PIRANTI LUNAK UNTUK MENDESAIN PROGRAM DALAM BAHASA PEMROGRAMAN C BERDASARKAN HOARE LOGIC

Arnold Aribowo[1)], Pujianto Yugopuspito[2)], Julian Fetriandhy Altanijah[3)]
[1)]Jurusan Teknik Komputer, Fakultas Ilmu Komputer, Universitas Pelita Harapan
[2,3)]Jurusan Teknik Informatika, Fakultas Ilmu Komputer, Universitas Pelita Harapan
UPH Tower, Lippo Karawaci, Tangerang 15811, Indonesia
e-mail : arnold.aribowo@staff.uph.edu, pujianto.yugopuspito@staff.uph.edu

Abstrak
The purpose of Hoare Logic is to provide a set of logical rules in order to reason about the correctness of computer programs with the rigor of mathematical logic. Because of that, Hoare Logic becomes the axiomatic basis for computer programming with several rules to prove the correctness of program. Hence, we can apply the proven rules of Hoare Logic as the basis to design a program correctly according to Hoare Logic.
In this paper, Hoare Logic is applied in a software which is designed to help the user to design a program in C programming language correctly based on rules in Hoare Logic. When using this software, the user needs to know what program he will create and analyze an algorithm for it. After that, the user can use the software containing the rules of Hoare Logic and write the pseudo-code of C to design his program. At the end of this application, the user will obtain a source code of the program written in C programming language. This software is guaranteed to produce 100% correct output only if the users have the basic understanding of Hoare Logic as well as C program language before using this software.

***Keyword** : Hoare Logic,* Perancangan program

1. PENDAHULUAN

Hoare Logic (juga dikenal dengan sebutan *Floyd-Hoare logic*) adalah sebuah sistem formal yang awalnya diciptakan oleh C. A. R Hoare, dan kemudian dikembangkan lebih lanjut oleh Hoare dan pemikir lainnya. Pada dasarnya *Hoare Logic* adalah kumpulan aturan-aturan logika untuk membuktikan kebenaran dari sebuah program dengan logika matematika. Aturan-aturan dasar tersebut merupakan bagian-bagian dasar dari suatu program sebelum menjadi sebuah program yang kompleks [2].
Makalah ini tidak membahas proses pembuktian program menggunakan Hoare Logic, tetapi makalah ini membahas bagaimana proses desain program dapat dimulai dari bagian terkecil dan dibuat berdasarkan aturan yang sudah ada pada *Hoare Logic* untuk kemudian digabungkan menjadi satu kesatuan program yang benar dan sesuai dengan aturan-aturan yang terdapat di dalam *Hoare Logic*. Beberapa aturan yang digunakan pada makalah ini adalah assignment axiom schema, composition, conditional, while, consequence rule, dan hoare triple. Pada dasarnya, yang dimaksud *precondition* pada makalah ini adalah keadaan yang harus dipenuhi untuk terjadinya eksekusi program dan *postcondition* yang dimaksud adalah efek yang dihasilkan dari eksekusi program dengan *precondition* yang ada. *Input* data yang dimasukkan oleh pengguna baik untuk *precondition* dan program adalah pernyataan yang valid di dalam bahasa pemrograman C. Diasumsikan pengguna sudah membuat algoritma dari permasalahan yang akan dikodekan sebagai sebuah program serta sudah menentukan *precondition,* dan perintah untuk setiap *rule* yang akan digunakan. *Output* dari penerapan *Hoare Logic* ini adalah program dalam bahasa C, dengan pembatasan hanya digunakan sebatas fungsi *void main()* saja, tidak dapat menggunakan *OOP*, dan fitur *Function* yang menggunakan fungsi *Return*.

2. TINJAUAN PUSTAKA

2.1 *Hoare Logic*

Hoare Logic, atau juga dikenal dengan nama *Floyd-Hoare Logic*, adalah sistem formal yang diciptakan oleh C. A. R. Hoare, dan dikembangkan lebih lanjut oleh Hoare bersama peneliti lainnnya. Dasar dari *Hoare Logic* adalah *Hoare Triplet* yang menjelaskan bagaimana sebuah eksekusi dari sepotong kode berpindah kondisi di dalam perhitungan. Bentuk dari *Hoare Triplet* adalah [1] :

$$\{P\}\ C\ \{Q\}$$

Di mana P dan Q adalah *Assertions* (formula dalam *predicate logic*) dan C adalah *Command*. P disebut dengan *precondition* dan Q dengan *postcondition*. Notasi ini dapat diinterpretasikan "*precondition* P akan menghasilkan *postcondition* Q ketika *command* C dieksekusi".

