MODUL PEMROGRAMAN BERORIENTASI OBJEK

(CCC210)

MODUL 01

PENDAHULUAN

DISUSUN OLEH

INDRIANI NOOR HAPSARI, ST, MT

UNIVERSITAS ESA UNGGUL

2020

MODUL 1 - PENDAHULUAN

A. Kemampuan Akhir Yang Diharapkan

Setelah mempelajari modul ini, diharapkan:

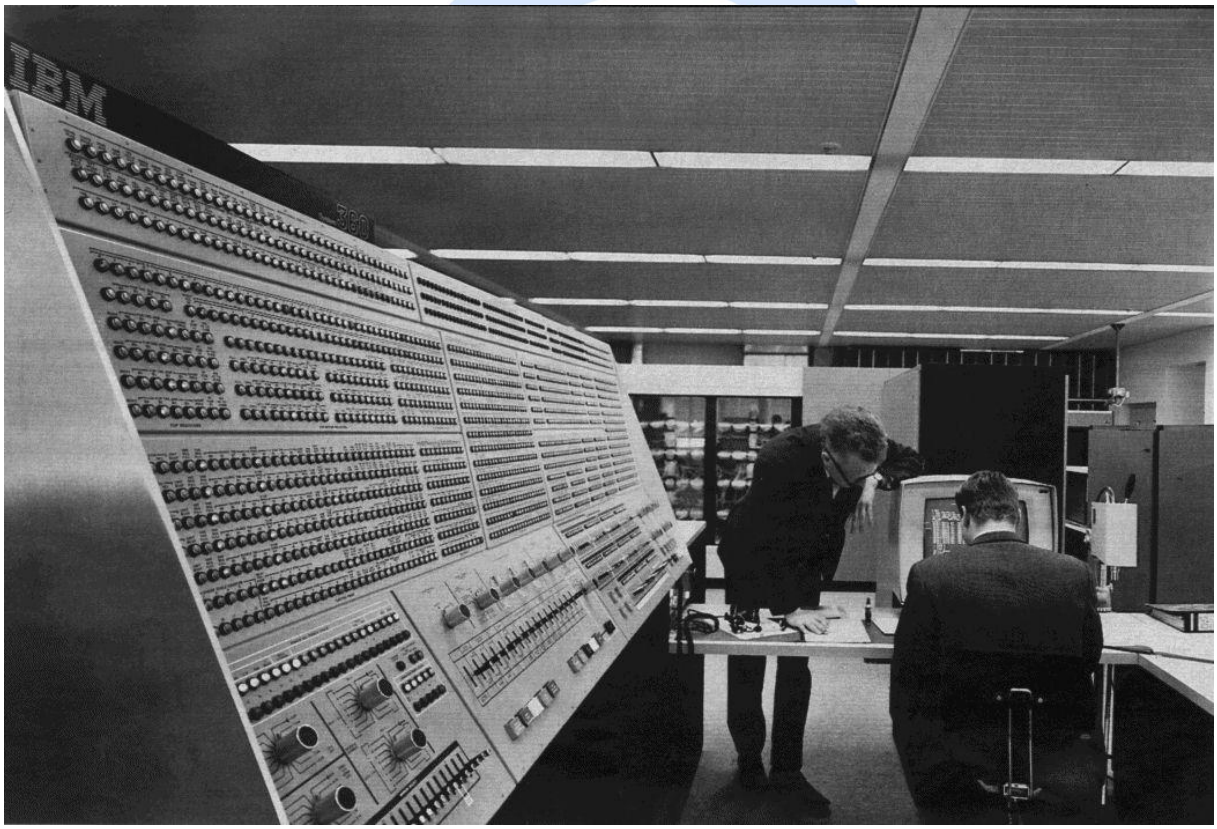
1. Mahasiswa dapat membedakan paradigma pemrograman terstruktur dengan pemrograman berorientasi objek
2. Mahasiswa memahami manfaat pemrograman berorientasi objek
3. Mahasiswa memahami karakteristik utama dalam pemrograman berorientasi objek
4. Mahasiswa dapat menerapkan konsep Object Oriented (OO) sederhana dalam bahasa pemrograman

