MODUL PRAKTIKUM
EL2208 PRAKTIKUM PEMECAHAN MASALAH DENGAN C

Oleh

Dr. Reza Darmakusuma, S.T, M.T.

Dismas Widyanto

Elkhan Julian Brillianshah

Irfan Tito Kurniawan

PROGRAM STUDI TEKNIK ELEKTRO

SEKOLAH TEKNIK ELEKTRO DAN INFORMATIKA

INSTITUT TEKNOLOGI BANDUNG

2023

DAFTAR ISI

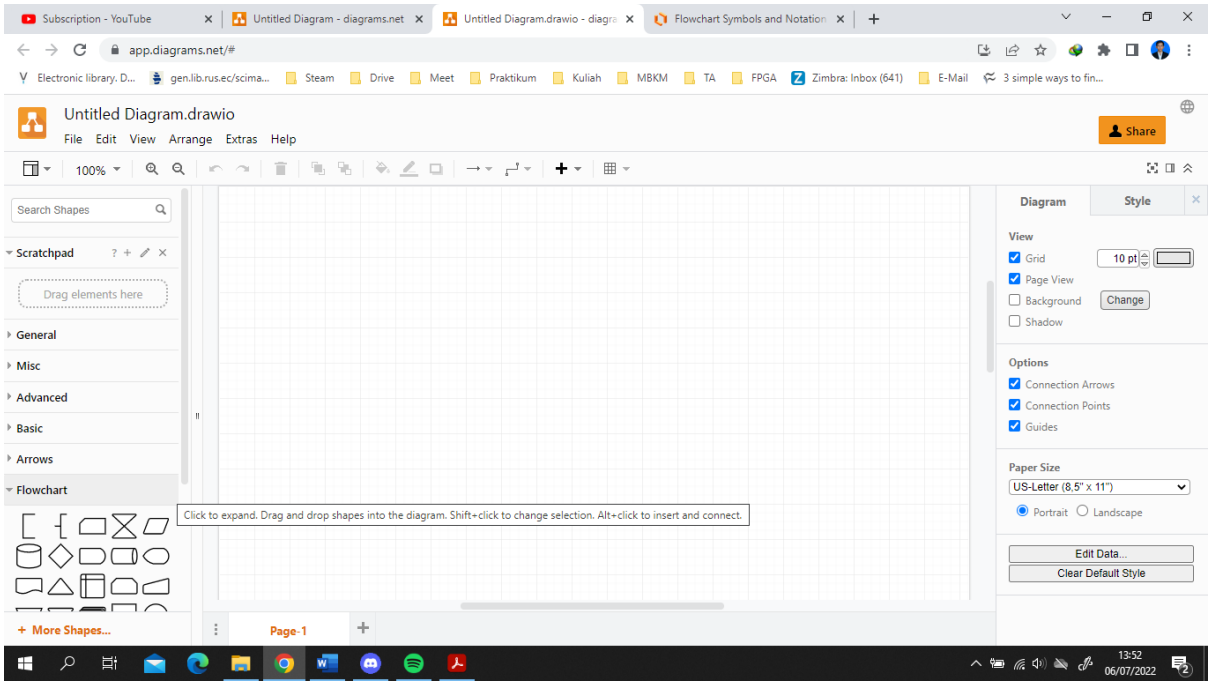
DAFTAR GAMBAR

DAFTAR TABEL

DAFTAR KODE

FLOWCHART

Secara umum pembuatan *flowchart* dibebaskan dengan catatan mengikuti kaidah pada umumnya, namun sangat disarankan untuk pembuatan *flowchart* menggunakan aplikasi khusus untuk membuat *flowchart* seperti draw.io (https://app.diagrams.net/) atau lucidchart (https://www.lucidchart.com/). Dalam dokumen ini akan digunakan aplikasi draw.io dalam pembuatan *flowchart*. Silakan akses aplikasi melalui tautan yang disediakan kemudian gunakan menu *flowchart* yang tersedia pada bagian kiri tampilan Anda seperti pada Gambar 1.



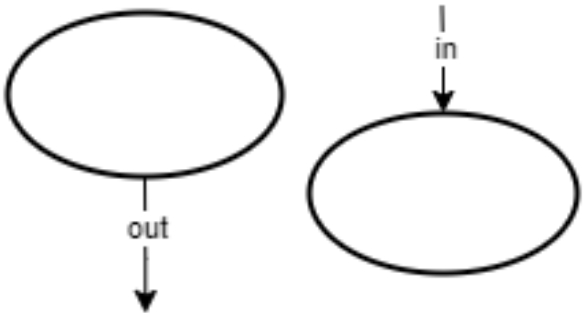
Gambar 1 Tampilan pada Draw.io

I. Kaidah *Flowchart*

Terdapat enam buah bentuk blok yang perlu Anda perhatikan dalam membuat *flowchart*. Masing-masing blok memiliki kegunaan dan kaidah yang berbeda-beda. Berikut merupakan penjelasan dari masing-masing blok.

I.I. Blok Start/End

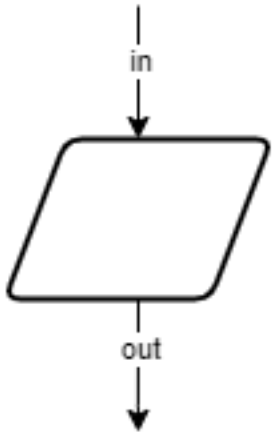
Blok ini digunakan untuk mengawali atau mengakhiri sebuah *flowchart*. Blok ini memiliki masing-masing satu *input* atau satu *output* saja. Bentuk yang digunakan adalah oval dengan satu *output* untuk blok end (kanan) dan satu *input* untuk blok start (kiri).



Gambar 2 Blok Start (kiri) dan Blok End (kanan)

I.II. Blok Input/Output

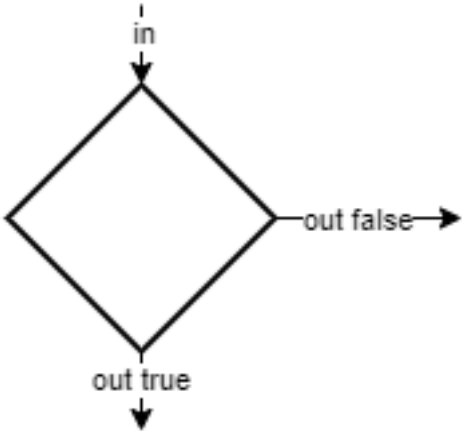
Blok input/output digunakan untuk mendefinisikan proses *input*/*output* yang dilakukan pada program. Blok ini berbentuk jajar genjang dengan satu buah *input* dan satu buah *output*.



