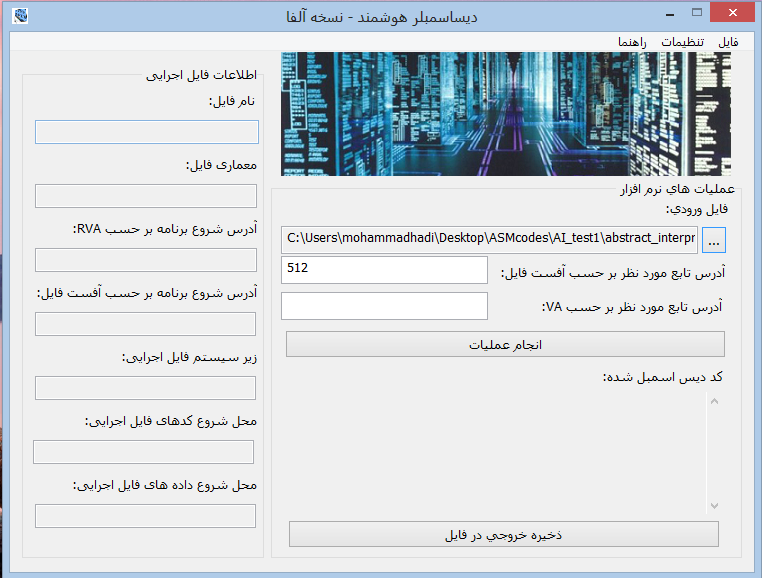
# برنامه دیس اسنبلر برای اینکه پرشهایی که به درون مقداری که در حافظه یا در ثبات‌ها است را به دست آورد.

این برنامه همان طور که در شکل 1 نیز مشاهده می‌شود شامل 3 بخش است. بخش اول شامل اطلاعاتی عمومی در مورد فایل PE و ساختار آن است که فقط برای نمایش به کاربر مد نظر قرار گرفته است. بخش دوم برای واردکردن نشانی فایل و اجازه گرفتن دستور برای دیس هم گذاری کردن است. با دادن نشانی در این قسمت و با تلیک کردن بر روی دکمه انجام عملیات کار تحلیل آغاز می‌شود. لازم به توضیح است که بعد از اتمام عملیات جهت چاپ خروجی که در قسمت 3 نشان داده می‌شود، می‌توان با تلیک کردن بر روی ذخیره خروجی در فایل، این کار را انجام داد. با تلیک کردن این مورد محیطی برای انتخاب محل نشانی ذخیره‌سازی باز می‌شود و بعد از انتخاب به طور خودکار کار ذخیره‌سازی انجام می‌شود.

شکل 1 برنامه دیس اسنبلر

در قسمت منو ها همان طور که ملاحظه می شود در 3 منو وجود دارد یکی با نام فایل است که در آن 2 منو دیگر وجود دارد که یکی برای باز کردن فورمی برای انتخاب فایل اجرایی و دیگری خروج از برنامه وجود دارد.

در قسمت راهنما دو زیر منو وجود دارد که یکی برای راهنمای نرم افزار و دیگری درباره نرم افزار می باشد.

منو تنظیمات شامل 5 زیر منو است که 3 تای اول برای انجام عملیات ساختن گراف جریان کنترلی بسیار اهمیت دارد توجه داشته باشید که وجود این 3 تیک کنار این 3 زیر منو حتما وجود داشته باشد زیرا که اگر وجود نداشته باشد قسمت تفسیر انتزاعی آن کار خواهد کرد.

# طریقه طراحی

## ایجاد گراف جریان کنترلی

برای ایجاد گراف جریان کنترلی 5 کلاس در نظر گرفته شده است.

1. کلاس instruction برای نگهداری اطلاعات هر دستور
2. کلاس block برای نگهداری یک بلاک که شامل تعدادی از دستورات و بلاک های ورودی به این بلاک و خروجی از این بلاک است.
3. کلاس lable برای نگهداری اطلاعات مورد نظر هر lable. مشخص می کند که لیبل مورد نظر به کدام بلاک اشاره می کند و چندمین دستور آن بلاک می باشد.
4. کلاس codepair وظیفه نگهداری دستورات اسنبلی را بر عهده دارد.
5. کلاس CFGD که کلاس کنترلی برای کلاس های دیگر است.

### کلاس instruction

این کلاس وظیفه نگهداری از اطلاعات یک دستور و مشخص می کند که چه متغیری را این دستور استفاده یا تغییر می دهد را دارد. متغیر های این تابع بصورت دو نوع public و private در جدول زیر مشخص شده اند.

جدول 1 جدول متغیرهای کلاس instruction

|  |  |  |  |
| --- | --- | --- | --- |
| نام متغیر | نوع دسترسی | نوع متغیر | توضیحات |
| Opcode | public | string | Opcode دستور را در خود ذخیره می کند |
| D | public | vector<string> | متغیر های که در این دستور تعریف می شوند در این متغیر ذخیره می شود. |
| U | public | vector<string> | متغیر های مورد استفاده در این دستور را در خود ذخیره می کند |
| Inivar | public | vector<string> | تمام متغیر هایی که در این دستور چه مورد استفاده قرار می گیرند و چه تعریف می شوند در این دستور قرار می گیرند. |
| Valid | private | Bool | اگر دستور معتبر نباشد این متغیر مقدار false را به خود می گیرد. |
| ini | Private | string | کل جمله دستور را در خود نگهداری می کند. |
| code | private | string | کد باینری دستور را در خود نگهداری می کند. |
| linemumber | private | int | شماره خط دستور را در خود نگه داری می کند. |
| codesindex | private | vector<codepair\*>\* | مجموعه کد هایی که برای تعیین def-use مورد استفاده قرار می گیرد را در خود نگهداری می کند. |

حال بعد از تعریف متغیر ها می توان توابع موجود در این کلاس را در جدول شماره 2 مشاهده کرد.