B. Outline Topik

C. Uraian

1. Latar Belakang

Sepanjang sejarah pemrograman, meningkatnya kompleksitas program telah mendorong kebutuhan akan cara yang lebih baik untuk mengelola kompleksitas tersebut. Pemrograman berorientasi objek adalah jawaban atas kebutuhan itu. Pendekatan pemrograman telah berubah secara dramatis sejak penemuan komputer. Untuk lebih memahami korelasi antara peningkatan kompleksitas program dan perkembangan bahasa komputer, berikut ini dijelaskan perkembangan pemrograman.



Gambar 1 System/360 Model 91 front panel

Ketika komputer pertama kali ditemukan, pemrograman dilakukan dengan menggunakan "*front panel*" komputer untuk diubah ke instruksi mesin biner (Gambar 1). Selama program hanya terdiri dari beberapa ratus instruksi, pendekatan ini berhasil. Saat program berkembang, bahasa *assembly* ditemukan sehingga pemrogram dapat menangani program yang lebih besar dan semakin kompleks dengan menggunakan representasi simbolis dari instruksi mesin (bahasa *assembly*

disebut sebagai bahasa tingkat rendah). Ketika program terus berkembang, bahasa tingkat tinggi dikembangkan untuk memberi *programmer* lebih banyak alat yang dapat digunakan untuk menangani kompleksitas.

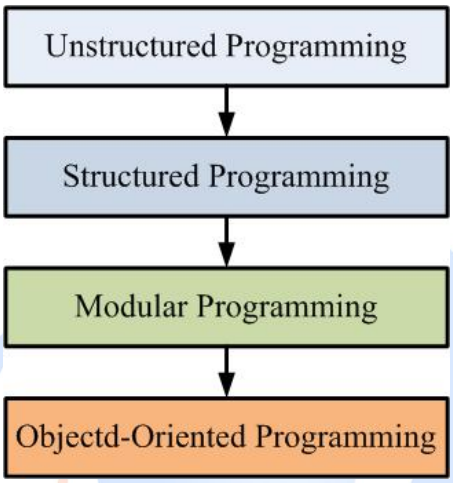
Bahasa komputer pertama yang banyak digunakan adalah FORTRAN. Meskipun FORTRAN merupakan langkah pertama yang sangat mengesankan, bahasa ini bukanlah bahasa yang jelas dan mudah dipahami. Tahun 1960-an lahirlah pemrograman terstruktur, yang merupakan metode pemrograman yang didorong oleh bahasa seperti C. Dengan bahasa terstruktur, untuk pertama kalinya, menulis program yang cukup kompleks dimungkinkan dengan cukup mudah. Namun demikian, bahkan dengan metode pemrograman terstruktur, apabila proyek mencapai ukuran tertentu, kompleksitasnya melebihi apa yang dapat dikelola oleh *programmer*. Pada akhir 1970-an, banyak proyek mendekati atau pada titik ini.

Menanggapi masalah ini, cara baru untuk program mulai muncul: pemrograman berorientasi objek (OOP). Dengan OOP, seorang programmer dapat menangani program yang lebih besar dan lebih kompleks. Masalahnya adalah C tidak mendukung pemrograman berorientasi objek. Keinginan akan versi C yang berorientasi objek pada akhirnya mengarah pada pembuatan C++.

Dalam analisis akhir, meskipun C adalah salah satu bahasa pemrograman profesional yang paling disukai dan banyak digunakan di dunia, ada saatnya kemampuannya untuk menangani kompleksitas terlampaui. Begitu sebuah program mencapai ukuran tertentu, ia menjadi sangat kompleks sehingga sulit untuk dipahami. Tujuan C++ adalah untuk membantu *programmer* memahami dan mengelola program yang lebih besar dan lebih kompleks. Selain bahasa C++, telah banyak dikembangkan bahasa pemrograman lain yang juga menerapkan konsep OO, diantaranya yaitu Java, C#, Python, dan sebagainya.

