Garbage collector in C

Project IOOPM

2016

Björnen

Johan Windahl, Sahand Shamal, Hampus Falk,
Simon Lövgren, Anto Juko and Karl Johansson

January 13, 2016

Contents

1 Extra Achievments

Other than Y68, we also want to be graded on these achievements:

- Johan Windahl:
  - Y64 Scrum
  - Y65 Code standard
  - Y66 Code review
  - Y67 Group reflections - Y67 Individual reflections
  - Y69 Regression testing
- Simon Lövgren:
  - Y64 Scrum
  - Y65 Code standard
  - Y66 Code review
  - Y67 Group reflections - Y67 Individual reflections
  - Y69 Regression testing
- Anto Juko:
  - Y64 Scrum
  - Y65 Code standard
  - Y66 Code review
  - Y67 Group reflections - Y67 Individual reflections
  - Y69 Regression testing
- Karl Johansson:
  - Y64 Scrum
  - Y65 Code standard
  - Y66 Code review
  - Y67 Group reflections - Y67 Individual reflections
  - Y69 Regression testing
- Sahand Shamal:
  - Y64 Scrum
  - Y65 Code standard
  - Y66 Code review
  - Y67 Group reflections - Y67 Individual reflections
  - Y69 Regression testing
- Hampus Falk:
  - Y64 Scrum
  - Y65 Code standard
  - Y66 Code review
  - Y67 Group reflections - Y67 Individual reflections
  - Y69 Regression testing

2 Group reflection (2.2)

Before the break we met only a couple times. Everyone had to read up on the subject and understand what had to be done, thus meetings wouldn't do much. However after the break we met every day and continued our work on the project. Our communication was excellent as we could ask each other questions directly. If we had more time we'd continue doing this as our productivity skyrocketed during these meetings. Our supervisor was quick to answer questions we may have had as well as create time for us to discuss our progress and answer questions.

Coordinating the project took little time, as we almost immediately managed to divide it into three distinct parts. These parts made it possible to work in pairs without having to rely on code from other pairs. This made things easy to coordinate as the borders were clear and everyone became an expert in their area. Everyone had something to do at all times and if one assignment was smaller then they could take care of other essentials.

Decisions to be made became easy due to the clear split between work areas. The groups knew what had to be done in order to reach our goal. This made the planning easy, once everyone is done, the process of putting it together would commence. However we may have underestimated the time it would take.

A problem with our way of splitting up was that we did not really know exactly what the others were doing. We believe this was mainly due to time constraints as another rotation or two would've put us in the other groups forcing us to read up on what was going on, thus understanding all the parts that would create the final product.

2.1 Group reflection on pair programming (Y67)

We immediately gravitated towards working together on two different computers, thus we did not follow the pair programming template in its truest form. While scenarios existed where working in front of one computer was better than divided we usually split the work and compensated the potential lack in quality by cross reviewing the code and ask for help from each other when needed. Working together was better when there was a more complex problem at hand. Otherwise during the more trivial functions there were no problem to split. The same thing applies to writing tests as we found it to be a waste of time doing it together. By doing this we could get the best of two worlds, high code quality with a fast working tempo.

The best part about pair programming was how well you learned when working together. Through discussion and different perspectives you could truly find the best solution to problems, as well as knowledge of how a certain problem got solved, which could be used in the future.

Swapping keyboard writers were great for learning. If the more experienced person is the one always writing the other one might be left out of the creative process as many decisions is made during writing. By changing writer the high

tempo that might have been created is slowed down and the more experienced partner will have to explain the process behind the decisions being made.

Our favored partner combination was someone with experience together with a less experienced. By doing this you can ensure that the code will have a good quality as well as that both partners are learning. By doing this the level of thought is maximized as they tend to think on two different levels. Where the more experienced programmer may complicate matters the one with less experience can take the complexity down a notch which will result in a more readable code.

2.2 Ongoing usage of code review (Y66)

Tried to use it a lot, did not prioritize it though. We used forks with pulls requests, and we did code reviews within the pairs aswell.
What we think code-review would improve:

- You do not push mediocre or half-done code.
- Write more understandable code.
- Avoid shortcuts.
- Code quality overall would improve if you knew someone would review it.

We liked this form of code-reviewing a lot more than the previous one in the beginning of the course.
Things we would do differently if we redid the project:

- More code reviewing
- Allocate more time to review code
- Be more courageous, tell group members if their code are not good enough.

2.3 Using Scrum with Trello on appropriate parts (Y64)

Our process: First we did some research to know how to split up the whole project into large parts. Our result: stack trace, heap, allocate object and integration + GC algorithm. After that each pair did more research on each large part, started to divide everything down to small parts that could be handled. We liked Trello and used it a lot. It gave us a good overview of the process.