Gambar 3 Blok Input/Output

I.III. Blok Decision

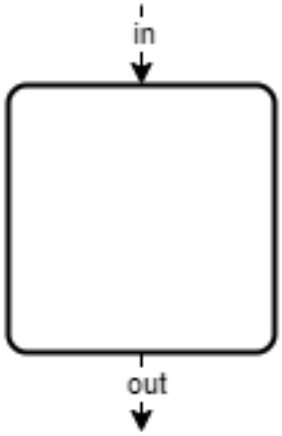
Blok decision merupakan blok yang digunakan untuk menggambarkan proses kondisional yang terdapat pada program. Blok ini juga digunakan untuk menggambarkan proses perulangan. Blok decision berbentuk belah ketupat dengan satu buah *input* dan dua buah *output*. Letak *output true* dan *output false* tidak harus sama dengan contoh yang diberikan tetapi dapat disesuaikan dengan kebutuhan asalkan masih jelas terdapat satu buah *input* dan dua buah *output* pada blok ini.



Gambar 4 Blok Decision

I.IV. Blok Process

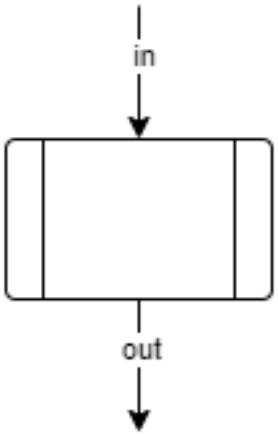
Blok process merupakan blok yang digunakan untuk merepresentasikan sebuah proses yang dilakukan oleh program. Blok ini berbentuk persegi dengan sebuah *input* dan sebuah *output*.



Gambar 5 Blok Process

I.V.Blok Predefined Process

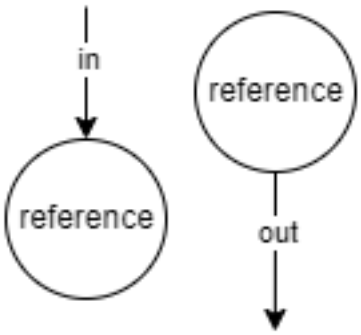
Blok ini merupakan blok yang digunakan untuk melambangkan sebuah fungsi buatan yang digunakan pada program utama. Blok ini memiliki bentuk persegi dengan tambahan dua garis di bagian dalam serta sebuah *input* dan sebuah *output*. Fungsi yang digunakan pada blok ini juga perlu digambarkan *flowchart*-nya pada bagian terpisah dari fungsi utama.



Gambar 6 Blok Predefined Process

I.VI. Blok Reference

Blok ini digunakan sebagai pemisah *flowchart* apabila terlalu panjang. Blok ini berbentuk lingkaran yang diisi dengan nomor atau huruf referensi serta terdiri dari sebuah *input* atau sebuah *output*.



Gambar 7 Blok Reference

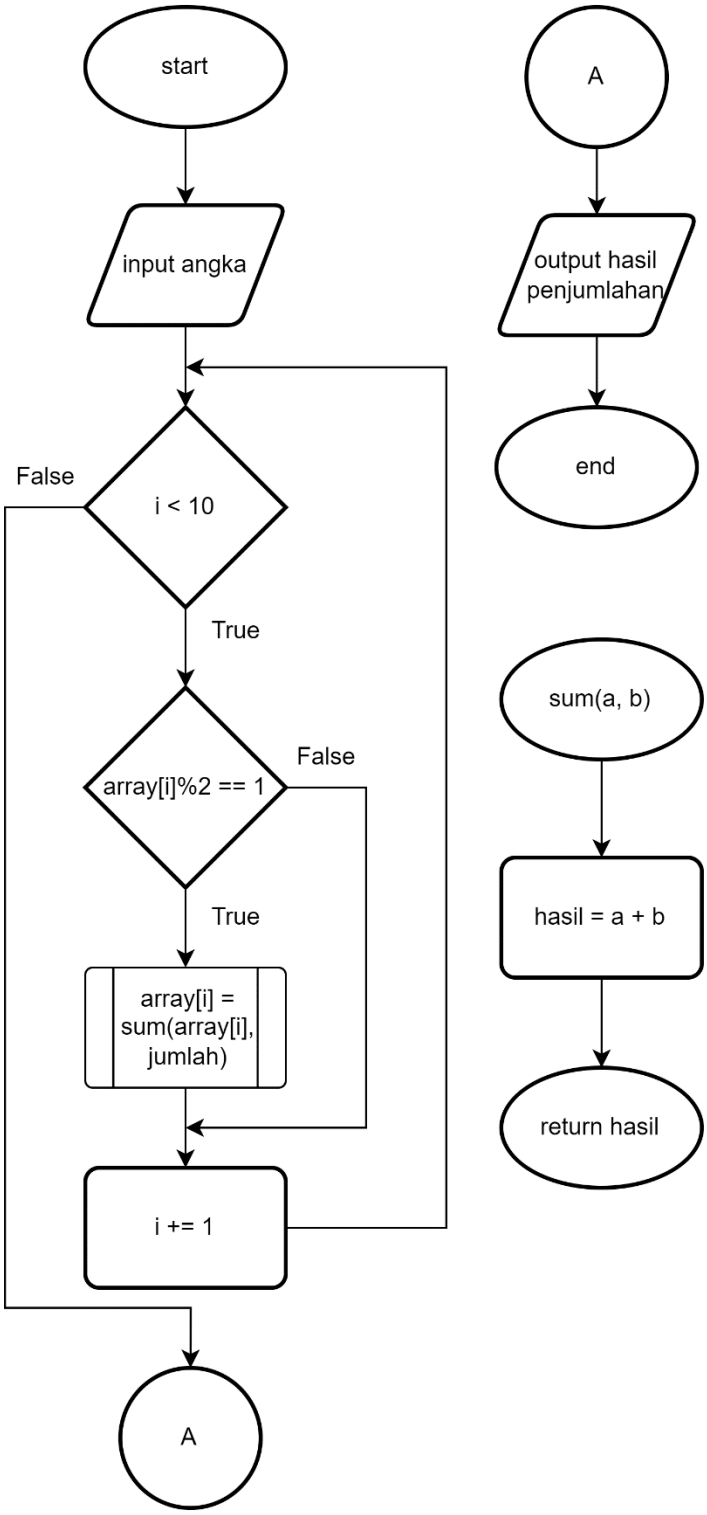
II. Contoh *Flowchart*

Diberikan contoh kode yang ditampilkan pada Source Code 1 berikut. Dengan kode tersebut akan dihasilkan *flowchart* yang sesuai pada Gambar 8. Kode yang diberikan merupakan kode yang memiliki sebuah fungsi untuk melakukan penjumlahan. Kode tersebut akan digunakan untuk melakukan penjumlahan dari setiap elemen ganjil dalam sebuah *array*.