2. Evolusi Bahasa Pemrograman

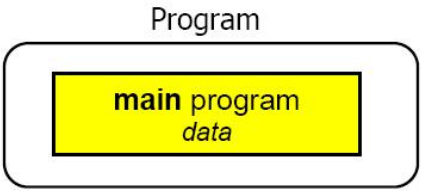
Evolusi bahasa pemrograman dimulai dari pemrograman tidak terstruktur, pemrograman terstruktur, pemrograman modular, dan pemrograman berorientasi objek. Evolusi bahasa pemrograman ditunjukkan pada Gambar 2.



Gambar 2 Evolusi Bahasa Pemrograman

A. Pemrograman Tidak Terstruktur

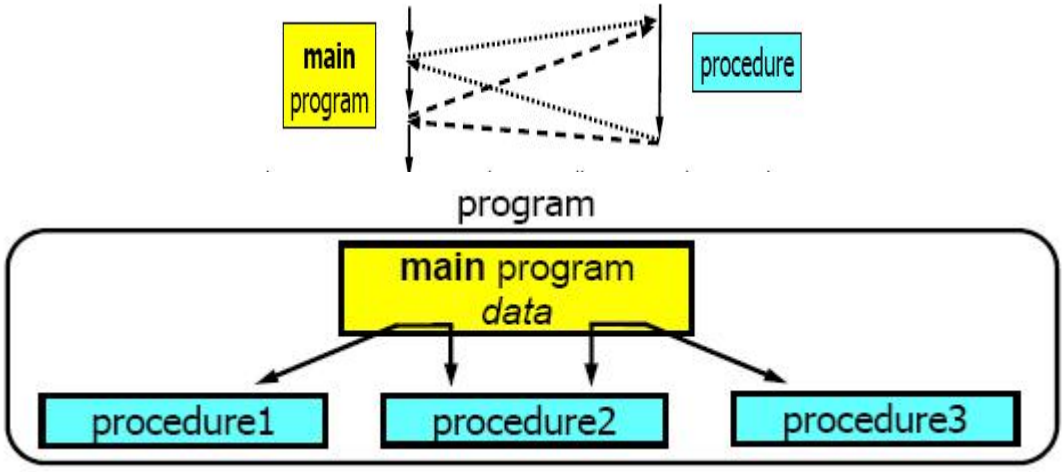
Pada awal komputer ditemukan, instruksi program dituliskan secara tidak terstruktur yang berisi urut-urutan instruksi untuk dieksekusi oleh mesin. Pada pemrograman tidak terstruktur, program utama secara langsung mengoperasikan data yang dideklarasikan sebagai variabel global. Serangkaian *statement* kode program yang sama harus di-*copy* jika diperlukan di tempat yang berbeda. Contoh bahasa pemrograman tidak terstruktur yaitu bahasa *assembly* dan versi awal dari bahasa Basic, Fortran, dan COBOL. .Abstraksi pemrograman tidak terstruktur ditunjukkan pada Gambar 3.



Gambar 3 Abstraksi Pemrograman Tidak Terstruktur

B. Pemrograman Terstruktur / Prosedural

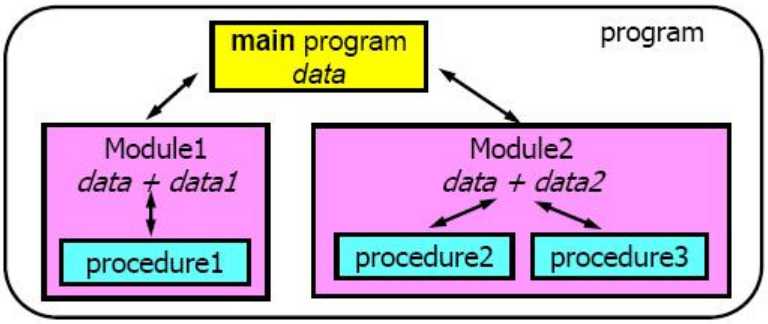
Untuk mengatasi kompleksitas pemrograman pada saat itu, dikembangkan pemrograman terstruktur (dikenal juga dengan istilah pemrograman prosedural) yang mengkombinasikan antara urut-urutan *statement* kode program ke dalam sebuah prosedur dengan pemanggilan dan pengembalian prosedur. Program utama mengkoordinasikan pemanggilan prosedur dan mem-*passing* data ke prosedur sebagai parameter. Contoh bahasa pemrograman terstruktur diantaranya adalah bahasa Basic, C, Fortran, dan Pascal. .Ilustrasi pemrograman terstruktur ditunjukkan pada Gambar 4.



