Escape Analysis dalam Bahasa Pemrograman Berbasis Garbage Collection

Zidane Firzatullah - 13521163
Program Studi Teknik Informatika
Sekolah Teknik Elektro dan Informatika
Institut Teknologi Bandung, Jalan Ganesha 10 Bandung
E-mail: 13521163@std.stei.itb.ac.id

***Abstract*—Makalah ini menjelaskan proses escape analysis pada bahasa pemrograman berbasis garbage collection khususnya bahasa pemrograman Java menggunakan pendekatan graf. Graf akan dibentuk melalui proses incremental pada baris fungsi menggunakan regular expression yang akan menentukan kategori statement pada suatu baris. Escape analysis menjadi penting karena bisa mengurangi penggunaan garbage collector untuk bahasa dengan garbage collector sehingga meningkatkan performa dari program. Escape analysis dilakukan dengan cara menganalisis apakah object tertentu akan keluar dari method pembuat object.**

Keywords—Regular Expression, Breadth-First Traversal, BFS, Escape Analysis, Garbage Collector, Heap Memory, Stack Memory, Java.

I. Pendahuluan

Bahasa pemrograman Java saat ini cukup mendominasi pasar karena kemudahan penggunaan akibat garbage collector dan banyaknya sistem existing yang diimplementasikan menggunakan Java. Dengan kemudahan memory management menggunakan garbage collector, terdapat beberapa masalah dalam performa karena memori object akan selalu dialokasikan di heap dan menyebabkan bertambahnya overhead pada saat proses garbage collection.

Dengan bertambahnya demand dan pengguna teknologi di Dunia, tentunya performa program menjadi sangat penting. Performa program bisa menjadi penyebab pengguna menggunakan service tertentu karena akan memberikan kenyamanan yang lebih saat penggunaan program. Performa yang buruk juga akan meningkatkan tagihan server jika tidak dilakukan optimasi karena response timenya yang meningkat dan juga memerlukan processing power yang lebih besar.

Pada makalah ini, akan diperlihatkan proses analisis pada bahasa pemrograman Java untuk mengurangi alokasi pada heap memory yang disebut sebagai escape analysis menggunakan pendekatan graf yang menggambarkan keterhubungan antar objek.

II. Landasan Teori

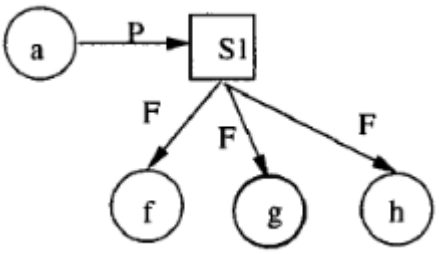
A. Algoritma Breadth-First Traversal

Breadth-First Traversal adalah algoritma traversal pada graph dengan cara memulai traversal pada node tertentu lalu melakukan eksplorasi tetangga langsung node tersebut dengan menggunakan struktur data queue (First In First Out) hingga queue tersebut kosong. Algoritma ini juga menggunakan suatu struktur data tambahan, set, untuk menandakan apakah suatu node sudah pernah dijumpai sebelumnya, set ini akan disebut visited set.

Adapun algoritma lengkap dari algoritma Breadth-First Traversal adalah sebagai berikut:

1. Buat objek struktur data queue dan set
2. Pilih node sebagai node awal proses traversal, lalu masukkan ke dalam queue
3. Jika queue kosong, maka akhiri proses traversal. Jika belum, ambil node teratas dari queue, lalu masukkan id unique node ke dalam set