Source Code 1 Contoh Kode untuk Pembuatan Flowchart

```
/*EL2208 Praktikum Pemecahan Masalah dengan C
*Modul             : Contoh kode untuk Flowchart
*Pembuat           : Dismas Widyanto
*Deskripsi         : Program untuk melakukan penjumlahan pada setiap
bilangan ganjil dalam sebuah array
*/


#include<stdio.h>

int sum(int a, int b){
      int hasil;

      hasil = a + b;

      return hasil;
}


int main(){
      int array[10] = {1,2,3,4,5,6,7,8,9,10};
      int jumlah;

      printf("Masukkan angka yang akan dijumlahkan: ");
      scanf("%d",&jumlah);

      for(int i=0;i<10;i++){
            if(array[i]%2 == 1){
                  array[i] = sum(array[i], jumlah);
            }
      }

      printf("Hasil:\n");
      for(int i=0;i<10;i++){
            printf("%d\n",array[i]);
      }
}
```

Flowchart yang dibuat terdiri dari dua buah bagian yaitu bagian `main` dan bagian fungsi `sum`. Perlu diperhatikan bahwa pemanggilan fungsi `sum` pada bagian `main` akan menggunakan blok predefined process. *Flowchart* pada bagian main juga diberikan contoh penggunaan blok reference untuk memberi tanda apabila sebuah *flowchart* terlalu panjang.



Gambar 8 Contoh Flowchart dari Kode

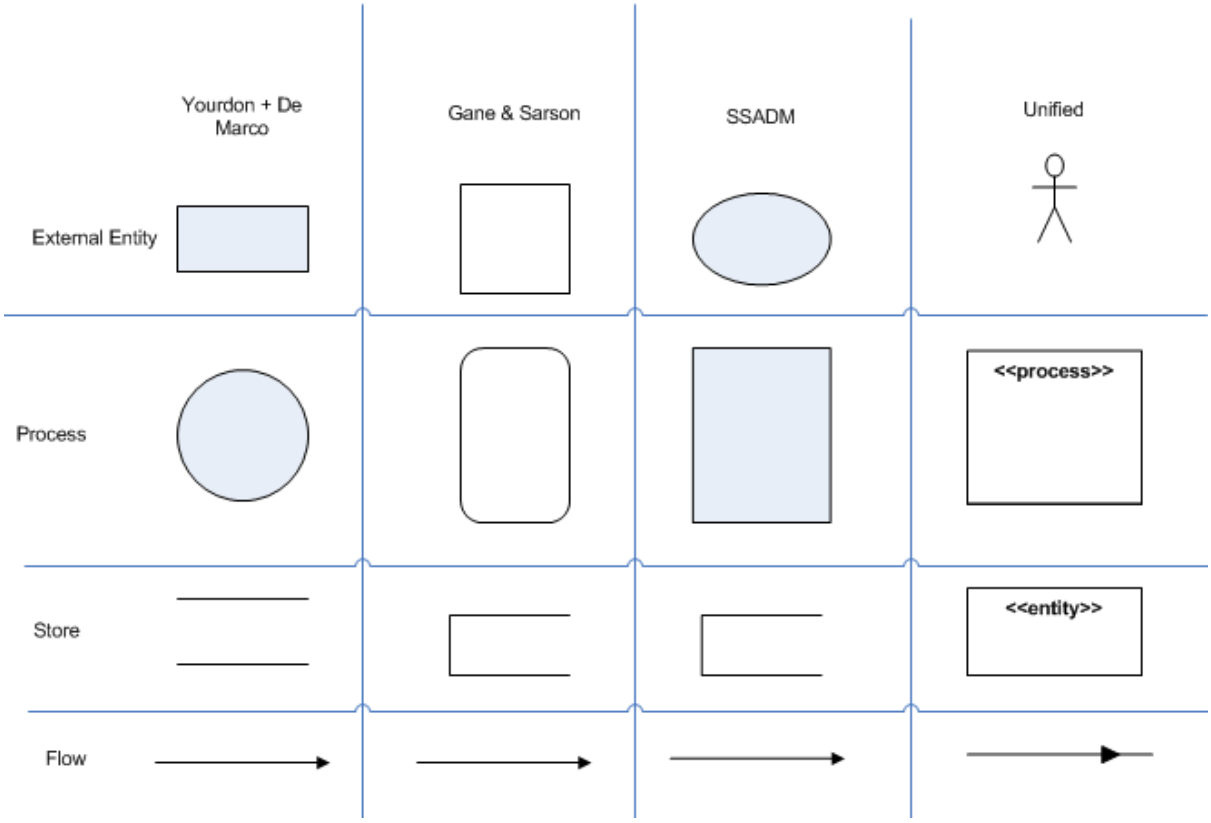
Perhatikan bahwa setiap blok pada *flowchart* hanya memiliki satu *input* dan *output* kecuali blok kondisional yang memiliki dua *output*. Perhatikan pula bahwa blok *Start/End* pada *flowchart* fungsi `sum` memuat parameter fungsi serta nilai yang di-*return* oleh fungsi tersebut.

DATA FLOW DIAGRAM

Sama seperti *flowchart*, pembuatan *Data Flow Diagram* atau DFD dibebaskan dengan catatan mengikuti kaidah pada umumnya, namun sangat disarankan untuk pembuatan DFD menggunakan aplikasi seperti draw.io (https://app.diagrams.net/) atau lucidchart (https://www.lucidchart.com/). Dalam dokumen ini akan digunakan aplikasi draw.io dalam pembuatan DFD.

I. Kaidah DFD

Terdapat empat buah simbol yang perlu Anda perhatikan dalam membuat DFD. Setiap simbol merepresentasikan hal yang berbeda-beda dan juga memiliki kegunaan dan kaidah yang berbeda-beda. Simbol-simbol tersebut dapat digambarkan dengan beberapa cara seperti pada Gambar 8.



Gambar 9 Berbagai Notasi Simbol DFD

Notasi DFD yang dapat Anda gunakan pada praktikum ini adalah Yourdon & De Marco atau Gane & Sarson. Berikut merupakan penjelasan dari masing-masing simbol.