Gambar 4 Ilustrasi Pemrograman Terstruktur

C. Pemrograman Modular

Pada pemrograman modular, prosedur yang memiliki fungsionalitas serupa dikelompokkan dalam sebuah modul terpisah. Ide ini mengimplementasikan prinsip *code-reuse*, dimana modul cukup dituliskan sekali untuk dapat digunakan di beberapa program yang berbeda. Program utama mengkoordinasikan pemanggilan prosedur yang ada di dalam modul-modul yang terpisah. Bahasa pemrograman C telah dikembangkan dengan menerapkan pemrograman modular. Ilustrasi pemrograman modular ditunjukkan pada Gambar 5.



Gambar 5 Ilustrasi Pemrograman Modular

D. Pemrograman Berorientasi Objek.

Semakin berkembangnya kebutuhan pemrograman, pemrograman modular masih dirasa terlalu kompleks untuk membuat program berskala besar. Walaupun program telah ditulis dengan struktur yang baik, program berukuran besar menjadi terlalu kompleks. Hal ini disebabkan karena pemrograman ini memungkinkan akses bebas terhadap data global. Selain itu, atribut dan perilaku program masih dipisahkan yang mana kurang merepresentasikan pemodelan di dunia nyata. Oleh karena itu, dikembangkan pemrograman berorientasi objek yang membungkus atribut (data) dan perilaku (prosedur) ke dalam sebuah objek.

3. Pemrograman Terstruktur vs Pemrograman Berorientasi Objek

Pemrograman berorientasi objek mengambil ide-ide terbaik dari pemrograman terstruktur dan menggabungkannya dengan beberapa konsep baru. Pemrograman berorientasi objek menyediakan cara pengorganisasian program yang berbeda dan lebih baik. Dalam pengertian yang paling umum, sebuah program dapat diorganisasikan dalam salah satu dari dua cara berikut:

1. Program diorganisasikan di sekitar kodenya (apa yang terjadi)
2. Program disusun di sekitar datanya (siapa yang terpengaruh)

Dengan teknik pemrograman terstruktur, program biasanya diorganisasikan di sekitar kode. Pendekatan ini dapat dianggap sebagai "kode melakukan aksi terhadap data" Program berorientasi objek bekerja sebaliknya. Mereka diatur di seputar data, dengan prinsip kuncinya adalah "data yang mengontrol akses ke kode". Dalam bahasa berorientasi objek; didefinisikan data dan aksi yang diizinkan terhadap data tersebut. Jadi, sebuah tipe data mendefinisikan dengan tepat jenis operasi apa yang dapat diterapkan pada data tersebut. Perbedaan antara pemrograman terstruktur dan pemrograman berorientasi objek dijelaskan pada tabel berikut ini.

Structural (Procedural) Programming	***Object Oriented Programming***
Pemrograman menggunakan struktur	Pemrograman dibangun di atas

Structural (Procedural) Programming	*Object Oriented Programming*
kontrol yang didefinisikan dengan baik ● Kondisional, perulangan, berurutan, ekspresi, dan *assignment* ● Data (variabel, array, struktur) dipisahkan dari operasinya ● Pemrograman ini menyediakan abstraksi dari *hardware*	pemrograman terstruktur. Pemrograman didasarkan pada konsep objek ● Objek membungkus data dengan operasinya ● Memungkinkan penyembunyian informasi (*information hiding*), sehingga program dapat diorganisasikan dengan cara yang mudah dikelola.
Program biasanya diorganisasikan di sekitar kode. Pendekatan ini dapat dianggap sebagai “kode melakukan aksi terhadap data”.	Program diorganisasikan di sekitar data. Pendekatan ini dapat dianggap sebagai “data mengendalikan akses ke kode”. Kita mendefinisikan data dan operasi yang diijinkan terhadap data tersebut.