2.2 *Partial Correctness* dan *Total Correctness*

Dalam teorinya, *Hoare Logic* dapat diterapkan dalam dua keadaan, yaitu *Partial Correctness* dan *Total Correctness* [2]. Namun, pada makalah ini, *Hoare Logic* akan diterapkan untuk keadaan *Partial Correctness* karena sulitnya untuk memastikan jika suatu program akan berhenti dan menghasilkan hasil yang diinginkan.

2.3 Aturan-aturan *Hoare Logic*

Di dalam *Hoare Logic* terdapat beberapa aturan (*rules*) yang sudah ditetapkan. Aturan tersebut antara lain adalah [2]:

1. Axiom of Assignment

$$\{P[x/E]\}\ x:=E\ \{P\}$$

Notasi di atas menjelaskan bahwa di dalam P, keberadaan x dapat digantikan oleh ekspresi E. *Postcondition* dari aturan ini adalah P dengan keadaan x sudah digantikan dengan E.

2. Rule of Consequences

$$\frac{P' \rightarrow P,\ \{P\}\ S\ \{Q\},\ Q \rightarrow Q'}{\{P'\}\ S\ \{Q'\}}$$

Aturan ini menunjukkan bahwa *precondition* P dapat dikuatkan yang ditunjukan dengan notasi P' -> P dan *postcondition* Q dapat dilemahkan yang ditunjukan dengan notasi Q -> Q'. Notasi tersebut menunjukkan bahwa sisi sebelah kiri dari operator -> lebih kuat dibandingkan sisi sebelah kanannya.

3. Rule of Composition

$$\frac{\{P\}\ S\ \{Q\},\ \{Q\}\ T\ \{R\}}{\{P\}\ S;T\ \{R\}}$$

Aturan ini diterapkan pada program S dan T yang berjalan secara sekuensial. Pada notasi di atas, terlihat jika diketahui *precondition* P kemudian S dieksekusi menghasilkan *postcondition* Q, lalu kemudian Q dijadikan *precondition* dengan T dieksekusi sehingga menghasilkan *postcondition* R dapat dinotasikan menjadi lebih singkat dengan menjadikan P sebagai *precondition* lalu program S dieksekusi yang dilanjutkan dengan program T dieksekusi sehingga menghasilkan *postcondition* R.

4. Rule of Conditional

$$\frac{\{B \wedge P\}\ S\ \{Q\}\ ,\ \{\neg B \wedge P)\ T\ \{Q\}}{\{P\}\ S;T\ \{R\}}$$

Aturan ini diterapkan pada program yang dieksekusi jika suatu kondisi tertentu terpenuhi. Pada contoh di atas, B menyatakan kondisi dan P sebagai *precondition*. Jika kondisi B terpenuhi maka S akan dieksekusi dan jika kondisi B tidak terpenuhi maka T yang akan dieksekusi untuk menghasilkan *postcondition* Q.

5. Rule of Iteration

$$\frac{\{P \wedge B\}\ S\ \{P\}}{\{P\}\ \text{while B do S done}\ \{\neg B \wedge P\}}$$

Aturan ini diterapkan pada program yang mengandung iterasi atau pengulangan untuk kondisi tertentu. Pada notasi di atas, B menyatakan kondisi sebagai patokan untuk melakukan iterasi dan P sebagai *precondition.* Jika kondisi B terpenuhi maka S akan dieksekusi berkesinambungan hingga menghasilkan *postcondition* dimana kondisi B tidak terpenuhi lagi.

3. METODE PENELITIAN

Dalam aplikasi ini, pengguna memberikan masukan berupa *Pseudo-code* dalam bahasa C. *Pseudo-code* yang dimasukkan oleh pengguna merupakan hasil analisis dari algoritma aplikasi tertentu yang akan dibuat oleh pengguna dengan menggunakan aplikasi ini. *Pseudo-code* ini merupakan *Statement* yang akan dimasukkan ke dalam *Precondition* dan perintah di dalam *Hoare Logic* untuk kemudian dirangkai menjadi barisan program dalam bahasa C.