جدول 2 توابع کلاس instruction

|  |  |  |
| --- | --- | --- |
| نام تابع | نوع دسترسی | توضیحات |
| Instruction (string ins, int line, vector <codepair\*>\* codesindex); | public | این تابع سازنده این کلاس است که دستور و شماره خط و کل دستورات اسنبلی را به عنوان ورودی دریافت می کند. |
| ~instruction(); | public | این تابع ویرانگر این کلاس است |
| void print\_inis(); | public | این تابع برای نمایش خروجی در کنسول است. |
| string to\_write\_inis(); | public | این تابع برای نمایش خروجی در فایل است که یک جمله برای خروجی را ایجاد کرده و در خروجی قرار می دهد |
| string to\_write\_iniss(); | public | ایجاد کننده اطلاعات برای نمایش بصورت گرافیکی |
| bool isBranch(string\* branches, int len); | public | می گوید که آیا این دستور شرطی است یا خیر. |
| bool islableform(); | public | آیا این دستور در خود لیبل دارد یا خیر |
| string getlablename(); | public | اگر در این دستور لیبل وجود دارد آن را در خروجی قرار می دهد در غیر این صورت مقدار "" را در خروجی ظاهر می کند. |
| string gotowhere(string\* branches,int len, int\* returnindexbranches); | public | اگر دستور شرطی را پیدا کرده اید این تابع می گوید این شرط به کجا می پرد در صورت درست بودن. این تابع مجموعه شرط ها را می گیرد و تعداد این شرط ها و خروجی محل پرش است و returnindexbranches ایندکس مورد نظر را بر می گرداند. |
| bool isa\_Valid\_instruction(); | public | مقدار متغیر valid را بر میگرداند |
| void calculateDefandUseID(); | public | با فراخوانی این تابع مقدار های متغیر های U, D, inival ایجاد می شود. |
| void loadinstruction(string ins); | public | در این قسمت متغیر های این دستور از هم جدا شده و در کل بعد از ساخت یک شی از این کلاس این تابع فراخوانی خواهد شد. |
| string deletespace(string s); | public | این تابع برای پاک کردن فاصله ها مورد استفاده قرار می گیرد. یک جمله را گرفته و فاصله ها را پاک می کند و آن را در خروجی قرار می دهد. |
| bool isDef(string s); | public | یک متغیر را می گیرد می بیند که آیا قبلاً این متغیر را در مجموعه متغیر هایی که در D است قرار دارد یا خیر |
| bool isUse(string s); | public | یک متغیر را می گیرد می بیند که آیا قبلاً این متغیر را در مجموعه متغیر هایی که در U است قرار دارد یا خیر |
| void deleteComments(); | public | بعضی از دستورات اسنبلی ممکن است کامنت گذاشته شده باشند که این دستور وظیفه خذف آن ها را دارد. و تاثیر این تابع بر روی متغیر ini است. |
| void deletesamenameinVector( vector<string> X); | public | یک تابع برای اینکه متغیر های مشترک در یک مجموعه را پاک کند. |
| void setDEFUSE(bool use,bool def,string arg); | private | متغیر org را دریافت کرده و اگر تعریف باشد در مجموعه D و اگر متغیر مورد استفاده باشد در مجموعه U قرار می دهد و در هر دو حالت در inival قرار میگیرد. |
| string getString(vector<string> s); | private | یک مجموعه از جمله ها را دریافت کرده و در خروجی یک جمله ایجاد کرده و بر می گرداند. |
| vector<string> split\_string(string s,char c); | private | یک جمله را دریافت کرده و با کاراکتر C آن را از هم جدا می کند و همه را در یک مجموعه از جملات قرار می دهد. |
| vector<string> split\_string\_with\_E(string s); | private | با مشاده کاراکتر های \*، /، =، + جمله ها را از هم جدا می کند. |
| string split\_first\_string(string s,char c); | private | با مشاهده اولین C از ابتدا تا آنجا را بر میداد و در صورت پیدا نشده مقدار  "nofound" را بر می گردانند. |
| int getindex(); | private | با استفاده از متغیر codeindex ایندکس مورد نظر این opcode را برای تحلیل متغیر های استفاده و تعریف پیدا می کند. |
| bool isID(string s); | private | آیا این متغیر بصورت یک متغیر یا رجیستر است یا خیر |
| bool isptr(string s); | private | آیا این متیغر بصورت اشاره گر است یا خیر |

### کلاس codepair

این کلاس برای نگهداری اطلاعات مربوط به انواع دستورات مورد استفاده قرار می گیرد. این کلاس دارای دو متغیر است. یکی string opcode که برای نگهداری نام دستور و index که شماره نوع عملکرد سیستم است که باید برای ایجاد متغیرهای تعریف و استفاده در تابع calculateDefandUseID مورد استفاده قرار بگیرد.

از این کلاس شی هایی در تابع loadOPCodes در کلاس CFGD ایجاد می شوند. تا در کلاس instruction مورد استفاده قرار بگیرد.

### کلاس lable

این کلاس برای نگهداری اطلاعات یک برچسب است. در این کلاس 3 متغیر نگهداری می شوند. یکی string name که نام برچسب را خود ذخیره می کند و دیگری block\* blockL است که خود بلاکی را که این برچسب در آن ذخیره شده را در خورد نگه داری می کند. int blockini نیز برای نگهداری شماره دستوری که در این بلاک وجود دارد است.

### کلاس block

این کلاس برای نگهداری اطلاعاتی در مورد basic block است. هر شی از این کلاس شامل دستوراتی است که در این بلاک قرار می گیرند و مجموعه بلاک هایی که به این بلاک وارد می شوند و بلاک هایی که از این بلاک خارج می شوند.

متغیر های این کلاس را در جدول 3 ملاحظه می کنید.