I.I. External Entity

External Entity merupakan sebuah simbol yang menggambarkan orang, departemen, atau sistem eksternal yang memberikan data kepada program atau menerima data dari program. Simbol ini akan menggambarkan cara program Anda berinteraksi dengan dunia luar. Hal yang digambarkan oleh simbol ini merupakan hal berada di luar program Anda seperti pengguna program.

I.II. Process

Sesuai dengan namanya, simbol Process menggambarkan sebuah proses dalam program Anda yang menerima masukkan dan memberikan keluaran dengan bentuk atau konten yang berbeda. Simbol ini dapat menggambarkan satu proses sederhana atau beberapa proses sekaligus. Kompleksitas proses yang digambarkan oleh simbol ini akan dijelaskan pada bagian selanjutnya.

I.III. Data Store

Simbol ini menggambarkan sebuah tempat penyimpanan data yang akan digunakan oleh satu atau beberapa proses di masa depan. Dalam konteks praktikum ini, sebuah file eksternal yang digunakan oleh program dapat digambarkan sebagai sebuah Data Store dalam DFD Anda.

I.IV. Data Flow

Data Flow merupakan simbol yang merepresentasikan jalur pergerakan data dalam program Anda. Simbol ini digambarkan dengan sebuah panah dengan teks yang menyatakan data yang melalui jalur tersebut. Terdapat beberapa peraturan dalam menggambarkan Data Flow, peraturan-peraturan tersebut adalah:

1. External Entity tidak dapat memberikan data ke External Entity lain tanpa melalui sebuah Process
2. External Entity tidak dapat memberikan data ke sebuah Data Store tanpa melalui sebuah Process
3. Data Store tidak dapat memberikan data kepada sebuah External Entity tanpa melalui sebuah Process
4. Data Store tidak dapat mengirim atau menerima data dari Data Store lain tanpa melalui sebuah Process

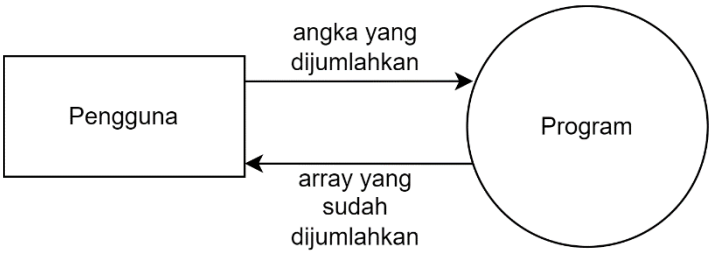
I.V. Pembuatan DFD

DFD dibuat dalam beberapa level yang dimulai dari level 0. DFD level 0 disebut juga dengan Context Diagram. DFD level 0 menggambarkan interaksi program dengan dunia luar sehingga hanya berisikan External Entity dan satu Process yang merangkum semua proses program.

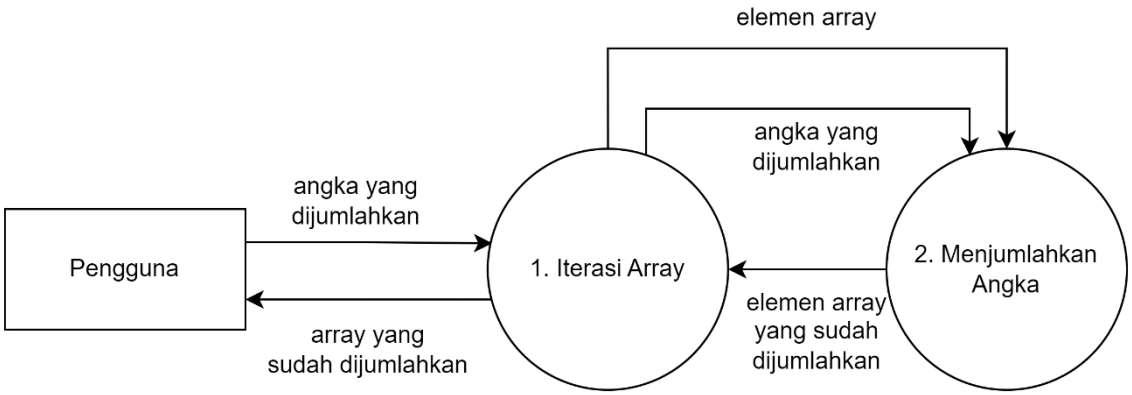
Process pada DFD level 0 akan dipecah menjadi beberapa Process lainnya di DFD level 1. Perlu diingat bahwa Data Flow pada DFD level 1 harus konsisten dengan Data Flow pada DFD level 0. Jika diperlukan, Process-Process pada DFD level 1 dapat dipecah lebih lanjut menjadi beberapa Process lainnya di DFD level 2.

II. Contoh DFD

Dengan kode Source Code 1, dapat dihasilkan DFD level 0 dan DFD level 1 menggunakan notasi Yourdan & DeMarco seperti pada Gambar 10 dan Gambar 11. Perhatikan bahwa DFD menggambarkan aliran data pada program, proses-proses yang ada dalam program, serta interaski progran dengan dunia luar.



Gambar 10 Contoh DFD Level 0 atau Context Diagram



Gambar 11 Contoh DFD Level 1

ANALISIS KOMPLEKSITAS RUANG DAN WAKTU

Analisis komplekitas ruang dan waktu merupakan bagian dari analisis algoritma yang dilakukan secara teoritis (apriori). Analisis algoritma dilakukan untuk menentukan kompleksitas dari sebuah algoritma berdasarkan jumlah waktu, *storage*, atau resource lainnya yang dibutuhkan untuk menjalankan algoritma tersebut hingga selesai. Sesuai dengan namanya, kompleksitas ruang merepresentasikan ukuran ruang *memory* yang digunakan oleh sebuah algoritma selama siklus hidupnya sedangkan kompleksitas waktu merepresentasikan jumlah waktu yang dibutuhkan untuk menjalankan sebuah algoritma sampai selesai. Kompleksitas ruang dan waktu dapat dinotasikan menggunakan beberapa cara, namun notasi yang digunakan pada praktikum ini adalah notasi Big O.

I. Analisis Kompleksitas Ruang

Analisis kompleksitas ruang dapat dilakukan dengan langkah-langkah berikut:

1. Menghitung ruang *memory* yang dialokasikan secara statis
2. Menghitung ruang *memory* yang dialokasikan secara dinamis (*Auxiliary Space*)
3. Menghitung total ruang yang dibutuhkan
4. Menyatakan analisis menggunakan notasi Big O

Ruang yang dialokasikan secara statis berasal dari deklarasi variabel-variabel yang bersifat pasti (contoh: variabel yang tidak dideklarasi di dalam blok kondisional atau *loop*). Ruang yang dialokasikan secara dinamis berasal dari deklarasi variabel yang tidak pasti atau ukurannya dapat berubah selama eksekusi algoritma (contoh: alokasi *memory* di *stack* ketika memanggil fungsi, deklarasi variabel dalam blok kondisional, alokasi *memory* dinamis, dll.).