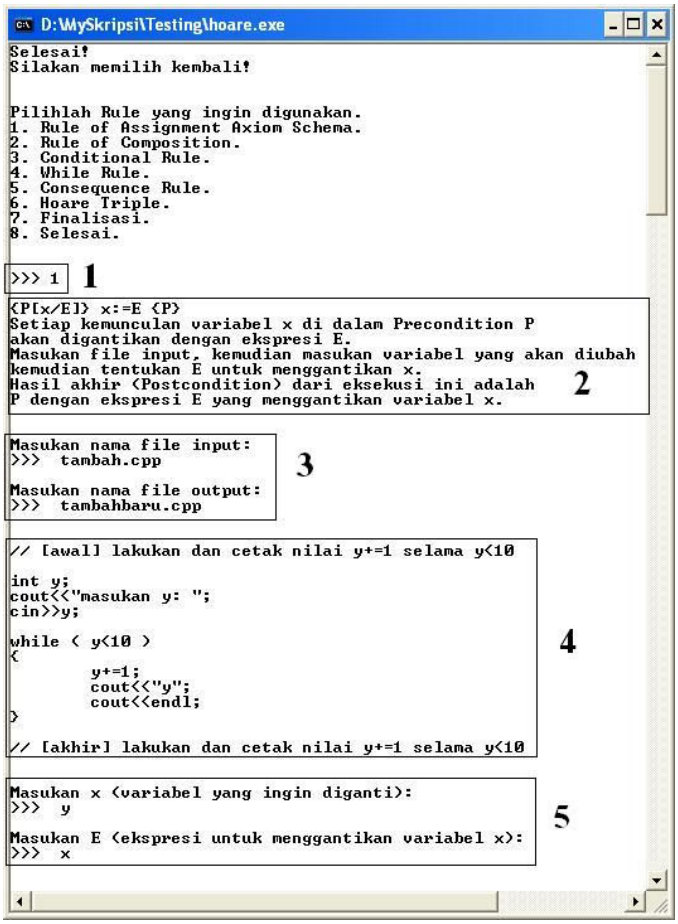
Sebelum memasukkan *Pseudo-code* yang sudah disiapkan tadi, pengguna harus menyiapkan algoritma dari program yang akan dibuat terlebih dahulu. Pilihan-pilihan yang ada untuk membuat program dibuat berdasarkan *Hoare Logic* yang sudah dijelaskan konsepnya pada bagian Tinjauan Pustaka, dengan modifikasi pada bagian *precondition*, dan *postcondition*.

4. HASIL DAN PEMBAHASAN

Pada bagian ini dipaparkan mengenai hasil piranti lunak yang telah dibuat. Menu pertama yang muncul ketika aplikasi ini dijalankan adalah pilihan dari aturan-aturan dalam *Hoare Logic* yang dapat digunakan (Gambar 1). Tanda ">>>" terlihat pada bagian bawah dari menu ini sebagai penanda bahwa pengguna dapat memasukkan pilihannya berupa angka antara 1-8 sesuai pilihan pengguna. Pada makalah ini hanya diperlihatkan 2 aturan saja.

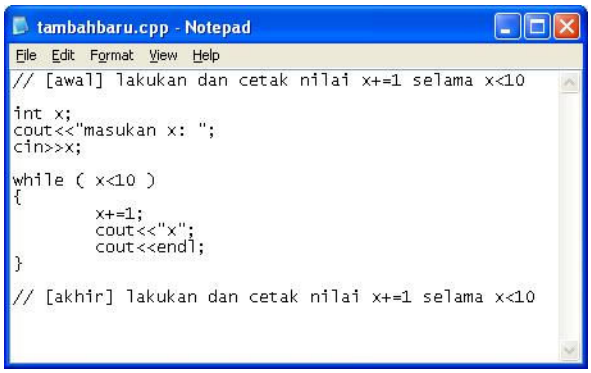
Gambar 1. Tampilan awal dari program

Pada pilihan "1", pengguna dapat menggunakan "*Rule of Assignment Axiom Schema*". Untuk dapat menggunakan aturan ini, pengguna memasukkan file *input* yang berisikan data yang akan mengalami perubahan, dan file *output* yang akan menampung perubahan tersebut. Setelah itu, akan muncul isi dari file *input* tersebut. Hal selanjutnya adalah memilih variabel yang akan diganti (dalam kasus ini adalah "y") dan variabel pengganti (dalam kasus ini adalah x). Setelah menekan *Enter,* maka menu akan terlihat seperti pada Gambar berikut :



