalprogramas.com/pseint/

[4] Primers passos amb GDB
http://www.lsi.us.es/~javierj/ssoo_ficheros/GuiaGDB.htm

[5] Flag –g del compilador gcc
http://stackoverflow.com/questions/5179202/gcc-g-what-will-happen

[6] Tractament de strings
http://www.cplusplus.com/reference/cstring/

[7] Tipus i grandària de variables en C
http://maxus.fis.usal.es/fichas_c.web/01xx_PAGS/0101.html

[8] Executar fitxers de comandes amb GDB
https://sourceware.org/gdb/onlinedocs/gdb/Command-Files.html

[9] Manual GDB
https://sourceware.org/gdb/onlinedocs/gdb/

[10] Comandes del depurador GDB
http://www.eis.uva.es/~fergay/III/enlaces/gdb.html

[11] Carregar dades a un fitxer.txt des de el programa de control DOS
https://social.technet.microsoft.com/Forums/es-ES/11354d91-1e90-43be-9dc3-5f0dae6dbac9/comando-ping?forum=wcxpes

[12] Comandes programa de control DOS
http://www.taringa.net/post/hazlo-tu-mismo/13687424/Archivos-BAT-Comandos-MS-DOS.html

[13] Que és la metodologia incremental
https://procesosoftware.wikispaces.com/Modelo+Incremental

[14] Model de la metodologia incremental
https://es.wikipedia.org/wiki/Metodolog%C3%ADa_de_desarrollo_de_software#Incremental