جدول 3 متغیر های کلاس block

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| نام متغیر | نوع دسترسی | | نوع متغیر | توضیحات |
| Bname | Private | String | | نگهدارنده نام بلاک |
| insbuf | Private | vector<instruction\*> | | نگهدرنده مجموعه دستورات این بلاک |
| \_IN | Private | vector<block\*> | | نگهدارنده بلاک هایی که به این بلاک وارد می شوند. |
| \_OUT | private | vector<block\*> | | نگهدارنده بلاک هایی که این بلاکها به آن خارج می شود. |
| \_OUTTEXT | private | vector<string> | | نگهدارنده نوع و نام خروجی که انجام می دهد. ایندکس های این متغیر با ایندکس های متغیر \_OUT باید یکی باشد. یک جفت هستند. |
| Issamplevalue | private | Bool | | در ساخت basic block ها، به مواردی برخورد می شود که بعد از پیدا محلی که باید به آن پرش انجام شود مشاهده می شود که محل این پرش در بازه کدی که پیدا شده و جود نداد یا پرچسبی که پرش به آن داده می شود پیدا نشده است در این صورت این متغیر برای آن بلاک به مقدار true مقدار گزاری می شوند. |

درتوابع این کلاس نیز در جدول شماره 4 مشخص شده است.

جدول 4 توابع کلاس block

|  |  |  |  |
| --- | --- | --- | --- |
| نام تابع | نوع دسترسی | | توضیحات |
| block(); | | public | تابع سازنده این کلاس است |
| block(string Blockname); | | public | تابع سازنده این کلاس است که نام این بلاک را نیز با خود در یافت می کند و مقدار Bname را برابر مقدار ورودی قرار می دهد. |
| ~block(); | | public | تابع ویرانگر این کلاس است. |
| void add\_new\_instruction(instruction\* ins); | | public | دستور جدیدی را به انتهای دستور های قبلی اضافه می کند. |
| void print\_ins\_out(); | | public | برای نمایش در کنسول مورد استفاده قرار می گیرد. |
| string write\_to\_ins\_out(); | | public | تحت عنوان یک جمله کل بلاک را در اختیار فراخوان کننده قرار می دهد جهت چاپ و یا ... |
| instruction\* init(int index); | | public | دستور ایندکس مورد نظر را می دهد. باید دقت شود که خارج از بازه نباشد. |
| instruction\* first\_init\_pop(); | | public | اولین دستور را از مجموعه دستورات برداشته و آن را در خروجی قرار می دهد. دیگر در این بلاک وجود نخواهد داشت. |
| instruction\* init\_pop\_at(int index); | | public | دستور ایندکس مورد نظر را می دهد. و آن را از مجموعه دستورات این بلاک حذف می کند. باید دقت شود که خارج از بازه نباشد. |
| int size(); | | public | تعداد دستورات موجود در این بلاک را بر میگرداند. |
| bool iniisemply(); | | public | مقدار حاصل از متغیر Issamplevalue را بر می گرداند. |
| void add\_new\_OUT(block\* b,string text); | | public | این تابع بلاک جدیدی را در صورت نبودن آن در مجموعه اضافه می کند و نوع فراخوانی را نیز تحت یک جمله دریافت کرده و با هم به ترتیب در \_OUT و \_OUTTEXT قرار می دهد. |
| void add\_new\_IN(block\* b); | | public | این تابع بلاک جدیدی را در صورت نبودن آن در مجموعه \_IN اضافه می کند. |
| string getname(); | |  | مقدار Bname را بر می گرداند که نام بلاک در آن ذخیره شده است |
| void DELETERELATIONIN(); | | public | تمام بلاک هایی که به این بلاک وارد میشدند، ارتباط بین آنها را حذف می کند یعنی دیگر آن بلاک ها به این بلاک وترد نمی شوند. |
| void deleteout(string blockname); | | public | خروجی از این بلاک به بلاکی که نام آن از ورودی دریافت می شود را خذف می کند. |
| string getlablename(); | | public | برچسب اولین دستور را بر می گرداند. |
| void clone\_pop\_OUT(vector<block\*>\* A); | | public | یک مجموعه را دریافت کرده و از آن یک کپی در \_OUT قرار می دهد و بعد کل مجموعه اولیه را که در A بوده پاک می کند. |
| vector<block\*>\* getOUT(); | | public | \_OUT را بر می گرداند. |
| vector<string>\* getOUTTEXT(); | | public | \_OUTTEXT را بر می گرداند. |
| vector<block\*>\* getIN(); | | public | \_IN را بر می گرداند. |
| void setsample(); | | public | مقدار Issamplevalue را true می کند. |
| bool issample(); | | public | مقدار Issamplevalue را بر می گرداند. |
| void clearout(); | | public | مجموعه های \_OUT و \_OUTTEXT را خالی می کند. |
| void clearoutelse(string elsestring); | | public | همه بلاک های موجود در مجموعه \_OUT و \_OUTTEXT را خالی می کند مگر نوع پرش آن با نوع ورودی که مشخص شده یکی باشد آ نها را پاک نمی کند. |
| void Copyoutsinto(block\* nb); | | public | مقدار \_OUT و \_OUTTEXT این بلاک را در بلاک ورودی اضافه می کند. لازم به توجه که با تغییر کردن \_OUT، \_IN بلاکی که در \_OUT بوده نیز تغییر میکند. |
| void Copyoutsintoelse(block\* nb,string elsestring); | | public | مقدار \_OUT و \_OUTTEXT این بلاک را در بلاک ورودی اضافه می کند مگر نوع پرش آن با نوع ورودی که مشخص شده یکی باشد آ نها را پاک نمی کند. لازم به توجه که با تغییر کردن \_OUT، \_IN بلاکی که در \_OUT بوده نیز تغییر میکند. |
| void change\_continue\_OUT\_to(string text); | | public | نوع پرش هایی که به صورت continue بوده را به نوع ورودی تبدیل می کند. |
| block\* getOUTContinue(); | | public | بلاکی که در مجموعه \_OUT قرار دارد و نوع پرش آن continue است را بر می گرداند. |
| void remove\_IN(string Bname); | | public | بلاکی که نام آن با نام ورودی به این تابع یکی است را از مجموعه حذف می کند. |
| void remove\_OUT(string Bname); | | public | بلاکی که نام آن با نام خروجی از این تابع یکی است را از مجموعه حذف می کند. |
| bool jumpisfor(string s); | | public | در صورت اینکه نوع opcode آخرین دستورالعملی که در این بلاک وجود دارد با ورودی یکی بود مقدار true را بر می گرداند در غیر این صورت مقدار false را بر می گرداند. |
| bool samejumpisfor(string s); | | public | در صورت اینکه نوع opcode آخرین دستورالعملی که در این بلاک وجود دارد با ورودی مشابه بود مقدار true را بر می گرداند در غیر این صورت مقدار false را بر می گرداند. |
| block\* getOUTfor(string call); | | public | نوع پرشی که با پرش وروی یکی باشد را بر می گرداند. |
| vector<block\*> getOUTsfor(string call); | | public | نوع پرشی هایی که با پرش وروی یکی باشد را بر می گرداند. |
| void Delete\_IT\_return\_IN(); | | public | زمانی که در بلاکی در اواسط آن بدلیل وجود پرشی می خواهد بلاک را برش بدهد زمانی که پرش قبلی این بلاک از نوع call بوده باشد باید IN های این بلاک پاک شوند و به بلاک بعدی منتقل داده شوند که این تابع این کار را انجام می دهد. |
| bool isnewerthan(block \* b); | | public | در صورت این بلاک از بلاک ورودی شماره آن بزرگتر باشد مقدار درست در غیر این صورت مقدار نادرست را بر می گرداند. |
| bool notfound(vector<block\*> bs,block\* b); | | private | آیا در مجموعه بلاک های ورودی بلاک b وجود دارد در صورت پیدا شده مقدار false و در صورت پیدا نشدن مقدار true را برمی گرداند. |