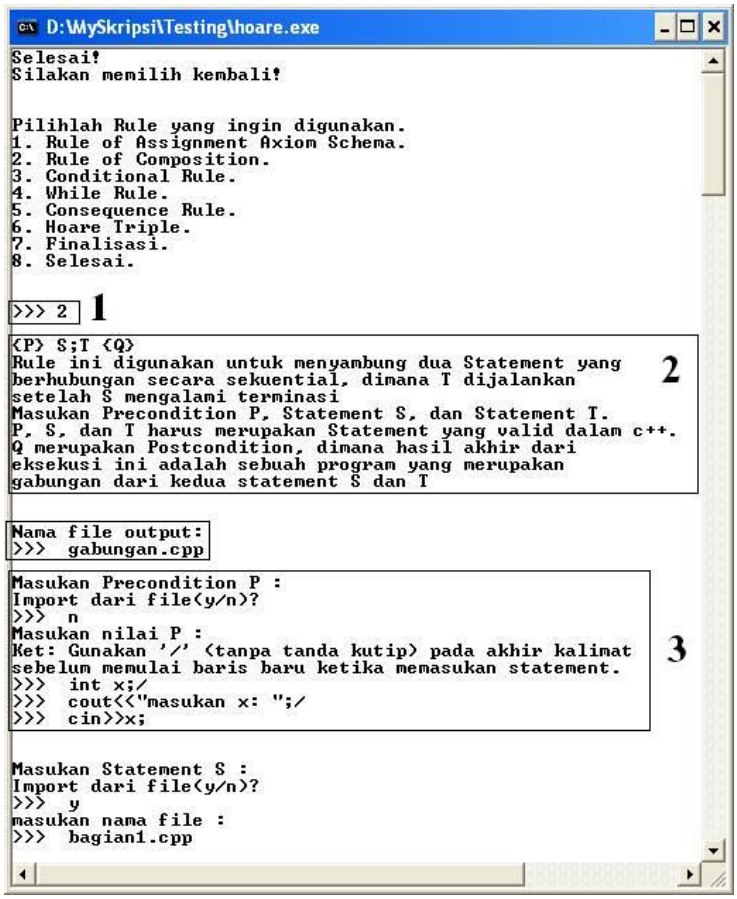
Gambar 2. Tampilan contoh ketika *Assignment Axiom Schema* digunakan

Setelah menekan tombol *Enter*, pengguna dapat memeriksa pada file *output* "tambahbaru.cpp" (Gambar 3). Pada *file* tersebut terlihat sudah terjadi perubahan jika dibandingkan dengan isi pada file *input* "tambah.cpp" seperti terlihat pada gambar 2, kotak nomor 4.



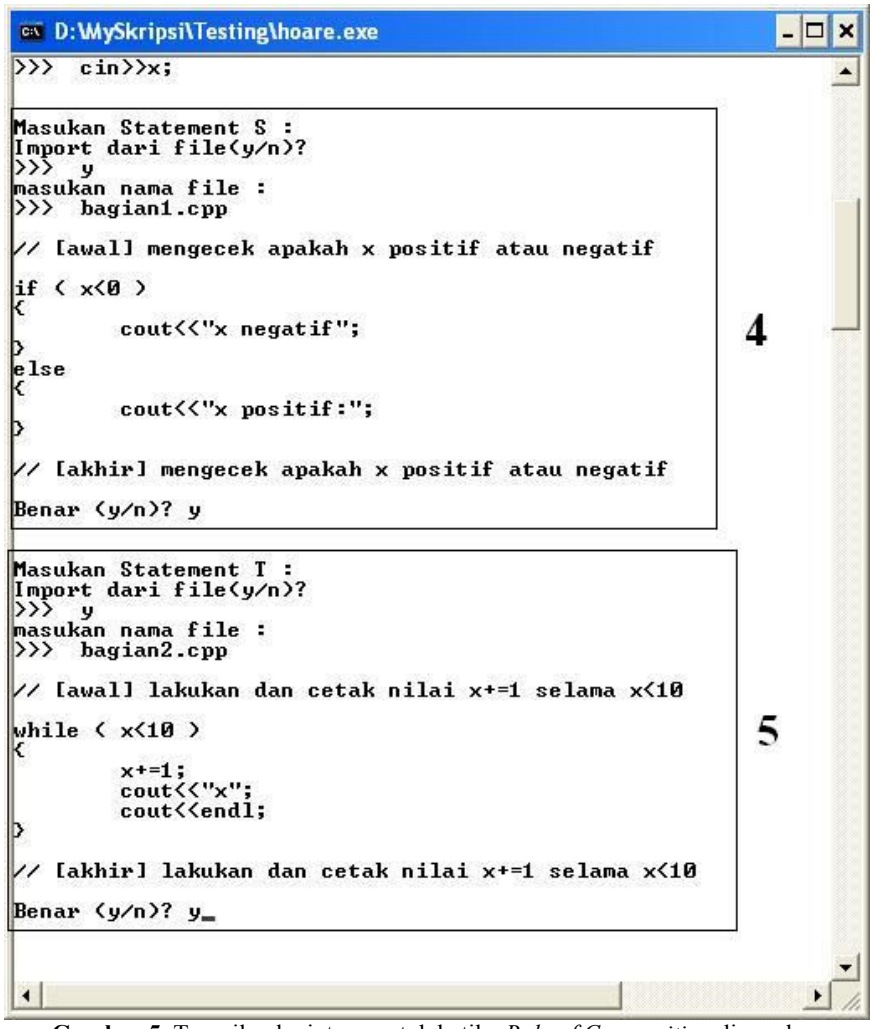
Gambar 3. Hasil perubahan yang sudah dilakukan