II. Analisis Kompleksitas Waktu

Analisis kompleksitas waktu dapat dilakukan dengan langkah-langkah berikut:

1. Menghitung berapa kali s*tatement-statement* dalam program dieksekusi
2. Menyatakan analisis menggunakan notasi Big O

Mirip dengan analisis kompleksitas ruang, jumlah eksekusi dapat bernilai statis maupun dinamis. Jumlah eksekusi *statement* dapat bernilai dinamis karena berada di dalam sebuah *loop* atau karena berada di dalam sebuah fungsi rekursif.

III. Contoh Analisis Kompleksitas Ruang dan Waktu

Sebagai contoh, akan dilakukan analisis kompleksitas ruang dan waktu untuk kode Source Code 1 yang merupakan modifikasi dari Source Code 1. Dengan itu, analisis dapat dilakukan seperti pada subbab-subbab berikut.

Source Code 2 Contoh Kode untuk Analisis Kompleksitas Ruang dan Waktu

```
/*EL2208 Praktikum Pemecahan Masalah dengan C
*Modul             : Contoh kode untuk Flowchart
*Pembuat           : Elkhan Julian Brillianshah
*Deskripsi         : Program untuk melakukan penjumlahan pada setiap
bilangan ganjil dalam sebuah array
*/

#include<stdio.h>

int sum(int a, int b){
	int hasil;

	hasil = a + b;

	return hasil;
}

int main(){
	int N;
	int jumlah;

	printf("Masukkan jumlah angka yang akan dimasukkan: ");
	scanf("%d",&N);

	int array[N];
	for(int i=0;i<N;i++){
		array[i] = i + 1;
	}

	printf("Masukkan angka yang akan dijumlahkan: ");
	scanf("%d",&jumlah);

	for(int i=0;i<N;i++){
		if(array[i]%2 == 1){
			array[i] = sum(array[i], jumlah);
		}
	}

	printf("Hasil:\n");
	for(int i=0;i<N;i++){
		printf("%d\n",array[i]);
	}
}
```

II.II. Analisis Kompleksitas Ruang

Analisis kompleksitas ruang dimulai dengan menghitung ruang *memory* yang digunakan program dan dialokasikan secara statis. Berdasarkan kode, dapat diamati bahwa variabel yang pasti akan dideklarasikan adalah variabel `jumlah, i,` dan `N` saja. Untuk menyederhanakan perhitungan, akan diasumsikan bahwa tipe data `int` memiliki ukuran 1 satuan ruang sehingga total ruang yang dialokasikan secara statis adalah 3 satuan ruang.

Setelah itu, akan dilakukan perhitungan ruang *memory* yang dialokasikan secara dinamis. Berdasarkan kode, dapat diamati bahwa satu-satunya variabel yang ukurannya tidak pasti adalah variabel `array` saja yang memiliki panjang sebesar `N` sehingga berukuran N * 1. Selain itu, terdapat variabel `hasil` yang dideklarasikan di dalam fungsi `sum`. Dengan itu, didapat total ruang yang dialokasikan secara dinamis adalah N + 1 satuan ruang.

Total ruang yang dibutuhkan dapat dihitung dengan menjumlahkan ruang yang dialokasikan secara statis dan dinamis. Berdasarkan perhitungan-perhitungan sebelumnya, didapat bahwa total kebutuhan ruang adalah N + 4 satuan ruang. Notasi Big O dari hasil tersebut dapat dibuat dengan mengambil suku dengan derajat terbesar dan menghilangkan koefisien dari suku tersebut sehingga didapat kompleksitas ruang dalam notasi Big O adalah O(N).

II.III. Analisis Kompleksitas Waktu

Analisis kompleksitas waktu dilakukan dengan menghitung berapa kali *statement- statement* dalam program dieksekusi. Demi kesederhanaan, fungsi *scanf* dan *printf* diasumsikan akan memiliki waktu eksekusi yang konstan sehingga dapat diperlakukan sebagai satu buah *statement*. Jika *loop-loop* dalam fungsi main dihiraukan, maka didapat bahwa terdapat 7 baris *statement* yang dieksekusi hanya sekali saja (deklarasi variabel, fungsi *printf*, dan fungsi *scanf*).

Pada *for loop* pertama dalam fungsi main, dapat diamati bahwa *assigment* elemen dari variabel *array* akan dilakukan sebanyak N kali sehingga bagian tersebut dapat dianggap akan mengeksekusi *statement* sebanyak N kali. *For loop* kedua di dalam fungsi main berisikan dua buah statement (*statement if* dan *statement assignment*), namun terdapat eksekusi fungsi sum yang akan mengeksekusi 3 buah *statement* sehingga bagian tersebut dapat dianggap akan mengeksekusi *statement* sebanyak N * (1 + 1 + 3) atau 5N kali. Isi dari *loop* terakhir pada fungsi main merupakan eksekusi dari fungsi *printf* saja sehingga dapat dianggap bahwa bagian tersebut akan mengeksekusi *statement* tersebut sebanyak N kali. Dengan itu, didapat bahwa program secara keseluruhan akan mengeksekusi *statement* 7N + 7 kali yang dapat ditulis dalam notasi Big O sebagai O(N).

Referensi:

[1] https://www.geeksforgeeks.org/analysis-algorithms-big-o-analysis/

[2] https://www.tutorialspoint.com/time-and-space-complexity-in-data-str

[3] https://www.geeksforgeeks.org/understanding-time-complexity-simple-examples/

MODUL I
OVERVIEW OF C LANGUAGE

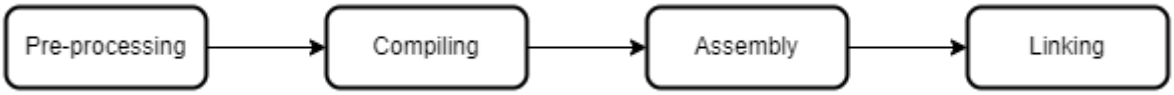
I. Tujuan

1. Melatih kemampuan untuk melakukan dekomposisi masalah
2. Memberikan pemahaman praktikan terkait proses kompilasi pada Bahasa C
3. Memberikan pemahaman praktikan terkait dasar-dasar Bahasa C seperti variabel, tipe data, proses *assignment*, dan *input/output* sederhana
4. Memberikan pemahaman praktikan terkait struktur kondisional dan perulangan pada Bahasa C
5. Memberikan pemahaman praktikan terkait bentuk data *array*

II. Materi

II.I. Kompilasi Bahasa C

Proses kompilasi pada Bahasa C terbagi menjadi empat buah tahap seperti digambarkan pada Gambar 12. Masing-masing proses akan menerima masukan dan mengubahnya menjadi sebuah keluaran hingga akhirnya menjadi sebuah program yang dapat dieksekusi.