### کلاس CFGD

این کلاس که یک کلاس کنترلی برای قسمت ساختن گراف جریان کنترلی است. با فراخوانی تابع DOCFG و دادن متغیرها به آن می توان کار را آغاز کرد. و خروجی را می توان یا از طریق

1. متغیر bbuf که مجموعه از basic block های لینک شده به هم هستند در نظر گرفت.
2. یا می توان خروجی را با استفاده از تابع write\_to\_file در فایل ریخت.
3. با استفاده از تابع getallbb می توان آن چیزی که در تابع write\_to\_file در خروجی خود در فایل قرار می دهد را به عنوان رشته از کاراکترها که بصورت ورودی دریافت کرده به فراخوان زننده بدهد.
4. یا از طریق تابع print\_block\_out می توان در کنسول خروجی در فایل ریخته شده را مشاهده کرد.

متغیر های کلاس CFGD در جدول شماره 5 مشخص شده اند.

جدول 5 متغیر های کلاس CFGD

|  |  |  |  |
| --- | --- | --- | --- |
| نام متغیر | نوع دسترسی | نوع متغیر | توضیحات |
| bbuf | public | vector<block\*> | مجموعه بلاک های ایجاد شده را در خود نگهداری می کند. |
| labels | public | vector<lable\*> | مجموعه برچسب ها را در خود نگهداری می کند |
| Branches | public | String\* | مجموعه دستورات شرطی را در خود نگهداری می کند |
| Bnum | public | Int | تعداد دستورات شرطی را در خود نگهداری می کند |
| Wtofile | public | Bool | در صورت true بودن خروجی را در فایل می ریزد. |
| stocons | public | bool | در صورت true شدن خروجی را در کنسول نمایش می دهد |
| fileforout | private | string | آدرس فایل خروجی را در خود نگهداری می کند |
| Fileforgvout | private | String | آدرس جهت ذخیره سازی فایل برای ایجاد فایل pdf را در خود نگهداری می کند. |
| Codes | private | vector<codepair\*>\* | مجموعه دستورات زبان اسنبلی را در خود نگهداری می کند |
| Pdfnumbercfg | private | Int | تعداد فایل های pdf ای را که این کلاس ساخته را در خود نگهداری می کند. |
| branchesset | private | Bool | در صورت اینکه مجموعه دستورات شرطی آماده شده باشند این متغیر مقدار درست را به خود می گیرد. |
| Codesset | private | bool | در صورتی که دستورات زبان اسنبلی آماده شده باشند این متغیر مقدار درست را به خود می گیرد. |

توابع این کلاس نیز در جدول 6 مشخص شده اند.