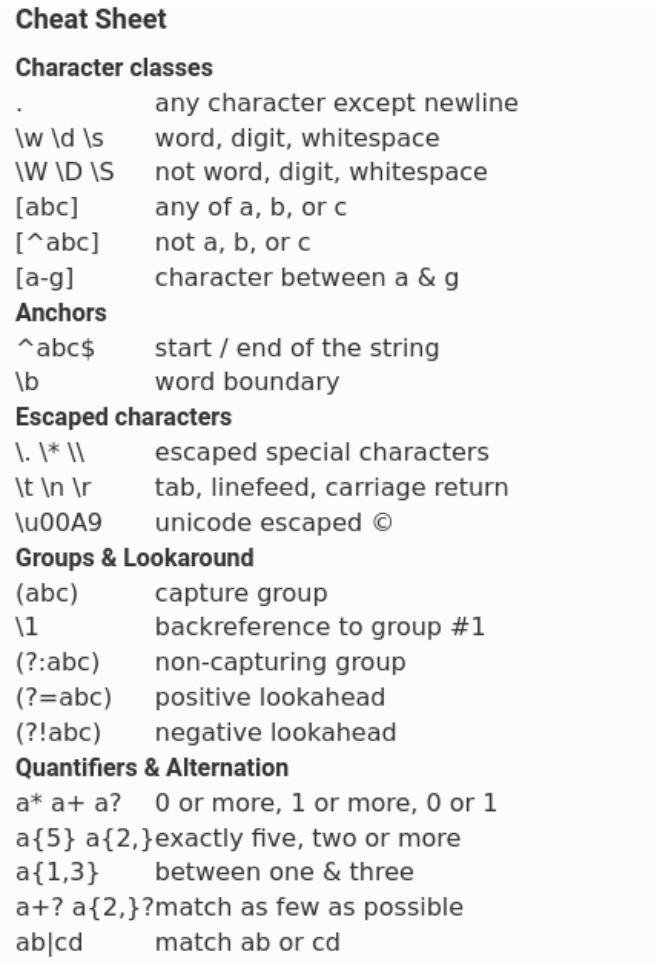
   Jika diperlukan, dalam langkah ini bisa dilakukan hal-hal lain sesuai kebutuhan
4. Iterasi setiap node tetangga lalu periksa satu-persatu apakah node tetangga tersebut sudah pernah dijumpai sebelumnya menggunakan visited set. Jika belum pernah dijumpai, masukkan node ke dalam queue
5. Lanjutkan ke langkah 3.



Gambar 1. Contoh Graf. Sumber: J.D. Choi, M. Gupta, M. Serrano, V.C Sreedhar, and S. Midkiff, “Escape Analysis for Java,”

B. Regular Expression

Regular Expression adalah sebuah pola dalam bentuk string yang bisa digunakan untuk proses pencocokan text oleh regular expression engine. Terdapat beberapa syntax regular expression yang biasa disebut sebagai regular expression flavor. Flavor regular expression yang paling umum adalah Perl-compatible regular expressions atau biasa disebut PCRE. PCRE diadopsi oleh banyak bahasa pemrograman seperti Perl, Python, JavaScript, PHP, Ruby, dan lain-lain.



Gambar 2. Referensi singkat regular expression. Sumber: regexpal.com

C. Bahasa Pemrograman Java

Java adalah bahasa pemrograman high-level C-like dengan orientasi sebagai bahasa pemrograman berbasis objek. Java pada awalnya di-desain oleh James Gosling di Sun Microsystems dan di-rilis pada bulan Mei 1995. Java akan di-compile terlebih dahulu ke dalam intermediary language yang disebut sebagai bytecode. Bytecode merupakan intermediary language yang akan digunakan oleh Java Virtual Machine (JVM) untuk dilakukan proses konversi dari bytecode menjadi native machine code saat runtime. Karena desain dari JVM, bytecode yang disimpan dalam file .class atau dikumpulkan dalam 1 archiving file (umumnya .jar) bisa dijalankan di seluruh mesin yang memiliki implementasi JVM. Kemampuan ini di-iklankan dengan kalimat *write once, run anywhere*.

D. Java Virtual Machine

Java Virtual Machine (JVM) adalah program yang melakukan loading, verifikasi, translasi bytecode, dan eksekusi program Java. Java Virtual Machine bisa disebut sebagai inti dari bahasa pemrograman Java karena seluruh aktivitas yang berkaitan dengan bytecode dilakukan disini.

Java memiliki slogan *write once, run anywhere*. implementasi Java Virtual Machine lah yang memungkinkan hal tersebut, karena dependency terhadap mesin dipindahkan yang awalnya di native code menjadi kepada implementasi Java Virtual Machine. Jadi, meskipun bytecode platform-independent, Java Virtual Machine tetaplah platform-dependent karena harus mencocokkan native code yang ditranslasi berdasarkan bytecode yang dimiliki. Slogan *write once, run anywhere* mungkin bisa lebih dirasakan impactnya saat komputer memiliki kemampuan dan operating-system dengan spesifikasi yang berbeda, pada bahasa C mungkin integer memiliki size yang berbeda pada tiap mesin sehingga code mungkin tidak bisa digunakan kembali. Tetapi, penggunaan JVM sekarang masih sangat relevan dan berguna. Beberapa alasannya adalah sebagai berikut:

1. Ketika developer melakukan kompilasi via operating system Windows dan MacOS, maka tidak perlu dilakukan kompilasi bytecode ulang jika ingin menjalankan jar file Java pada server Linux. Ini memberikan fleksibilitas kepada developer untuk memilih operating system dan mengurangi beban storage pada server karena tidak perlu menyimpan source code pada server untuk dilakukan kompilasi.
2. Java Virtual Machine melakukan optimasi yang sangat lengkap sehingga memungkinkan program yang dihasilkan akan menjadi jauh lebih efisien dibandingkan code C yang dihasilkan dengan development time yang sama. Ini juga memungkinkan developer untuk mengedepankan readability karena code yang kurang optimal tetapi mudah untuk dibaca bisa dioptimasi oleh Java Virtual Machine.
3. Java Virtual Machine memaksa pengguna untuk melakukan memory management otomatis via garbage collector. Mayoritas memory yang dialokasikan saat runtime by default akan disimpan di heap memory dan ketika beberapa parameter Java Virtual Machine menandakan bahwa object tertentu sudah bisa dihapus dari memory, garbage collector akan melakukan reclaiming atas memory tersebut. Tetapi desain ini tentunya memberikan overhead kepada performa karena terdapat beberapa program yang berjalan secara paralel untuk menjalankan memory management yang otomatis. Overhead performance yang disebabkan oleh heap allocation bisa dikurangi dengan cara melakukan escape analysis pada code program seperti yang akan kita lakukan dalam paper ini.

E. Teknik Memory Management

1. Manual Memory Management

   Teknik ini memungkinkan developer untuk mengatur penggunaan memory secara penuh tanpa intervensi program pembantu seperti virtual machine. Karena seluruh memory diatur secara manual oleh developer, maka teknik ini sangat membantu untuk meningkatkan performa program karena tidak ada

virtual machine yang berjalan untuk melakukan management memory.

Tetapi tanpa adanya virtual machine, tentunya program yang dihasilkan sangat berisiko untuk malah membebankan performa program yang disebabkan oleh salah penggunaan memory. Salah satu masalah yang banyak disebabkan oleh teknik ini adalah memory leak. Memory leak adalah masalah yang terjadi ketika memory yang dialokasikan tidak dikembalikan kepada operating system sehingga program menggunakan jumlah memory yang tidak wajar.

Bahasa populer yang menggunakan teknik ini adalah C dan C++. Keywords yang umum digunakan adalah malloc (memory allocation) dan free (membebaskan memory)

2. Garbage Collection

Garbage collection memungkinkan developer untuk tidak memikirkan memory management karena setiap memory yang dialokasikan akan dibebaskan secara otomatis oleh garbage collector. Teknik ini memudahkan developer karena mengurangi kompleksitas program dan juga mempercepat development time.

Garbage collection umumnya diimplementasikan menggunakan menggunakan pendekatan Mark & Sweep, Reference Counting, dan Copying.

3. Resource Acquisition is Initialization (RAII)

RAII adalah teknik yang melakukan binding sebuah resource (termasuk memory) kepada sebuah object pemilik (owner object). Pada umumnya program akan melakukan alokasi pada stack sebanyak mungkin, tetapi ketika sebuah resource memiliki ukuran yang terlalu besar maka resource tersebut akan disimpan dalam heap memory. object pemilik memiliki tanggung jawab untuk membebaskan memory yang digunakan oleh resource yang dimilikinya melalui destruktor.

Ketika object pemilik tidak diakses lagi via stack memory, maka destruktor objek tersebut akan dipanggil dan membebaskan resource yang dimiliki. Teknik ini menjadikan memory management terikat erat dengan lifecycle object dan deterministik, artinya aksi pembebasan memory dilakukan pada titik yang sama setiap saat dan developer bisa mengatur kapan destruksi object harus dilakukan.

4. Automatic Reference Counting(ARC)

Teknik ini menyimpan berapa banyak reference object yang menunjuk ke suatu objek. Reference counter akan otomatis bertambah ketika terdapat reference baru dan berkurang jika ada reference yang berubah. Ketika reference count bernilai 0, maka object akan dibebaskan secara otomatis.