- Another sprint would've been perfect for us, also another week. Our first sprint was two weeks long and we only switched partners once. The last few days of the project we worked together in school and didn't have any "real pairs" in a strict sense.
- Our planning was over-optimistic, though we did not know how hard the task ahead was.

- We had a good granularity level and broke everything down to small tasks in Trello.
- We did both over- and underestimate different tasks. For example, We thought the stack part would be larger and the GC algorithm part to be smaller.

Things we would do differently if we redid the project:

- More short standing meetings: What did our pair do today? Did we accomplish it? What are we doing tomorrow? So the rest of the people know what's going on outside their pair.
- Start integrating the different modules earlier
- Use more todo-lists.
- Plan more.

2.4 Using a code-standard (Y65)

We chose Apache with doxy-comments. `http://httpd.apache.org/dev/styleguide.html` We assigned 1 person to be responsible for the the code structure and formatting in the beginning. If anyone was uncertain about anything, they just asked him, and he had the last word when it came to which format to use. We liked using that method, cleared alot of things before we started with the implementation.
We used a linked list module (list.h + list.c) from Johans Assignment 2, it can be deviating from our standard.

2.5 Using regression testning (Y69)

Surprisingly good work with our tests. We developed our tests in parallel with the development of the modules. We assigned 1 passionate person to be responsible for all the tests, in the beginning. `https://github.com/IOOPM-UU/bjornen/tree/master/tests`

3 What's missing

- More complete tests for our GC.
- Internal tests for the object-module is not complete.
- Potential seg.fault if you send pointers that are not (stack->heap) to the object functions.
- 'Double' not tested in object-function object_to_bit_vector.
- Magic constants in many of the object-functions.
- More extensive testing with the GC threshold higher than 0.5.

- Our GC algorithm, allocates memory via malloc, outside our heap. Rewrite the algorithm to not use external lists for both performance and compliance.
- Refactor our gc_depth function, it's too complicated at this moment.
- Gcov results.
- Check our GC with Valgrind, and fix eventual leaks.
- Performance tests
  - For large programs that does fit in the memory. Check our GC's allocating performance vs malloc.
  - For large programs that doesn't fit in the memory. Check our GC's performance vs malloc.
  - Modify a Inlupp2 and check our GC's performance vs malloc.
  - Tests for the list-module.

4 Planning a new sprint

We would rotate pairs once more and divide the responsibility of these tasks among the pairs, the tasks would've been done in this order:

1. Refactor our gc_depth algorithm, it's too complicated at this moment.
2. Modify a Inlupp2 and check our GC's performance vs malloc.
3. Write the other performance test programs, fix eventual bugs. Do the tests.
4. For better performance and instead of writing tests for the list-module, we would rewrite the gc_depth algorithm to not use external lists at all.
5. Make more complete tests for our GC, fix eventual bugs.
6. Get Gcov results.
7. Check our GC with Valgrind, and fix our leaks.
8. Fixing the magic constants in many of the object-functions.
9. Fixing a 'Double' test in object-function object_to_bit_vector.

5 Individual-reflections

5.1 Using Pair programming (Y67 and 2.1.1)

5.1.1 Johan Windahl

The pair programming worked well! This whole project was a lot more advanced than the previous assignments. Therefore it was a lot smoother to solve some of the hard issues when you got another person to discuss solutions/problems

with. With simpler tasks that need no assistance it can be somehow boring. We used pair-programming but both people had computers in front of them, and if the task was more advanced all the focus was on one screen.

I think pair programming is good if there is some difference in experience between the users, and if you make sure to switch "writer" regularly. Not that effective if the difference is to high though.

The code quality in my opinion improves with pair programming when both persons has to "accept" the code.

The time we had were a bit tight, we used the 2 first week as one sprint and only switched partners once. If we had another week to the deadline we had probably switched partners again.
We could have been more strict about the pair programming and worked more in pairs.

5.1.2 Karl Johansson

I like pair programming when the task is a bit more complicated. Then I think it helps if you are two who can reason and discuss the solution while writing the code. But if it is an easier task then I think it is unnecessary if only one writes code. In that case I think it is better if you discuss what should be done and then do it separately.

To some extent I think one might produce better code if one of the programmers has an overview of the code and might catch things that the one writing the code misses.

I think it is a good technique when one of the programmers are inexperienced and the other is more experienced, so that the inexperienced one can learn from the one with more experience. But if it is a good technique for programming pairs with similar experience I think it depends on the task.

The advise I would give is, to discuss the task and plan on what you should do/write before starting and to have patience and listen to each other. It is also very important to be able to give and receive feedback.