جدول 6 توابع کلاس CFGD

|  |  |  |
| --- | --- | --- |
| نام تابع | نوع دسترسی | توضیحات |
| CFGD(void); | public | تابع سازنده این کلاس |
| ~CFGD(void); | public | تابع ویرانگر این کلاس |
| void DOCFG(char\*,bool); | public | تابعی که ساختن گراف جریان کنترلی را آغاز می کند. دستورات به حالت مجموعه ای از کاراکترها به این کلاس داده می شود عملیات ایجاد گراف آغاز می شود. |
| void print\_block\_out(vector<block\*> inss); | public | کل مجموعه به همراه بلاک ها را درکنسول نمایش می دهد. |
| void makefilenameforout(); | public | برای خروجی آدرس را ایجاد می کند. |
| void Do\_Blocking(vector<block\*>\* inss); | public | بعد از ایجاد شناسایی دستورات همه آنها در یک بلاک به این تابع داده می شود و این تابع است که کار ایجاد بلاک بندی را انجام می دهد. |
| void setbranchkind(); | public | این تابع مقدار های اولیه متغیر Branches را مقدار دهی می کند. |
| block\* get\_block\_lable(string \_name,vector<block\*>\* inss,bool docontinue); | public | بلاکی را که برچسب \_name دارد را به عنوان خروجی نمایش می دهد. در صورت یافت نشدن مقدار NULL را در خروجی قرار می دهد. |
| void write\_to\_file(vector<block\*> inss); | public | برای نمایش در فایل و ذخیره در فایل مورد استفاده قرار می گیرد. |
| bool sep\_block(block\* b,int startindex,block\* tonb); | public | برای شکستن یک بلاک به دو بلاک های b و tonb از شماره دستور startindex به بعد است. |
| void clear(); | public | برای پاک سازی مقدار های گراف جریان کنترلی |
| void getallbb(char\*); | public | برای دریافت همه دستورات بلاک شده تحت عنوان یک مجموعه کاراکتر که اشاره گر آن از طریق ورودی به این تابع آمده باشد. |
| vector<block\*> TraceForFindReturn(block\* blocklabel); | private | برای بلاک هایی کهک فراخوانی انجام می دهند باید جایی که انتهای فراخوانی می رسد نیز پیدا شود. این کار را این تابع انجام می دهد.  از آنجایی که یک تابع ممکن است چندین ret داشته باشد بصورت مجموعه ای از بلاک ها بر می گرداند. |
| void makeGraphinPDF(); | private | به فایل سازنده pdf دستور ساخت را می دهد. |
| void merge2vector(vector<block\*>\* v1,vector<block\*> v2); | private | دو تا مجموعه را دریافت می کند و آن ها را با هم مخلوط می کند و در مجموعه اولی قرار می دهد در صورت اینکه قبلاً در این مجموعه نبوده باشند. |
| void makefilenameforgvout(); | private | آدرس ایجاد فایل .gv را مشخص می کند |
| long StringHexTointConverter(string hex); | private | برای تبدیل یک جمله از بصورت عدد hex به long مورد استفاده قرار می گیرد. |
| vector<string> split\_string(string s,char c); | private | یک جمله را دریافت کرده و با کاراکتر C آن را از هم جدا می کند و همه را در یک مجموعه از جملات قرار می دهد. |
| void loadOPCodes(); | private | برای مقدار دهی اولیه کردن متغیر Codes مورد استفاده قرار می گیرد. |
| void MakeGVfile(vector<block\*> inss); | private | برای ساختن pdf نیاز به ایجاد فایل .gv است. این تابع این عمل را انجام می دهد. |
| string maketruelable(string s); | private | مقدار خالص یک برچسب که آدرس آن باشد را بر می گرداند. اگر ابتدا و انتهای ان فاصله و یا کاراکتر h نوشته شده باشد را حذف می کند. |
| string getnextbnameoutsilde(); | private | برای ایجاد بلاک ها باید به آنها اسمی داد این تابع نام را برای این بلاک هایی که simple هستند ایجاد می کند. |
| string getnextbname(); | private | برای ایجاد بلاک ها باید به آنها اسمی داد این تابع نام را برای این بلاک هایی که simple نیستند ایجاد می کند. |

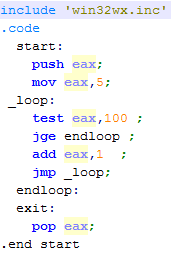
در این قسمت برای انجام آنالیز بر روی داده ها آنجام می شود تا آنالیز آنها انجام شود. برای بدست آوردن آنایز بازه ابتدا موارد روال زیر را اجرا می کنیم.

# تحلیل ایستای برنامه ها با استفاده از تفسیر انتزاعی با استفاده از راهبرد نقطه ثابت

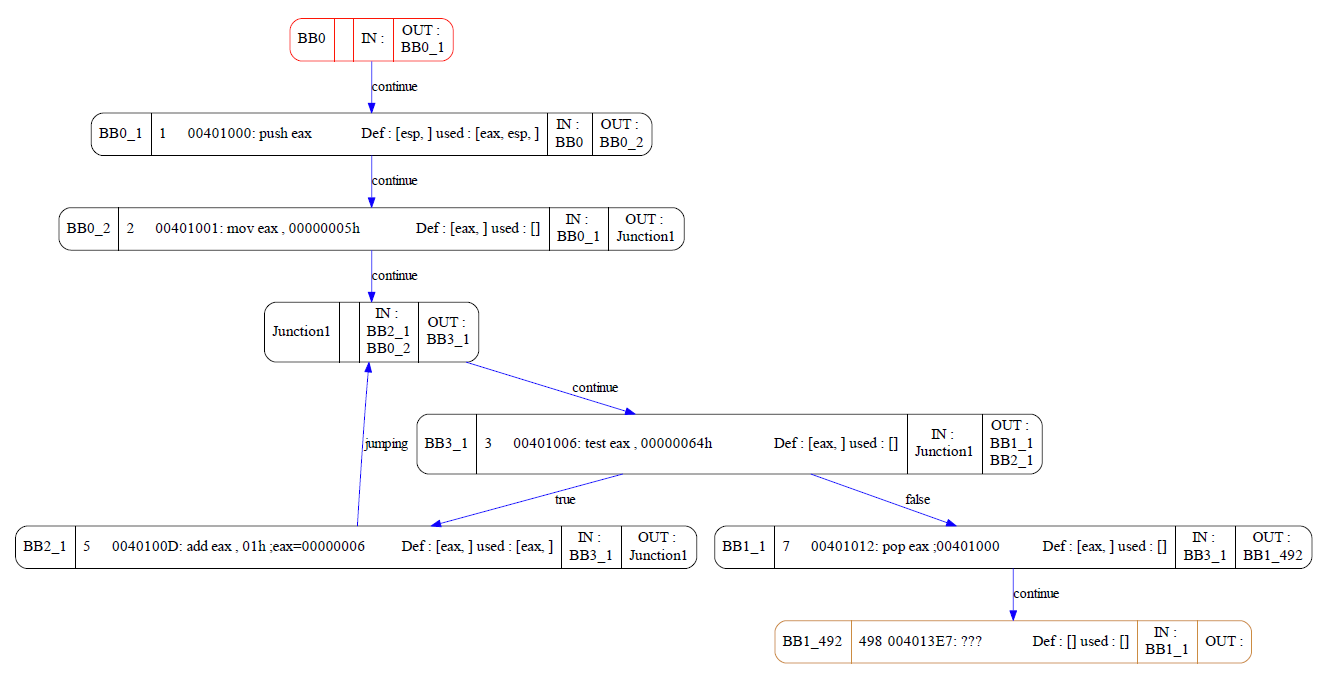
برای انجام این کار نیاز به بخش هایی داریم. برای اینکه نتیجه کار ما باید برای برنامه های دیس اسمبل شدن است. نیاز است زبان تقریبی را، شبهه زبان اسمبلی در نظر گرفت.