Gambar 12 Proses Kompilasi pada Bahasa C

Masing-masing proses memiliki kegunaannya masing-masing seperti dijelaskan berikut ini.

1. Pre-processing

   Pada proses ini sebuah *file* dengan ekstensi `.c` akan diolah terlebih dahulu sebelum *compiler* melakukan tugasnya. Pada tahap ini akan menghasilkan sebuah *file* dengan ekstensi `.i`. Pengolahan yang dilakukan pada tahap ini meliputi

   - Menghapus komentar
   - Menjabarkan definisi makro yang digunakan
   - Menjabarkan definisi dari *file* yang di-*include*

   Contohnya pada definisi `#define EULER 2.71828` akan diubah semua variabel `EULER` menjadi nilai yang sesuai. Definisi `#include <stdio.h>` akan diganti dengan isi dari *file* yang bersangkutan.

2. Compiling

   Tahap ini akan melakukan kompilasi *file* dengan ekstensi `.i` menjadi *file assembly* dengan ekstensi `.s`. *File assembly* merupakan sebuah *file* yang berisi baris-baris instruksi yang dapat dipahami oleh mesin. Instruksi ini biasa dikenal sebagai *instruction set*.

3. Assembly

   Pada tahap ini *file assembly* yang dihasilkan sebelumnya akan diubah menjadi sebuah *file* obyek dengan ekstensi `.o`. *File* obyek merupakan suatu *file* biner yang berisi instruksi yang dapat dipahami oleh mesin.

4. Linking

   Tahap ini akan menggabungkan *file* obyek yang dihasilkan sebelumnya dengan definisi fungsi yang dipanggil pada program yang dibuat. Selain itu, pada tahap ini juga ditambahkan baris kode yang diperlukan untuk memulai dan mengakhiri sebuah program.

Terdapat berbagai jenis *compiler* yang dapat digunakan untuk melakukan kompilasi Bahasa C. Namun, pada praktikum ini akan digunakan GCC sebagai *compiler* standar. Untuk melakukan kompilasi Bahasa C dengan GCC dapat digunakan *syntax* berikut

```
gcc -o <nama file output> <nama file source> -l<nama library>
```

Sebagai contoh apabila Anda memiliki *file* `main.c` sebagai program utama dan `lib.c` sebagai definisi fungsi serta dalam program Anda menggunakan *library* `math.h` maka Anda dapat melakukan kompilasi dengan menjalankan *syntax* berikut

Pada Sistem Operasi Linux **`gcc -o main main.c lib.c -lm`**
Pada Sistem Operasi Windows **`gcc -o main.exe main.c lib.c -lm`**

Referensi:

[1] https://www.geeksforgeeks.org/compiling-a-c-program-behind-the-scenes/

II.II. Tipe Data dan Variabel

Dalam Bahasa C terdapat dua buah tipe data dasar yang dapat digunakan yakni tipe integer dan tipe *floating-point*. Masing-masing tipe dijabarkan lebih lanjut melalui tabel berikut.

Tabel 1 Tipe Data Integer

Tipe	Ukuran Data	Rentang Nilai
char	1 byte	-128 sampai 127 atau 0 sampai 255
unsigned char	1 byte	0 sampai 255
signed char	1 byte	-128 sampai 127
int	2 or 4 bytes	-32,768 sampai 32,767 atau -2,147,483,648 sampai 2,147,483,647
unsigned int	2 or 4 bytes	0 sampai 65,535 atau 0 sampai 4,294,967,295
short	2 bytes	-32,768 sampai 32,767
unsigned short	2 bytes	0 sampai 65,535
long	8 bytes or (4bytes for 32 bit OS)	-9223372036854775808 sampai 9223372036854775807
unsigned long	8 bytes	0 sampai 18446744073709551615

Tabel 2 Tipe Data Floating-Point

Tipe Data	Ukuran Data	Rentang Nilai	Presisi
float	4 byte	1.2×10^{-38} sampai 3.4×10^{38}	6 angka desimal
double	8 byte	2.3×10^{-308} sampai 1.7×10^{308}	15 angka desimal
long double	10 byte	3.4×10^{-4932} sampai 1.1×10^{4932}	19 angka desimal

Tipe data tersebut digunakan untuk mendefinisikan variabel yang akan digunakan dalam program. Proses pendefinisian variabel dapat menggunakan *syntax* berikut

```
type variable_list;
```

Dengan `type` merupakan tipe data yang valid dan `variable_list` merupakan nama variabel yang terdiri dari satu atau lebih variabel. Sebagai contoh, berikut merupakan pendefinisian beberapa jenis variabel. Apabila terdapat nama variabel lebih dari satu maka dapat menggunakan tanda koma sebagai pemisah.

```
int i, j, k;
char c, ch;
float f, salary;
double d;
```

Pendefinisian variabel juga dapat dilakukan bersamaan dengan memberikan nilai awal. Proses ini dapat dilakukan dengan menggunakan *syntax* berikut

```
type variable_name = value;
```

Syntax di atas mirip dengan *syntax* sebelumnya hanya saja nilai awal langsung diberikan ketika mendefinisikan variabel. Contoh pendefinisian variabel dengan nilai awal dapat dilihat di bawah ini

```
int d = 3, f = 5;
byte z = 22;
char x = 'x';
```

Referensi:

[1] https://www.tutorialspoint.com/cprogramming/c_data_types.htm

[2] https://www.tutorialspoint.com/cprogramming/c_variables.htm

II.III. Operator

Operator merupakan sebuah simbol yang berfungsi sebagai penanda bagi *compiler* untuk melakukan suatu operasi. Dalam Bahasa C terdapat beberapa jenis operator yaitu *Arithmetic Operators*, *Relational Operators*, *Logical Operators*, *Bitwise Operators*, dan *Assignment Operators*. Selain kelima jenis operator tersebut ada beberapa operator penting lainnya yang tidak masuk ke dalam lima kategori tersebut. Penjelasan dari masing-masing jenis operator akan dijabarkan melalui bagian berikut ini.