5.1.3 Hampus Falk

I found the pair programming used within our group to work well. I believe we created some form of hybrid version of the pair programming template given to us. Instead of having two people in front of one computer we had one computer each and sat next to each other. By doing this we could get a lot more done as we had twice the manpower to create the code needed. By sitting next to each other we also maintained the possibility to ask and help one another when a problem might arise as well as cross review the code written.

On paper one might consider that the code created would not maintain the same type of quality as one created by the two of us, however I believe that the speed that the code could be written far outweigh the possible quality increase that could have been achieved. The code worked more than well in almost all cases.
As long as the partners respect each other and are open to feedback the pair programming could be very fruitful for both parties.

5.1.4 Anto Juko

The pros of pair-programming are that if there is a difficult problem to solve then you can take help from each other. It may also guarantee that the code will be written correct in the chosen coding standard and may also prevent bad naming of functions etc.. This due to the other person who's acting as a "co-pilot" and inspect the code continuously. But the cons is that it's not very time efficient when only writing the code on one computer. It's more efficient to sit together but with a computer individually. Then one person can for example make the functions and the other can make the test. If the both programmers have high or average programming skills then they are more capable of working individually and may only collaborate on same task if it gets to difficult. They will save a lot of time in that way when working on a project. But if they have uneven skills can lead to that the person who has less programming knowledge is not understanding what the other person has made. So then pair-programming can be a efficient way to learn programming when working on same computer, but not time efficient when making a large project. I can advice does who never tried pair-programming before to try it. If they find someone who has patient and understanding they can learn a lot by writing and getting corrected directly and also by watching the more skilled programmer.

5.1.5 Sahand Shamal

If the task is difficult pair programming can result in better code. Similarly, for difficult tasks the efficiency may increased when using pair programming, and decrease with easy ones.

Regarding individual skill-levels within pairs, this should balanced within reason. E.g. the difference can't be so high that one person doesn't understand what the other one is doing. If one person is a bit sharper, or does some things better, it can be educational for the other person (he or she could then learn things).

For people new to pair programming, I'd just advise to, from time to time, switch coder/reviewer, and also check that both of you understand what's happening.

I like pair programming as a complement to working individually. You'd then use pair programming when stuck or with difficult tasks. Spontaneously I think it's counterproductive to do this for easy tasks.

5.1.6 Simon Lövgren

I believe pair programming too cumbersome and time consuming to be an effective way of working. Most often it results in one part explaining code, aspects of programming or train of thought, which takes more time than simply writing the code. In many cases productivity is increased by distributing the tasks at hand, writing code individually. However, this is not always the case. In the case of an inexperienced programmer, programming alone may take longer than with an experienced partner. So, in aspect of teaching/learning to code, pair programming is a great technique as an inexperienced programmer is able to learn- and get tips & tricks from more experienced programmers.

What I have found to be a good technique is a middle-ground where both parties are able to work individually on tasks but are there for each other when

a hard problem arises, as well as for inspecting each others code.

My advice would be to respect the other party, his/her level of programming skill (whether higher, lower or equal) as well as bringing a smidge of patience.

5.2 Self-reflections (2.1.2)

5.2.1 Johan Windahl

My strengths: flexible, used to work with a lot of different kinds of people. I could have taken almost any role in the group. I took the leader role in this project. I am used to being a leader or being the boss managing other bosses in the working life. My weaknesses: Not as experienced/knowledgeable as a programmer as some of the other group members. I easily take the chilled-out-do-as-little-as-possible kind of role. I work really well in any team. I really enjoy working in a group, especially with new people. I also really liked the idea that you couldn't create your own groups. I do not think working alone is as profitable. I think this project was a appropriate ending of this course, It was fun and i really had to use a lot of the knowledge gained from the course. The assignments complexity overwhelmed me, and i underestimated the time needed for the assignment grossly. I learned a lot during this project!

5.2.2 Karl Johansson

I think I work well in a team and that other people think I am easy to work with. I like working in a team more than working alone, I believe I learn more if I am able to discuss the problem and possible solutions with someone rather than searching for it myself. If I am working in a team I believe I feel more confident to be able to solve a more complicated task than by myself.

For me to be able to take a prominent role I need to feel confident on my knowledge and in the beginning of this project that wasn't the case. But I always want to do my best and in this project it might have been in a less prominent role.

I think I'm good at getting an overall picture and overview of the situation. This project has made me realize that careful planning is required to be able to complete a project of this size and to find enough time. A issue for me has been that I underestimate the time a task will take.