همه ما می دانیم که زبان های اسمبلی را می توان به بلاک های پایه ای تقسیم کرد که هر یک از این بلاک ها توصیفات خاص خود را دارند. برای این منظور، این توصیفات را به 5 قسمت تقسیم می شود. “Entries”, “Assignments”, “Tests”, “Junctions”, “Exits”.

* بلاک ورودی یا start یا “Entries” به بلاکی گفته می شود که هیچ ورودی ندارد و یک خروجی دارد.
* بلاک “Assignments” به بلاکی گفته می شود که یک ورودی و یک خروجی دارد.
* بلاک “Tests” به بلاکی گفته می شود که یک ورودی دارد و 2 خروجی دارد که یک در صورت درست بودن شرط و دیگری برای نادرست بودن شرط است.
* بلاک “Junctions” به بلاکی گفته می شود که در آن ورودی های به این بلاک بیشتر از یک و یک خروجی دارد. این بلاک در خود دستوری ندارد بلکه گلوگاهی است برای ورودی یک بلاک دیگر.
* بلاک “Exits” یک ورودی دارد و هیچ خروجی ندارد.

برای نمونه در عکس 1 قطعه کدی به زبان اسمبلی نوشته شده است و با استفاده از کامپایلر Flat Assembler، فایل اجرایی تولید شده است.

عکس 1 کد نمونه، نوشته شده در زبان اسمبلی

بعد از بدست آوردن گراف جریان کنترلی این قطعه کد و تبدیل این گراف به گراف مورد نظر برای تحلیل ایستا در عکس 2 نشان داده شده است. همان طور که در شکل نیز مشاهده می شود، هر بلاک فقط شامل یک دستور باشد. بلاک BB0 نقطه آغازین برنامه و این بلاک از نوع ورودی است. بلاک BB1\_492 بلاکی است که برای خاتمه در نظر گرفته شده است. بلاک junction1 نیز بلاکی است که به دلیل وجود چند ورودی به بلاک BB3\_1 ایجاد شده است. این بلاک همان طور که ملاحضه می شود هیچ دستوری در آن وجود ندارد. بلاک BB3\_1 یک بلاک Test است و دارای دو یال خروجی است که یکی در صورت درست بودن شرط و دیگری در صورت نادرست بودن شرط به وجود می آید. مابقی بلاک ها نیز جزء بلاک های عملیاتی و تساوی هستند که با نام Assignments، معرفی شده اند.

عکس 2 نمودار گراف جریان کنترلی برای قطعه کد موجود در عکس 108D0C9EA79F9BACE118C8200AA004BA90B02000000080000000E0000005F005200650066003300360033003300330034003300300036000000 .

اگر هر بلاک اولیه را یک Node در نظر بگیرید. یک مجموعه از این بلاک های اولیه را نیز Nodes نام گزاری می کنید. مجموعه Arcs را با عضو های نیز به عنوان یال های این گراف جریان کنترلی تعریف می شود. . برای نمونه برای گراف جریان کنترلی بالا مجموعه بصورت زیر می شود.

در این صورت گرافی خواهید داشت با راس های Nodes و یال های Arcs. در این صورت دوباره می توان همان گراف جریان کنترلی موجود در عکس 2 را به وجود آورد.

برای ادامه توابعی نیز بکار خواهند رفت که بر روی مجموعه Arcs اعمال می شود. یکی از این توابع بدست آوردن ورودی های و خروجی های هر نود یا بلاک اولیه است. به ترتیب این موارد را با نام a\_proc و a\_succ نام گزاری می کنیم. توابع دیگری نیز با نام های a\_succ\_t برای بدست آوردن خروجی هایی از یک بلاک Test که در صورت درست بودن اجرا می شود و a\_succ\_f برای بدست آوردن خروجی هایی از یک بلاک اولیه Test که در صورت نادرست بودن اجرا می شود، تعریف می شود.

تا این جای تحلیل، بلاک های اولیه ای قابل فهم توسط ماشین ایجاد شده است. لازم به توضیح است که هر یک از این بلاک های اولیه در خود یک state دارند که وضعیت برنامه را در خود حفظ می کند. بنابراین با کنار هم قرار دارد این state ها مجموعه با نام states به وجود می آیند.

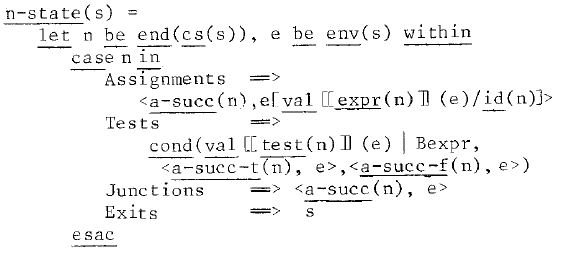
همان طور که گفته شده هر state نشان دهنده وضعیت هر دستور و محیط اطراف آن می باشد. اگر این جمله را بخواهیم به صورت فرمال نشان دهیم تعریف می شود.

در داخل هر محیط مقدار هر متغیر در داخل کد تا آن لحظه اجرایی قرار داده می شود. پس محیط اینگونه تعریف می شود که: محیط یعنی هر متغیر در این لحظه چه مقدار یا مقدارهایی دارد.

در ادامه به تابعی نیاز پیدا خواهید کرد با نام Cond این تابع 3 ورودی دارد. Cond(b,e1,e2) این تابع گفته می شود که اگر مقدار Boolean b به شما در این محیط ارسال شود خروجی ان چه محیطی خواهد بود. در حالت کلی خروجی این تابع برابر یکی از , e1, e2, می شود. در شرایطی b برابر با , true, false, باشد. در حالت کلی در شرایطی که هیچ مشکل نباشد این تابع بصورت زیر تعریف می شود.

If b then return e1 else return e2

در ادامه به تابعی نیاز پیدا می کنیم که با نام n-state تعریف می شود. این تابع کار این را دارد که در هر state مشخص کند که چه state دیگری بعد از این دستور اجرا خواهد داشت. این تابع دارای 4 بخش است.

