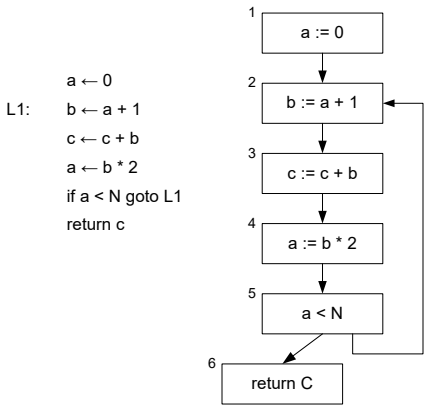
Анализа животног века

Теоријске основе

Да би се омогућило да различите промењљиве које нису истовремено у употреби деле исти ресурс, неопходна је информација о животном веку промењљиве. Животни век промењљиве започиње дефиницијом промењљиве (упис почетне вредности), а завршава се последњом употребом (последње очитавање садржаја промењљиве). Промењљива је у неком сегменту жива ако садржи вредност која ће бити коришћена касније у току извршења програма. Први корак у анализи је конструисање графа тока управљања.

Граф тока управљања

Граф тока управљања се конструише тако што се свака инструкција представи чвором графа, а затим се за свака два чвора *x* и *y*, такве да *x* претходи *y*, повлачи стрелица од *x* ка *y*.



Слика 1 Једноставан пример графа тока управљања

Животни век промењљивих тече преко стрелица графа, па се из тог разлога одређивање опсега живота своди на проблем тока података.

У граф тока се уводе додатне информације да би се омогућила једноставнија анализа. Чвор графа тока има излазне стрелице које воде до чворова наследника и улазне стрелице које воде од чворова претходника. Дефинишу се скупом свих чворова претходника pred[n] и скупом свих чворова наследника succ[n]. Даље се дефинише скуп промењљивих def[n] које чвор n дефинише, тј. задаје им вредност (исто што и *destination*), као и скуп промењљивих use[n] које чвор n користи, тј.очитава (исто што и *source*).

Промењљива је жива на стрелици графа ако постоји усмерена путања од те стрелице до чвора који користи ту промењљиву, а да путања не прелази преко иједног чвора који дефинише исту промењљиву. Промењљива је жива на улазу чвора ако је жива на било којој улазној стрелици чвора (скуп in[n]), а жива је на излазу у случају да је жива на било којој излазној стрелици чвора (скуп out[n]). Информација о животу промењљивих се може изразити на основу def[n] и use[n].

Алгоритам анализе животног века промењљивих

Скуп out[n] је унија „in“ скупова свих наследника чвора n, односно унија свих промењљивих s које припадају скупу „in“ сваког наследника чвора n. Скуп in[n] чине све промењљиве из скупа use[n] плус промењљиве које јесу у out[n], али нису у def[n]. Одатле следе једначине тока података за анализу животног века промењљивих:

out[n] ← $U_{s \in succ[n]}$in[s]

in[n] ← use[n] U (out[n]-def[n])

Решење ових једначина се добија следећим алгоритмом, исказаним у псеудокоду:

for each n

in[n] ← {}; out[n] ← {}

repeat

for each n

in'[n] ← in[n]; out'[n] ← out[n][1]

out[n] ← $U_{s \in succ[n]}$in[s]

in[n] ← use[n] U (out[n]-def[n])

until in'[n] = in[n] and out'[n] = out[n] **for all** n

Петља псеудокода „**for each** n" означава да се тело петље мора извршити за свако **n** (у овом случају за сваку инструкцију, тј. чвор графа тока), али не одређује којим редоследом. И заиста, да би се добило решење једначина животног века није важно којим редоследом ће унутрашња петља итерирати кроз инструкције. Међутим, брзина проналажења решења ипак зависи од редоследа итерирања. Показује се да се до решења долази у мање итерација спољашње петље уколико унутрашња петља итерира од последње инструкције ка првој, него од прве ка последњој. **Питање:** Шта се дешава ако унутрашња петља итерира од прве ка последњој инструкцији? Урадити ручно неколико првих рачунања за пример са слике 1.

За пример са слике 1 дати алгоритам са петљом организованом од последњег чвора (6) ка првом чвору (1) доводи до решења једначина у три итерације. Резултати за сваку итерацију су дати у следећој табели:

	succ	use	def	I		II		III	
				out	in	out	in	out	in
6		c			c		c		c
5	2,6	a		c	ac	ac	ac	ac	ac
4	5	b	a	ac	bc	ac	bc	ac	bc
3	4	bc	c	bc	bc	bc	bc	bc	bc
2	3	a	b	bc	ac	bc	ac	bc	ac
1	2		a	ac	c	ac	c	ac	c

Табела 1 Приказ итерација алгоритма животног века променљивих a, b и c из примера

Реализација у библиотеци *libLivenessAnalysis.lib*

Приликом реализације задатака је потребно користити постојеће структуре и функције. Једна инструкција описана је структуром `Instruction`. Она садржи тип инструкције (`type`), листе претходних и наредних инструкција `pred` и `succ`, листе променљивих које се користе и дефинишу: `use` и `def`, као и `in` и `out` листе, које представљају излаз из фазе анализе животног века променљивих. Више инструкција описује се као листа `Instructions` која се у функцији `makeExample()` попуњава примером кода.

[1] Листе in' и out' су копије садржаја листа in и out на почетку сваке итерације (унутрашња for петља).

Промењиве су описане у структури `Variable` и садрже назив (`name`) и позицију (`pos`) као једина поља од интереса за ову вежбу. Више промењивих описано је помоћу листе `Variables`.

Типови и функције од значаја из библиотеке *libLivenessAnalysis.lib*:

```
typedef std::list<Variable*> Variables;
      // листа променљивих
bool variableExists(Variable* variable, Variables variables);
      // провера постојања промењиве „variable“ у листи „variables“
typedef std::list<Instruction*> Instructions;
      // листа инструкција
```

Све операције могу се извести постојећим методама листе (List) из библиотеке стандардних шаблона (STL – Standard Template Library): *insert* (додавање једног елемента у листу или додавање дела друге листе), *sort* (сортирање листе), *unique* (брисање елемената који нису јединствени из **сортиране** листе), и *push_back* (додавање елемента на крај листе). Плитка копија листе се може једноставно направити коришћењем оператора доделе (=). Поређење једнакости две листе уграђених типова (у овом случају показивача) може се урадити коришћењем уобичајених оператора за поређење (==, !=).

За пример се узима листа инструкција са слике 1. Она садржи 3 промењиве A, B и C и 6 инструцкија (чворова). Попуњавање листе инструцкија по датом примеру је реализовано у функцији `makeExample()`.

ЗАДАТАК

1. Реализовати **део** алгоритма за одређивање животног века променљивих. Потребно је реализовати **само унутрашњу петљу** из алгоритма, ону која у обрнутом редоследу итерира кроз инструкције. Резултат упоредити са колоном I у Табели 1. Код писати у телу функције *livenessAnalysis*, чије заглавље је дато у датотеци *livenessAnalysis.h*, а празна дефиниција у датотеци *livenessAnalysis.cpp.*