Teknik ini mirip dengan teknik reference counting pada garbage collector, tetapi teknik ini melakukan pengubahan source code di titik objek akan memiliki reference count 0 sehingga tidak memerlukan adanya virtual machine yang mengendalikan pembebasan memory ketika reference count menjadi 0.

F. Garbage Collector

Garbage collector adalah program yang bertugas untuk melakukan garbage collection dan memastikan proses garbage collection dilakukan secara menyeluruh. Pada umumnya garbage collector diimplementasikan menggunakan pendekatan (algoritma) sebagai berikut:

1. Mark & Sweep

Mark & sweep membagi operasi garbage collection menjadi 2 tahap, yaitu tahap mark dan tahap sweep. Mark & sweep memberikan bit tambahan, bisa disebut sebagai mark bit, untuk melakukan garbage collection. Ketika awalnya object dibuat, mark bit akan di-set menjadi 0.Ketika proses garbage collection dilakukan seluruh object yang bisa di reference mark bitnya akan diubah menjadi 1, proses ini bisa dilakukan melalui algoritma traversal pada umumnya seperti Breadth-First Traversal dan Depth-First Traversal untuk menjumpai seluruh object. Pada tahapan sweep, seluruh object yang tidak bisa di reference, mark bitnya 0, memorinya akan dibebaskan dan bisa digunakan untuk object lain.

Kekurangan algoritma mark & sweep adalah proses ini cukup memakan waktu karena harus melakukan pausing saat runtime jika tidak dijalankan secara paralel. Algoritma ini juga semakin lama akan meningkatkan fragmentasi memori sehingga mungkin harus dilakukan beberapa proses tambahan agar fragmentasi virtual memory berkurang

2. Reference Counting

Reference counting adalah teknik dimana setiap objek heap yang dibuat akan memiliki reference counter, yaitu penghitungan seberapa banyak sebuah objek di reference oleh variable lain. Reference counter akan bertambah jika ada variabel reference baru yang menunjuk ke objek tersebut dan berkurang jika ada objek yang mengubah arah referensi. Ketika reference counter bernilai 0, maka object tersebut secara otomatis akan dihancurkan dan memori objek akan dikembalikan kepada proses.

Kekurangan teknik ini adalah ketika terdapat beberapa objek yang saling menyimpan reference, maka memori tidak akan di reclaim. Teknik ini juga mengharuskan setiap objek memiliki reference counter yang tentunya akan memakan memori.

3. Copying

G. Escape Analysis

Escape analysis adalah sebuah proses analisa pada saat compile time yang dilakukan untuk mengurangi penggunaan heap memory dan menggunakan stack memory untuk objek yang tidak akan di reference diluar

sebuah scope. Escape analysis akan sangat membantu performance program karena garbage collector pada umumnya memakan performa yang cukup besar ketika melakukan proses garbage collection. Ketika garbage collector menggunakan skema single thread, maka garbage collector akan melakukan pausing terhadap program. Bahkan ketika garbage collection bersifat parallel (multi-threaded), proses ini tentunya akan tetap memakan waktu lebih untuk proses thread scheduling dan memory region tempat garbage collection akan dikunci sebagai mekanisme menjaga konsistensi memori dan masalah konkurensi.

Escape analysis pada umumnya berkonsentrasi pada masalah berikut:

1. Ketika sebuah object dibuat dalam method dan object tidak keluar (escape) method tersebut, maka object akan dialokasikan ke stack frame milik method tersebut (stack memory). Proses ini dilakukan karena pada alokasi heap mungkin terdapat kasus dimana proses alokasi memori harus melibatkan synchronization untuk menghindari kemungkinan masalah konkurensi. Pada masalah ini, overhead garbage collector akan berkurang karena seluruh memori pada stack akan otomatis dikembalikan ketika method melakukan return.
2. Ketika object tidak keluar dari sebuah thread, maka proses sinkronisasi pada objek tersebut bisa dihilangkan dan mengurangi beban performance yang diakibatkan oleh penggunaan lock. Masalah ini juga bisa mengurangi waktu akses memori dengan penambahan aspek lokalitas memori karena memori bisa dialokasikan pada memori processor tempat thread tersebut dijadwalkan.