1. اگر این بلاک از نوع Assignments باشد state بعدی خروجی این بلاک یکی می باشد و آن هم succ این بلاک می باشد. محیط آن نیز انجام عملیات برای این بلاک است.
2. اگر بلاک از نوع Tests باشد، در این صورت از تابع Cond استفاده می کنیم.
3. در بلاک از نوع Junctions باشد، در این صورت همان مقدار محیط حال را برای ادامه در نظر می گیریم.
4. در صورتی که بلاک از نوع Exits باشد، در این صورت این بلاک، state بعدی بخواهد داشت.

عکس 3 شبهه کد برای بدست آوردن ادامه یک بلاک. مشخص کردن یک اینکه بلاک بعدی چه محیطی را خواهد داشت بکار می رود.

لازم به توجه است که در همه این توابع باید شرایط های , هستند را باشد در نظر داشته باشید.

فرض کنید که مجموعه با نام I-state داریم که state های مقداردهی یا initial state را در خود داشته باشد. این مجموعه که شامل هر state با محیطی است که متغیری دارد که مقدار آن معلوم نیست.

یک دنباله ای از محاسبات داریم که با initial state یا یک دنباله است.

*این یعنی اگر یک تابع را با نماد را در نظر بگیرید و این نماد همانی بودن این تابع را تعریف کند در این صورت می شود.*

*اگر بخواهیم از state اولیه به state نهایی برسد باید این state ها را دنباله گیری کند.*

# قسمت های پیاده سازی شده

برای این منظور فعلا به غیر از کلاس های instruction و block و CFG که به عنوان پروژه های پیشین به اجرا در آمده است کلاس هایی با نام Range و Abstract\_instrunction پیاده سازی شده اند.

## کلاس Range

قسمت header این کلاس در زیر قابل مشاهده است. این کلاس برای تعریف کردن بازه هر متغیر مورد استفاده قرار می گیرد. این کلاس همان طور که در زیر نیز مشاهده می شود خصوصیات و عملیات های بازه ای را شامل می شود.

|  |
| --- |
| #pragma once  #include <limits>  class Range  {  public:  Range(void);  Range(double \_Min\_value,double \_Max\_value);  ~Range(void);  bool hasonevalue(); // return ((Max\_value - Min\_value) == 0)  Range\* operatorUP(Range x); // paper : ref ->abstract interpretation : a unified model for static analysis of programs by construction or approximation of foxpoints. cousot  Range\* operatorDOWN(Range x); // paper : ref ->abstract interpretation : a unified model for static analysis of programs by construction or approximation of foxpoints. cousot  Range\* operatorADD(Range x);  Range\* operatorSUB(Range x);  Range\* operatorMUL(Range x);  Range\* operatorDIV(Range x);  Range\* operatorUNION(Range x);  Range\* operatorSUBESCRIPTION(Range x);  void set(double \_Min\_value,double \_Max\_value);  void set(double Max\_Min\_value); // max = min = max\_min;  double get\_\_Max\_value();  double get\_\_Min\_value();  bool valueisdef(){  return (\_valueisdef);  }  Range\* copy();  private:  double \_\_\_min(double x,double y);  double \_\_\_max(double x,double y);  bool \_valueisdef;  double Max\_value,Min\_value;  }; |

این کلاس برای انجام کار عملیات بر روی بازه ها بسیار مورد استفاده است. در ادامه گزارش ها به موارد استفاده این کلاس بیشتر توضیح داده خواهد شد.

## کلاس Abstract\_instrunction

کلاس Abstract\_instrunction نیز به عنوان کلاس اصلی برای انجام کار تحلیل ایستا مورد استفاده قرار میگیرد. در زیر کد مربوط به بخش header این قسمت نیز ملاحظه می شود.

|  |
| --- |
| class Abstract\_interpretation  {  public:  Abstract\_interpretation(void);  Abstract\_interpretation(vector<block\*>\* \_basicBlocks,vector<codepair\*>\* \_opcodes);  ~Abstract\_interpretation(void);  void setBasicBlocks(vector<block\*>\* \_basicBlocks);  void setOpcodes(vector<codepair\*>\* \_opcodes);  int start\_Abstract\_interpretation();  vector<block\*>\* getACopyofBasicBlocks();  vector<block\*>\* foreachinstructionmakeabasicblock(vector<block\*>\* bb);  void make\_pdf(vector<block\*>\* b);  void MakeGVfile(vector<block\*> inss);  void setbranchkind();  private:  //vector<Node\_b\*>\* getarcs(vector<block\*>\* for\_each\_ini);  string\* branches;  int bnum; // this set the number of branches String \* [?];  vector<block\*>\* basicBlocks; /\*همه بلاک های اولیه را در خود جای می دهد.\*/  vector<codepair\*>\* opcodes; /\*همه آپکد های زبان اسمبلی را در خود ذخیره دارد\*/  int PDFNUMBER;  }; |

در این کلاس به عنوان ورودی بلاک های اولیه را دریافت می کند. بعد با فراخوانی تابع start\_Abstract\_interpretation کار تحلیل ایستا برای این بلاک های اولیه اجرا می شود. در ابتدا با استفاده از تابع getACopyofBasicBlocks یک کپی از این بلاک اولیه می گیرد برای این که بلاک اولیه از دست نرود. بعد با کپی این بلاک ها عملیات تفسیر انتزاعی انجام می گردد.

دراین قسمت برای انجام کار تفسیر انتزاعی باید بلاک ها را به بلاک های یک دستوری ایجاد کرد و برای انجام کار باید نوع بلاک را تعیین کرد این کار را تابع foreachinstructionmakeabasicblock انجام می دهد. در زیر کد مربوط به این تابع را مشاهده می کنید.

این تابع در خود 5 حلقه for جدا از هم دارد که به اختصار هر کدام را توضیح می دهیم.