1. Arithmetic Operators

   Arithmetic operators merupakan operator yang menandakan perintah untuk melakukan operasi matematika. Berikut penjelasan masing-masing operator yang masuk ke dalam kategori ini. Contoh berikut menggunakan nilai A yaitu 10 dan nilai B yaitu 20.

Tabel 3 Arithmetic Operators

Operator	Deskripsi	Contoh
+	Melakukan penjumlahan.	A + B = 30
−	Melakukan pengurangan.	A − B = -10
*	Melakukan perkalian.	A * B = 200
/	Melakukan pembagian.	B / A = 2
%	Mengambil nilai sisa hasil bagi (modulo).	B % A = 0
++	Menambah nilai tipe data integer sebanyak 1.	A++ = 11
--	Mengurangi nilai tipe data integer sebanyak 1.	A-- = 9

2. Relational Operators

   Relational operators merupakan simbol yang menandakan hubungan dari dua buah nilai yang dibandingkan. Berikut ditampilkan dalam tabel penjelasan dari masing-masing simbol yang terdapat pada jenis operator ini. Contoh di bawah menggunakan nilai A yaitu 10 dan nilai B yaitu 20.

Tabel 4 Relational Operators

Operator	Deskripsi	Contoh
==	Membandingkan kedua nilai apakah sama. Jika kedua nilai sama maka akan dihasilkan kondisi True.	(A == B) is False.
!=	Membandingkan kedua nilai apakah berbeda. Jika kedua nilai berbeda maka akan dihasilkan kondisi True.	(A != B) is True.

>	Membandingkan apakah nilai di sebelah kiri lebih besar dari nilai di sebelah kanan. Jika benar maka akan dihasilkan kondisi True.	(A > B) is False.
<	Membandingkan apakah nilai di sebelah kiri lebih kecil dari nilai di sebelah kanan. Jika benar maka akan dihasilkan kondisi True.	(A < B) is True.
>=	Membandingkan apakah nilai di sebelah kiri lebih besar atau sama dari nilai di sebelah kanan. Jika benar maka akan dihasilkan kondisi True.	(A >= B) is False.
<=	Membandingkan apakah nilai di sebelah kiri lebih kecil atau sama dari nilai di sebelah kanan. Jika benar maka akan dihasilkan kondisi True.	(A <= B) is True.

3. Logical Operators

   Logical operators merupakan simbol yang menandakan perintah untuk melakukan operasi kondisi logika. Berikut ini dijelaskan simbol yang merupakan operator logika dalam Bahasa C. Contoh pada tabel menggunakan nilai A yaitu 1 dan nilai B yaitu 0.

Tabel 5 Logical Operators

Operator	Deskripsi	Contoh
&&	Operator logika AND. Ketika kedua kondisi bernilai True atau tidak 0 maka akan dihasilkan kondisi True.	(A && B) is False.
||	Operator logika OR. Ketika salah satu dari kedua kondisi bernilai True atau tidak 0, maka akan dihasilkan kondisi True.	(A || B) is True.
!	Operator logika NOT. Berguna untuk melakukan negasi dari suatu logika.	!(A && B) is True.

4. Bitwise Operators

   Bitwise operators merupakan simbol untuk melakukan perintah operasi pada domain biner. Penjelasan untuk masing-masing simbol dijelaskan melalui tabel berikut. Contoh berikut menggunakan nilai A yaitu 60 (0011 1100) dan nilai B yaitu 13 (0000 1101).

Tabel 6 Bitwise Operators

Operator	Deskripsi	Contoh
&	Operator biner AND	(A & B) = 12, i.e., 0000 1100
|	Operator biner OR	(A | B) = 61, i.e., 0011 1101
^	Operator biner XOR	(A ^ B) = 49, i.e., 0011 0001
~	Operator biner NOT atau negasi	(~A) = ~(60), i.e,. - 011 1100
<<	Operator biner LEFT SHIFT	A << 2 = 240 i.e., 1111 0000
>>	Operator biner RIGHT SHIFT	A >> 2 = 15 i.e., 0000 1111

5. Assignment Operators

Assignment operators merupakan simbol perintah untuk menyimpan dan/atau melakukan operasi. Penjelasan dari masing-masing simbol dapat diperhatikan pada tabel di bawah.

Tabel 7 Assignment Operators

Operator	Deskripsi	Contoh
=	Operator standar *assignment*. Menyimpan nilai dari operasi di sisi kanan ke variabel di sisi kiri.	C = A + B menyimpan nilai A + B ke dalam C
+=	Melakukan operasi penjumlahan antara sisi kanan dan kiri kemudian menyimpang nilainya pada variabel sisi kiri.	C += A setara dengan C = C + A
-=	Melakukan operasi pengurangan antara sisi kanan dan kiri kemudian menyimpang nilainya pada variabel sisi kiri.	C -= A setara dengan C = C - A
*=	Melakukan operasi perkalian antara sisi kanan dan kiri kemudian menyimpang nilainya pada variabel sisi kiri.	C *= A setara dengan C = C * A
/=	Melakukan operasi pembagian antara sisi kanan dan kiri kemudian menyimpang nilainya pada variabel sisi kiri.	C /= A setara dengan C = C / A
%=	Melakukan operasi modulo antara sisi kanan dan kiri kemudian menyimpang nilainya pada variabel	C %= A setara dengan C = C % A

	sisi kiri.	
<<=	Melakukan operasi LEFT SHIFT antara sisi kanan dan kiri kemudian menyimpang nilainya pada variabel sisi kiri.	C <<= 2 setara dengan C = C << 2
>>=	Melakukan operasi RIGHT SHIFT antara sisi kanan dan kiri kemudian menyimpang nilainya pada variabel sisi kiri.	C >>= 2 setara dengan C = C >> 2
&=	Melakukan operasi biner AND antara sisi kanan dan kiri kemudian menyimpang nilainya pada variabel sisi kiri.	C &= 2 setara dengan C = C & 2
^=	Melakukan operasi biner XOR antara sisi kanan dan kiri kemudian menyimpang nilainya pada variabel sisi kiri.	C ^= 2 setara dengan C = C ^ 2
|=	Melakukan operasi biner OR antara sisi kanan dan kiri kemudian menyimpang nilainya pada variabel sisi kiri.	C |= 2 setara dengan C = C | 2

6. Operator Lainnya

   Selain kelima jenis operator di atas terdapat beberapa operator lainnya yang penting dan umum digunakan dalam Bahasa C. Bagian ini membahas operator penting lainnya yang belum termasuk ke lima jenis di atas.