Pilihan selanjutnya adalan aturan "*Rule of Composition*" (Gambar 4 dan 5). Pada aturan ini, masukan file *output* dilakukan untuk menampung data hasil penggabungan dua file secara sekuensial. *Precondition* dimasukkan dahulu seperti pada kotak nomor 3. Jika ingin menggunakan *Precondition* dalam kondisi *True* saja, cukup dimasukkan spasi sekali kemudian klik *Enter*. Pada contoh ini, *Precondition* dimasukkan secara manual, karena pilihan jawaban dari pertanyaan "Import dari file (y/n)?" adalah "n". Untuk mempermudah proses *input*, pengguna dapat memulai baris baru untuk setiap kalimatnya dengan cara menambahkan simbol "/" pada bagian akhir dari *statement* sebelum menekan tombol *Enter*.



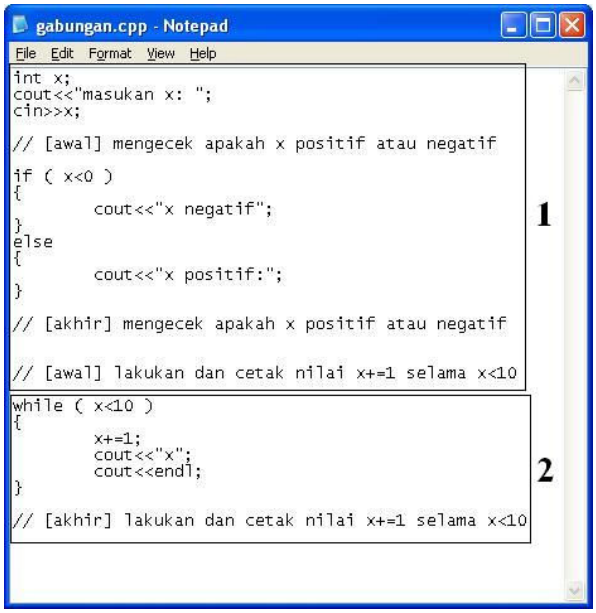
Gambar 4. Tampilan contoh ketika *Rule of Composition* digunakan

Tahapan selanjutnya adalah memasukkan file sekuential yang pertama yaitu *Statement* S. Pengguna dapat memilih antara memasukan secara manual atau melalui file yang sudah ada.



Gambar 5. Tampilan lanjutan contoh ketika *Rule of Composition* digunakan

Berikut adalah gambar file *output* yang berisikan *Precondition* P, *Statement* S dan T yang tadi dimasukkan.



Gambar 6. Hasil dari "*Rule of Composition*"

Pada piranti lunak yang dibuat terdapat fungsi tambahan yang terakhir untuk menambahkan *Header* dan *Void main()* pada program yang sudah dibuat dengan menggunakan *Hoare Logic* di atas.

5. KESIMPULAN

Piranti lunak yang dihasilkan ini dapat menerapkan beberapa aturan dasar pada *Hoare Logic* untuk mendesain sebuah program dalam bahasa C. *Hoare Logic* tidak diterapkan untuk proses verifikasi program. Proses pembuatan suatu program dimulai dari bagian yang paling dasar satu-persatu dengan menggunakan aturan *Hoare Logic* yang ada, lalu bagian dasar ini akan dapat digabungkan untuk menjadi satu kesatuan program seperti yang sudah direncanakan. Karena proses pembuatannya yang dimulai dari dasar tersebut, maka diharapkan kesalahan desain program dapat dihindari. Piranti lunak ini dapat dikembangkan lebih lanjut ke dalam bahasa pemrograman C++ dengan menambahkan konsep modular dan *Object-Oriented* yang merupakan keunggulan dari bahasa pemrograman C++.

6. DAFTAR PUSTAKA

C.A.R. Hoare, *An Axiomatic Basis for Computer Programming*, Journal of Communication of the ACM, 1969.
Hoare Logic-Wikipedia, http://en.wikipedia.org/wiki/Hoare_logic.