III. Implementasi

Pertama, definisikan regular expression sebagai berikut:

1. Mencocokkan definisi function pada Java.

   Regex: `(private|public|protected) .+\\w+\\(.*\\)\\s*\\{(.|[\\s])*?}`

2. Mencocokkan dot operator pada Java

   Regex: `\\w+\\.\\w+\\s*=\\s*.+;`

3. Mencocokkan *new* statement pada Java

   Regex: `.+\\s+\\w+\\s*=\\s*new .+\\(\\);`

4. Mencocokkan copy operator pada Java

   Regex: `\\w+\\s+\\w+\\s*=\\s*.+;`

5. Mencocokkan expression string literal pada Java

   Regex: `\".*\"`

6. Mencocokkan return statement pada Java

   Regex: `return\s+.+\s*;`

7. Mencocokkan add to list statement pada Java

   Regex: `\\w+\\.add\\(.+\\);`

Kedua, definisikan status escaping object yang terdiri atas:

1. return_escape: object escape melalui return statement pada function
2. arg_escape: object escape melalui argumen yang diberikan pada function
3. no_escape: object yang tidak escape dan bisa di-alokasikan di stack memory

Algoritma dari proses escape analysis adalah sebagai berikut:

1. Buat sebuah map dengan key berbentuk string dan value berisi holder object dari object asli (Node dalam graph).
2. Jika fungsi yang dicek memiliki argument, maka tambahkan object argument dan seluruh fieldnya ke dalam map yang sudah dibuat.
3. Jika ada baris fungsi yang belum diproses, lanjutkan proses. Jika sudah semua, maka lanjut ke langkah 9.
4. Jika baris sekarang match dengan regular expression untuk dot operator lakukan: 1. Jika right-hand side cocok dengan regular expression string literal, buat objek string baru 2. Jika tidak, maka cari objek dari right-hand side di node map dan buat node dummy jika objek tidak tersedia. Objek right-hand side kemudian ditambahkan kedalam node map jika belum dan tambahkan objek right-hand side sebagai neighbor node dari objek left-hand side. Lanjutkan ke langkah 3.
5. Jika baris sekarang match dengan regular expression new statement maka buat object dengan tipe sesuai right-hand side dan masukkan ke dalam node map. Lanjutkan ke langkah 3.
6. Jika baris sekarang match dengan regular expression return, maka dapatkan object yang akan direturn dari function tersebut lalu ubah escape status node holder menjadi return_escape. lanjut ke langkah 9.
7. Jika baris sekarang match dengan regular expression penambahan object pada list, maka tambahkan object yang akan ditambahkan kepada list menjadi neighbor dari node holder object list. Lanjutkan ke langkah 3.
8. Jika sampai ke langkah ini dan baris sekarang tidak match dengan regular expression copy, lanjutkan ke langkah 3.

   Jika match dengan regular expression copy, maka buat node baru dan tambahkan object right-hand side sebagai neighbor object baru.

9. Jika seluruh baris fungsi sudah diproses, maka lakukan breadth-first traversal terhadap seluruh node didalam node map. Seluruh node yang bisa diakses melalui return_escape akan memiliki escape status return escape dan berlaku juga untuk arg_escape.
10. Seluruh node yang memiliki status selain no_escape maka tidak diperbolehkan untuk dialokasikan di stack memory.

IV. Pengujian

Definisikan kasus uji dalam bentuk function Java. Program harus memberikan list dari object-object yang memiliki status no_escape (stack allocatable).

Berikut kasus uji yang dibuat:

```
private void method_satu(DummyObjectOne arg_1) {
    String s_1 = "sdasda";
    arg_1.name = s_1;
}
```

Gambar 3. Kasus Uji 1

```
private List<DummyObjectOne> method_dua() {
    List<DummyObjectOne> list = new ArrayList<>();
    DummyObjectOne f = new DummyObjectOne( name: "1-1");
    DummyObjectOne s = new DummyObjectOne( name: "1-2");
    DummyObjectOne thi = new DummyObjectOne( name: "1-3");
    DummyObjectOne fourth = new DummyObjectOne( name: "1-4");
    DummyObjectOne fifth = new DummyObjectOne( name: "1-5");

    list.add(f);
    list.add(s);
    list.add(thi);
    list.add(fourth);
    list.add(fifth);

    return list;
}
```

Gambar 4. Kasus Uji 2.

```
private void method_tiga() {
    DummyObjectTwo d_1 = new DummyObjectTwo( name: "nama-tiga");
}
```

Gambar 5. Kasus Uji 3.