4. Konsep Dasar Pemrograman Berorientasi Objek

Konsep fundamental dari pemrograman berorientasi objek adalah enkapsulasi data dan prosedur (fungsi) bersama ke sebuah unit yang disebut objek. Sebuah objek terdiri atas:

1. **nama** - cara untuk mengacu pada objek di dalam sebuah program (misalkan, objek Persegi dapat dipanggil dengan P1)
2. **atribut** - member data yang ada di dalam sebuah objek (misalkan, Persegi mempunyai atribut panjang dan lebar dengan tipe integer)
3. **fungsi** - operasi-operasi yang dapat dilakukan terhadap data di dalam sebuah objek (misalkan, objek persegi P1 memiliki fungsi untuk menghitung luas persegi)
4. **interface** - mendefinisikan cara programmer untuk mengakses data/fungsi dari sebuah objek (akan dijelaskan kemudian).

OOP menawarkan sejumlah manfaat bagi perancang program maupun pengguna. Manfaat utamanya diantaranya adalah sebagai berikut:

1. Menekankan pada data di atas prosedur.
2. Program dibagi-bagi dalam beberapa objek.
3. Struktur data dirancang sedemikian hingga mengkarakteristikkan objek.
4. Fungsi yang beroperasi pada data dari sebuah objek diikat bersama dalam sebuah struktur data.
5. Data disembunyikan dan tidak dapat diakses oleh fungsi eksternal.
6. Objek dapat berkomunikasi satu dengan lainnya melalui fungsi.
7. Data dan fungsi baru dapat ditambahkan dengan mudah di tempat manapun yang diperlukan.
8. Mengikuti pendekatan bottom-up dalam merancang program.
9. Melalui inheritance, OOP dapat mengeliminasi kode redundan dan dapat meng-extend penggunaan kelas yang sudah ada.
10. Program dapat dibangun dari modul-modul yang dapat berkomunikasi satu dengan lain, ketimbang harus membuat program dari awal. Hal ini dapat meningkatkan produktivitas dengan menghemat waktu membuat program.
11. Prinsip-prinsip "*data hiding*" membantu programmer membangun program yang aman yang tidak dapat diakses oleh kode di luar program.

12. Dimungkinkan membuat banyak instansiasi objek terpisah yang tidak saling mengganggu satu dengan lainnya.
13. Mudah untuk membagi-bagi pekerjaan dalam sebuah proyek, ke dalam beberapa objek.
14. Sistem OO dapat dengan mudah diupgrade dari sistem kecil ke sistem besar.
15. Teknik “message-passing” untuk komunikasi antar objek membuat deskripsi antarmuka dengan sistem eksternal menjadi lebih sederhana.
16. Kompleksitas *software* dapat dikelola dengan mudah.

Di samping itu, OOP juga memiliki fitur-fitur penting sebagai berikut:

1. **Natural**: mengikuti cara berpikir manusia (manusia melihat dunianya sebagai kumpulan objek yang saling berinteraksi)
2. **Abstraction**: menjelaskan makna sebuah entitas secara cepat dan mudah.
3. **Encapsulation**: Membungkus atribut (data) dan perilaku (fungsi) bersama dalam sebuah kelas
4. **Information Hiding**: dapat menyembunyikan detil yang tidak diperlukan
5. **Modular**: objek adalah entitas yang independen satu dengan lainnya.

5. Karakteristik Umum Pemrograman Berorientasi Objek

Untuk mendukung prinsip-prinsip pemrograman berorientasi objek, semua bahasa OOP, memiliki tiga ciri umum yaitu enkapsulasi, polimorfisme, dan pewarisan.