5.2.3 Hampus Falk

I enjoy being a part of a team and I like to take the leader role. However this only applies to situations where I am knowledgeable and believe I can help the team. In this case I did not feel experienced enough to take such a role. The difficulty of the project in its entirety overwhelmed me and I would have liked to be able to participate more than I could. This I blame my lack in experience.

I enjoy working in groups in most cases. In this case I loved is as the problem were of such a caliber that I would not have been able to make it on my own at all. In most cases I will prefer working alone as I haven't gotten skillful enough to immediately comprehend others code. In this case there was no problem as the pair programming removed this hinder.

As mentioned this project caught me off guard and I felt that the course had not prepared me for the difficulty it would present. I did not feel fully ready for the project, however I have learned a lot. I tend to learn the best by observing other peoples code and write code in a gradually harder fashion, this project surely applied to this.

5.2.4 Anto Juko

When working in a group project I only tend to take the leadership if my skills of that task is higher then average and I'm truly understand what the task is. In this project I chosen to not take a leader roll because I feel that I don't have the programming experience that a leader of this particular project should have.

I often listen to a discussions of a problem and try to solve the problem by myself. But I often tend to not say my thoughts and this is certainly not a good side of me. It can be because Im afraid of discussing a problem and specially when I'm not sure that I really understand the main task. So i really have to work on that, to say what I think and not be afraid of having wrong. Over all I find it very thanksgiving to work in a group When everyone is striving for the same goal.

During the course I have learnt very much by working with the pair-programming technique which also was used during this project in a slight different way. I have learned a lot in the project about how you really make a large program and how to connect all files between each other more then during the course. My emacs skills have also increased.

5.2.5 Sahand Shamal

In teams I can assume a leader position if nobody else does. However if someone else is doing that well I'll focus on something else. I try to do a good job and have somewhat of a serious role.

I'm somewhat good at solving problems and try to be thorough. However I can waste time on things that aren't very important (e.g. labels for things, tests). If I learned to prioritize better I could probably benefit by finishing tasks faster.

With the project I've mostly learned about working in a group. For example I better understand the importance of planning, and realistically so. I've also let go (a bit) of micro-managing during pair-programming or similar contexts.

Regarding technical things I've improved my C and Emacs skills a bit. Much of this learning has been through talking or chatting with other group members. Contrast this with learning during individual work, where I use the internet or e-books a lot more. I've also learned by looking at other people's code, work flow, documentation and "meta files" (makefiles, readme:s, etc.) and either shamelessly copying them (which I recommend to everyone), or incorporated an aspect of these things into my own work.

5.2.6 Simon Lövgren

My perception is that I work well within a team, as I am not afraid to take initiative or to deal/help with tasks outside my own scope. It seems I often find myself in some sort of leadership, as I want my thoughts and ideas to matter

within the team. In addition I dread getting stuck in a rut where I receive an instruction, process the instruction then wait for the next instruction.

First and foremost I get motivation from working in a team. Sure there are some frustrating moments every now and then, but seeing others work motivates me to be productive.

The project feels like it has been about applying what we've learned during the course. The rest of the course has had clear and strict instructions, where as the instructions for the project has been more of an interpretation.

The best learning method for me is "learn by doing", and so I've had a relatively even learning curve during the course.

5.3 Personal journals

5.3.1 Johan Windahl

https://docs.google.com/spreadsheets/d/1s8jsrth_etQlQKmWUESfhbjQQMAOj9yKBMGmkYRQmiE/edit?usp=sharing

5.3.2 Karl Johansson

https://docs.google.com/spreadsheets/d/1N2f92RaNU8-Rl0vrig2WpV1CX6uawacUBlWdg9FAm04/edit?usp=sharing

5.3.3 Hampus Falk

https://www.overleaf.com/3828945rfxvkn#/11011042/

5.3.4 Anto Juko

https://docs.google.com/document/d/1x0jpegwFa8P1SARRVRpocOrW3Eqm9iEhpua8xLDkDP4/edit?usp=sharing

5.3.5 Sahand Shamal

https://docs.google.com/spreadsheets/d/1gbxqdEhoMYereO9Z8xR_ImbUkPMATsGBeaiDL6rFCFY/edit?usp=sharing

5.3.6 Simon Lövgren

https://www.overleaf.com/read/ggnkrvhbbcyh

5.4 Time compilation

Person	Meetings	Planning	Design	Imp.	Test	Doc.	P.mortem
Johan:	14.37	20.78	8	24	17	5	27
Hampus:	6	16	2	28	16	3	16
Simon:	7	3.5	10	58.5	23	4.5	4.5
Karl:	10.5	14	14	33.5	25.5	4.5	5
Sahand:	7	2	2.5	39.3	35	3	4
Anto:	8.16	6	2	25	26.5	2	6