1. حلقه اول برای هر دستور یک بلاک ایجاد می کند و تا جایی که ممکن است ورودی و خروجی این بلاک را تعیین می کند.
2. حلقه دوم برای بدست آوردن خروجی های هر بلاکی است که مقدار خروجی آن در حلقه اول درست تعیین نشده است.
3. حلقه سوم برای بدست آوردن ورودی های هر بلاکی است که مقدار ورودی آن در حلقه اول درست تعیین نشده است.
4. حلقه چهارم برای بدست آوردن بلاک های Junction است و ورودی ها و خروجی های به آن را نیز مرتب می کند.
5. در نهایت حلقه پنجم وجود دارد که برای تعیین حالت بلاک های اولیه مورد استفاده قرار می گیرد.

|  |
| --- |
| vector<block\*>\* Abstract\_interpretation::foreachinstructionmakeabasicblock(vector<block\*>\* basicBlocks){  vector<block\*>\* ret = new vector<block\*>();  vector<string> firstlabel;  for(int i=0;i<basicBlocks->size();i++)  {  block\* p = basicBlocks->at(i);  block \*lastb = NULL;  for(int j =0 ;j<p->size();j++)  {  if(p->init(j)->getinstructin().find("add byte ptr [eax] , al")!=string::npos && p->size()>50){  continue;  }  //////////////////////////////////////////make block name  string blockname = p->getname();  blockname.insert(blockname.length(),"\_");  char c[20];  \_itoa\_s((j+1),c,sizeof(c),10);  blockname.insert(blockname.length(),c);  block \* b = new block(blockname);  /////////////////////////////////////////block maked;  instruction \* ini = p->init(j)->Copy();  int index ;  ini->gotowhere(branches,bnum,&index);  b->add\_new\_instruction(ini);  if(j==0 && p->size()-1 ==0){  for(int k=0;k<p->getIN()->size();k++)  b->add\_new\_IN(p->getIN()->at(k));  firstlabel.push\_back(b->getname());  for(int k=0;k<p->getOUT()->size();k++)  b->add\_new\_OUT(p->getOUT()->at(k),p->getOUTTEXT()->at(k));  if(lastb!=NULL){  lastb->add\_new\_OUT(b,CONTINUE);  b->add\_new\_IN(lastb);  }  }  else if(j==0)  {  for(int k=0;k<p->getIN()->size();k++)  b->add\_new\_IN(p->getIN()->at(k));  firstlabel.push\_back(b->getname());  }  else if(j == p->size()-1)  {  for(int k=0;k<p->getOUT()->size();k++)  b->add\_new\_OUT(p->getOUT()->at(k),p->getOUTTEXT()->at(k));  lastb->add\_new\_OUT(b,CONTINUE);  b->add\_new\_IN(lastb);  }  else{  lastb->add\_new\_OUT(b,CONTINUE);  b->add\_new\_IN(lastb);  }    if(index<0){  ret->push\_back(b);  lastb = b;  }else {  if(lastb!=NULL){  lastb->remove\_OUT(b->getname());  b->Copyoutsinto(lastb);  if(j==0){  firstlabel.pop\_back();  firstlabel.push\_back(lastb->getname());  }  }  }  }  }  for(int i=0;i<ret->size();i++)  {  block\* b = ret->at(i);  for(int j=0;j<b->getOUT()->size();j++){  if(string :: npos == b->getOUT()->at(j)->getname().find("\_"))  {  string s = b->getOUT()->at(j)->getname();  int value = -1;//atoi(s.c\_str());  for(int u=0;u<basicBlocks->size();u++)  {  if(basicBlocks->at(u)->getname().compare(s)==0){  value = u;  break;  }  }  if(value>=0){  string newname = firstlabel.at(value);  for(int r=0;r<ret->size();r++){  if(ret->at(r)->getname().compare(newname)==0)  {  if(i>230){  int x;  x=0;  }  b->getOUT()->insert(b->getOUT()->begin(),ret->at(r));  b->getOUT()->erase(b->getOUT()->begin()+j+1);  break;  }  }  }  }  }  }  for(int i=0;i<ret->size();i++)  {  block\* b = ret->at(i);  for(int j=0;j<b->getIN()->size();j++){  if(string :: npos == b->getIN()->at(j)->getname().find("\_"))  {  string s = b->getIN()->at(j)->getname();  int value = -1;//atoi(s.c\_str());  for(int u=0;u<basicBlocks->size();u++)  {  if(basicBlocks->at(u)->getname().compare(s)==0){  value = u;  break;  }  }  if(value>=0){  string newname = firstlabel.at(value);  for(int r=0;r<ret->size();r++){  if(ret->at(r)->getname().compare(newname)==0)  {  b->getIN()->erase(b->getIN()->begin()+j);  b->add\_new\_IN(ret->at(r));  j--;  break;  }  }  }  }  }  }  firstlabel.clear();  make\_pdf(ret);  int tempblocknumber=0;  int ret\_size = ret->size();  for(int i=0;i<ret\_size;i++){  if(ret->at(i)->getIN()->size()>1){  tempblocknumber++;  char c[20];  \_itoa(tempblocknumber,c,10);  string blockname = "Junction";  blockname.insert(blockname.length(),c);  block\* temp = new block(blockname);  ret->at(i)->Copyinsinto(temp);  ret->at(i)->clearin();  ret->at(i)->add\_new\_IN(temp);  temp->add\_new\_OUT(ret->at(i),"continue");  ret->push\_back(temp);  }  }  block\* start = new block("BB0");  ret->push\_back(start);  if(ret->size()>1){  start->add\_new\_OUT(ret->at(0),"continue");  ret->at(0)->add\_new\_IN(start);  }  for(int i=0;i<ret->size();i++)  {  if(ret->at(i)->pred()->size()==1 && ret->at(i)->succ()->size()==1)  ret->at(i)->blocktype = 2; // assignment  else if(ret->at(i)->pred()->size()==0 && ret->at(i)->succ()->size()==1)  ret->at(i)->blocktype = 1; //entry point  else if(ret->at(i)->pred()->size()==1 && ret->at(i)->succ()->size()==2)  ret->at(i)->blocktype = 3; //tests  else if(ret->at(i)->pred()->size()>1 && ret->at(i)->succ()->size()==1)  ret->at(i)->blocktype = 4; //junctions  else if(ret->at(i)->pred()->size()==1 && ret->at(i)->succ()->size()==0)  ret->at(i)->blocktype = 3; //exit  }  return ret;  } |