1. Enkapsulasi

Enkapsulasi adalah mekanisme pemrograman yang membungkus bersama kode dan data yang dimanipulasinya, dan menjaga keduanya aman dari gangguan dan penyalahgunaan dari eksternal. Dalam pemrograman berorientasi objek, kode dan data dapat disatukan bersama seperti halnya sebuah black box dibuat. Di dalam box terdapat seluruh data dan kode yang diperlukan untuk memanipulasi data tersebut. Ketika kode dan data dihubungkan bersama dengan cara ini, sebuah objek diciptakan. Dengan kata lain, sebuah objek merupakan perangkat yang mendukung enkapsulasi.

2. Polimorfisme

Polimorfisme (dari bahasa Yunani yang berarti “banyak bentuk”) merupakan fitur OO yang memungkinkan sebuah antarmuka untuk mengakses kelas generik dari sebuah aksi-aksi. Contoh sederhana dari polimorfisme ditemukan dalam roda kemudi/setir sebuah mobil. Roda kemudi merupakan antarmuka dari mobil, yang bentuknya sama walaupun mekanisme kerja kemudinya dapat berbeda-beda. Dengan demikian, cara user mengoperasikan roda kemudi akan sama pada mobil yang memiliki kemudi manual, *power steering*, ataupun mekanisme kemudi lainnya. Memutar kemudi ke kiri akan mengarahkan mobil belok ke kiri walau mekanisme kemudinya berbeda. Manfaat dari antarmuka yang seragam ini adalah, apabila kita sudah tahu sekali cara mengemudikan mobil, kita bisa mengemudikan berbagai tipe mobil lainnya.

Prinsip-prinsip ini juga diterapkan dalam pemrograman. Sebagai contoh, sebuah *stack* (yaitu sebuah *list* dengan mekanisme *first-in, last-out*). Kita mungkin memerlukan sebuah program yang memerlukan tiga jenis *stack* berbeda. Satu stack digunakan untuk menyimpan nilai integer, satu stack untuk menyimpan nilai desimal, dan satu *stack* untuk menyimpan nilai karakter. Pada kasus ini, algoritma untuk mengimplementasikan *stack* sama, walaupun data yang disimpan memiliki tipe yang berbeda. Pada pemrograman non-OO, kita harus membuat tiga set operasi stack

yang berbeda, dengan masing-masing set menggunakan nama yang berbeda. Namun, dengan adanya polimorfisme, kita cukup membuat satu set operasi *stack* yang dapat bekerja dalam tiga situasi berbeda. Dengan cara ini, sekali kita mengetahui bagaimana cara menggunakan satu *stack*, kita dapat menggunakan ketiga *stack* lainnya.

3. Pewarisan

Pewarisan (*inheritance*) adalah proses yang mana sebuah objek dapat memperoleh properti yang dimiliki oleh objek lainnya. Hal ini penting karena dapat mendukung konsep klasifikasi hirarki. Sebagian besar pengetahuan dibuat dapat dikelola dengan klasifikasi hirarki (*top-down*). Sebagai contoh, apel merah merupakan bagian dari klasifikasi (kelas) apel, yang merupakan bagian dari kelas buah, yang juga dibawah kelas makanan. Dengan demikian, kelas makanan memiliki kualitas tertentu (dapat dimakan, bernutrisi, dan sebagainya) yang mana juga dimiliki oleh kelas turunannya yaitu kelas buah. Pada kelas buah, selain memiliki kualitas yang dimiliki oleh kelas makanan, kelas buah juga memiliki kualitas spesifik (berair, manis, dan sebagainya) yang membedakannya dengan makanan lainnya. Kelas apel mendefinisikan kualitas spesifik dari sebuah apel (yaitu tumbuh di pohon, *non-tropical*, dsb). Kelas apel merah pun akan mewarisi seluruh kualitas/properti kelas induknya, dan memiliki kualitas spesifik yang hanya dimiliki oleh apel merah.