Adapun penjelasan dari kasus uji adalah berikut:

1. Kasus Uji 1: object s_1 tidak di return oleh method_satu, tetapi object tersebut disimpan di dalam object arg_1 dalam field name, maka object ini tidak boleh dialokasikan di stack memory. Maka, arg_1: arg_escape, s_1 arg_escape
2. Kasus Uji 2: object f, s, thi, fourth, dan fifth tidak di return oleh method_dua, tetapi object List bernama list memiliki element object-object tersebut, maka object f, s, thi, fourth, dan fifth tidak boleh dialokasikan di stack memory. Maka, f return_escape, s return_escape, thi return_escape, fourth return_escape, fifth return_escape.
3. Kasus Uji 3: object d_1 tidak dimasukkan ke dalam object lain dan tidak di return oleh method_tiga, maka d_1 boleh dialokasikan di stack memory. Maka d_1 no_escape.

Hasil Pengujian:

1. Kasus Uji 1

```
Status: ARG_ESCAPE key: arg_1
Status: ARG_ESCAPE key: s_1
Status: ARG_ESCAPE key: arg_1.name
```

Gambar 6. Hasil Kasus Uji 1.

2. Kasus Uji 2

```
Status: RETURN_ESCAPE key: s
Status: RETURN_ESCAPE key: thi
Status: RETURN_ESCAPE key: f
Status: RETURN_ESCAPE key: fifth
Status: RETURN_ESCAPE key: fourth
Status: RETURN_ESCAPE key: list
```

Gambar 7. Hasil Kasus Uji 2.

3. Kasus Uji 3

```
Status: NO_ESCAPE key: d_1
```

Gambar 8. Hasil Kasus Uji 3.

V. Kesimpulan dan Saran

Escape analysis adalah proses analisis saat compile-time untuk memastikan objek-objek yang bersifat lokal pada suatu function dialokasikan ke stack memory untuk mencapai performa yang lebih tinggi dan efisien. Escape analysis bisa mengurangi overhead yang disebabkan oleh garbage collector yang umumnya memakan waktu dan performa. Escape analysis bisa dilakukan dengan melakukan parsing terhadap source code via regular expression lalu membentuk graph yang menggambarkan keterhubungan antar objek dan terakhir menentukan reachability setiap objek via Breadth-First Traversal untuk menentukan apakah objek akan diakses oleh code luar method.

Kedepannya, makalah yang membahas masalah ini bisa menggunakan automata yang lebih baik seperti simple Context-Free Grammar untuk memastikan bahwa syntax benar saat proses parsing. Makalah ini baru membahas sebagian method escape analysis dan belum membahas escape analysis untuk threading. Untuk makalah yang lebih lengkap dan sempurna, pelengkapan escape analysis pada method dan riset tentang threading escape analysis akan sangat membantu.

Video Link at Youtube *(Heading 5)*

-

Ucapan Terima Kasih

Ucapan terima kasih yang sebesar-besarnya saya ucapkan kepada dosen pengampu mata kuliah IF2211 - Strategi Algoritma karena sudah membimbing saya selaku peserta dan mahasiswa Teknik Informatika ITB. Terima kasih secara khusus saya sampaikan kepada bapak Rinaldi Munir selaku dosen pengajar kelas K-1 mata kuliah Strategi Algoritma karena sudah menyampaikan materi perkuliahan dengan jelas dan efektif.

References

[1] J.D. Choi, M. Gupta, M. Serrano, V.C Sreedhar, and S. Midkiff, "Escape Analysis for Java," ACM SIGPLAN Notices vol. 34, pp. 1-19.

[2] C. Agarwal. "Mark-and-Sweep: Garbage Collection Algorithm". geeksforgeeks.org. https://www.geeksforgeeks.org/mark-and-sweep-garbage-collection-algorithm/ (accessed May 22, 2023).

[3] "Object lifetime and resource management (RAII)". learn.microsoft.com. https://learn.microsoft.com/en-us/cpp/cpp/object-lifetime-and-resource-management-modern-cpp (accessed May 22, 2023).

[4] D. K. Sasidharan. "Demystifying memory management in modern programming languages". dev.to. https://dev.to/deepu105/demystifying-memory-management-in-modern-programming-languages-ddd/ (accessed May 22, 2023).

PERNYATAAN

Dengan ini saya menyatakan bahwa makalah yang saya tulis ini adalah tulisan saya sendiri, bukan saduran, atau terjemahan dari makalah orang lain, dan bukan plagiasi.

Bandung, 22 Mei 2023

Zidane Firzatullah
13521163