Tanpa menggunakan hirarki, setiap objek akan mempunyai definisi seluruh karakteristik tersebut secara eksplisit. Dengan pewarisan, sebuah objek hanya perlu mendefinisikan kualitas uniknya sendiri, sedangkan kualitas umum diwariskan dari kelas induknya. Dengan demikian, mekanisme pewarisan memungkinkan sebuah objek menjadi instansiasi spesifik dari objek lain yang lebih generik.

6. Class

Class merupakan konsep fundamental pada bahasa berorientasi objek yang menyediakan sebuah blueprint untuk tipe baru (klasifikasi) dari sebuah objek. Kelas mendefinisikan data-data, fungsi, dan antarmuka objek yang akan diterima oleh kelas. Sebuah kelas juga mendefinisikan bagaimana objek-objek dari kelas tersebut berperilaku, dengan menyediakan kode yang mengimplementasikan fungsi-fungsi yang berkaitan dengan kelas. Seorang programmer dapat membuat satu atau lebih objek dari sebuah kelas. Hal ini mirip dengan seorang insinyur yang dapat membangun beberapa rumah dari satu set blueprint.

Sebagai contoh, didefinisikan sebuah kelas bernama **Rectangle** yang memiliki atribut **panjang** dan **lebar**. Kelas Rectangle membatasi akses data dari program eksternal dengan memberikan antarmuka (interface) "**private**" pada datanya. Program eksternal hanya dapat mengakses fungsi-fungsi yang diset "**publik**", misalkan, fungsi menghitung luas (**area**), dan mengeset atribut panjang dan lebarnya (**set_values**). Berdasarkan deskripsi tersebut, maka implementasi kelas Rectangle secara sederhana dalam bahasa pemrograman C++ ditunjukkan pada kode program berikut ini.

```
class Rectangle
{
      private:
      int panjang,lebar;

      public:
        Rectangle(int pLong, int pWidth);
        void set_values(int pLong, int pWidth);
        int area(void);
};
```

D. Latihan

1. Meningkatnya kompleksitas program telah mendorong kebutuhan akan cara yang lebih baik untuk mengelola kompleksitas tersebut.
   a) Benar
   b) Salah
   c) Tidak diketahui
2. Berikut ini adalah bahasa pemrograman yang juga menerapkan konsep OO, kecuali
   a) C++
   b) C
   c) Java
3. Pada paradigma pemrograman ini, program utama secara langsung mengoperasikan data yang dideklarasikan sebagai variabel global.
   a) Pemrograman Terstruktur
   b) Pemrograman Berorientasi Objek
   c) Jawaban tidak tersedia
4. Pada paradigma pemrograman ini, serangkaian *statement* kode program yang sama harus di-*copy* jika diperlukan di tempat yang berbeda.
   a) Pemrograman Terstruktur
   b) Pemrograman Berorientasi Objek
   c) Jawaban tidak tersedia
5. Pada paradigma pemrograman ini, data dan operasi terhadap data tersebut diikat bersama sehingga memungkinkan data tidak dapat dimanipulasi oleh fungsi eksternal.
   a) Pemrograman Terstruktur
   b) Pemrograman Berorientasi Objek
   c) Jawaban tidak tersedia
6. Mekanisme pemrograman yang membungkus bersama kode dan data yang dimanipulasinya, dan menjaga keduanya aman dari gangguan dan penyalahgunaan dari eksternal..
   a) Enkapsulasi
   b) Inheritance

c) Polymorphism

7. Salah satu prinsip OO yang memungkinkan sebuah antarmuka untuk mengakses kelas generik dari sebuah aksi-aksi.
   a) Enkapsulasi
   b) Inheritance
   c) Polymorphism
8. Salah satu prinsip OO yang mana sebuah objek dapat memperoleh properti yang dimiliki oleh objek lainnya.
   a) Enkapsulasi
   b) Inheritance
   c) Polymorphism

E. Daftar Referensi

Walter Savitch, *Problem Solving with C++*, Pearson International Edition, 2006.

http://www.mu.ac.in/myweb_test/MCA study material/

http://www.cs.fsu.edu/~xyuan/cop3330/

https://www.programiz.com/cpp-programming