Tabel 8 Operator Penting Lainnya

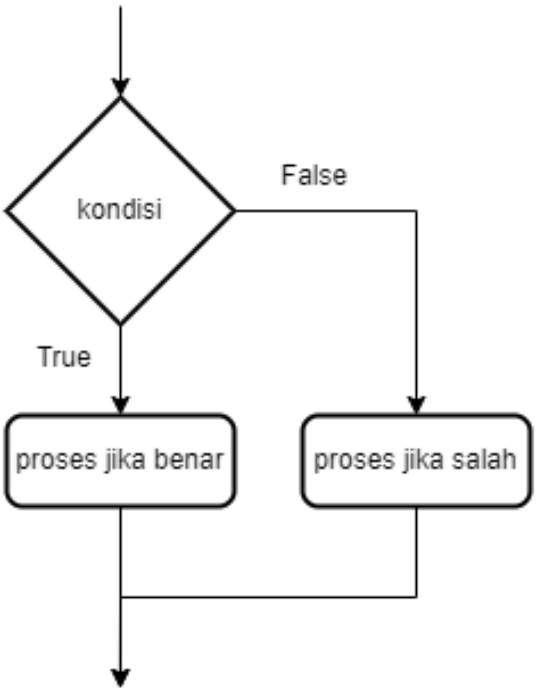
Operator	Deskripsi	Contoh
&	Me-*return* alamat memori dari variabel yang digunakan.	`scanf("%d", &i);` atau `mem = &a`, contoh pertama akan menyimpan nilai pada alamat variabel tersebut sedangkan contoh kedua artinya alamat memori dari variabel `a` disimpan pada variabel `mem`.
*	Pointer ke alamat memori dari sebuah variabel.	`*a`, merujuk pada data yang disimpan di memori a.

Referensi:

[1] https://www.tutorialspoint.com/cprogramming/c_operators.htm

II.IV. Kondisional

Struktur kondisional merupakan salah satu struktur yang sangat penting dalam pemrograman. Sebagian besar pola pada pemrograman akan menggunakan kondisional untuk menentukan pilihan yang mungkin terjadi dalam program tersebut. Kondisional akan memiliki struktur yang terdiri dari sebuah kondisi, operasi ketika kondisi benar, dan operasi ketika kondisi salah. Struktur tersebut dapat diterjemahkan menjadi *flowchart* berikut.



Gambar 13 Struktur Kondisional

Terdapat dua jenis kondisional yang terdapat dalam Bahasa C yaitu *if statement* dan *switch statement*. Masing-masing memiliki *syntax* yang berbeda. Penjelasan dari masing-masing jenis akan dijabarkan pada poin berikut ini.

1. If statement

   Kondisional jenis ini merupakan jenis yang sering digunakan dan ditemukan dalam pemrograman. Secara umum *if statement* akan mengikuti struktur seperti Gambar 13 dengan *syntax* seperti berikut.

```
if(kondisi){
```

```
      kode jika benar
}
else{
      kode jika salah
}
```

Struktur di atas dapat diubah atau disesuaikan sesuai kebutuhan. Apabila hanya diperlukan kode yang benar saja maka kode dapat diubah menjadi seperti berikut.

```
if(kondisi){
      kode jika benar
}
```

Struktur kondisional *if statement* juga dapat digunakan untuk lebih dari dua kondisi dengan menggunakan *syntax* berikut.

```
if(kondisi_1){
      kode jika benar pertama
}
else if(kondisi_2){
      kode jika benar kedua
}
else if ...


else{
      kode jika semua kondisi salah
}
```

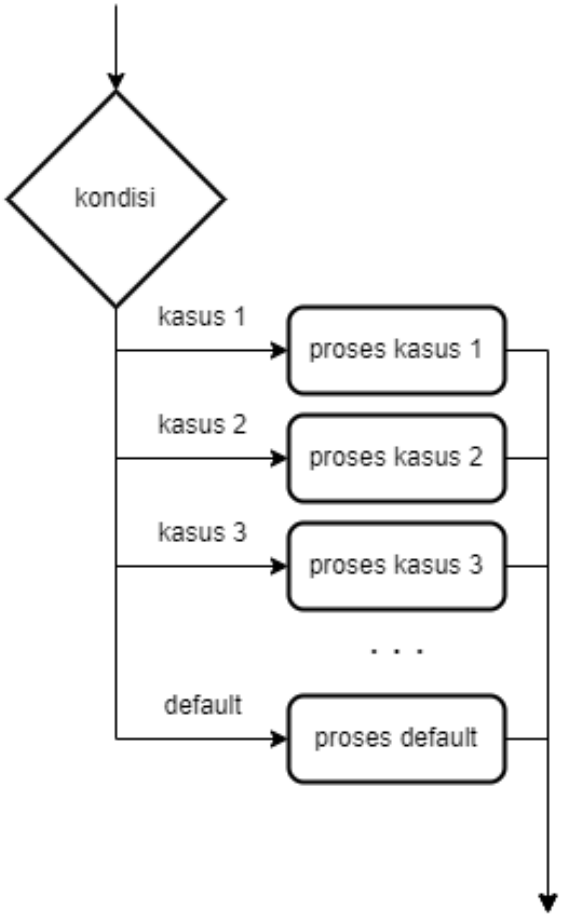
Selain variasi yang diberikan di atas, Anda juga dapat menggunakan *if statement* di dalam sebuah *if statement* lainnya. Struktur seperti ini biasa disebut dengan *nested if* dan memiliki *syntax* seperti berikut ini.

```
if(kondisi){
      if(kondisi_1){
            kode jika kondisi dan kondisi_1 benar
      }
      else{
            kode jika kondisi benar dan kondisi_1 salah
      }
}
else{
      if(kondisi_2){
            kode jika kondisi salah dan kondisi_2 benar
```

```
        }
        else{
              kode jika kondisi dan kondisi_2 salah
        }
}
```

2. Switch statement

   Switch statement merupakan struktur yang biasa digunakan ketika terdapat lebih dari dua buah pilihan. Struktur ini tidak terlalu sering digunakan dalam program dibanding dengan *if statement*. Secara umum *switch statement* akan memiliki struktur seperti pada *flowchart* berikut.



Gambar 14 Struktur Switch Statement

Flowchart di atas menunjukkan bahwa struktur ini sangat cocok ketika pilihan yang tersedia ada banyak. *Switch statement* secara umum dapat dibentuk dengan menuliskan *syntax* berikut.

```
switch(variabel kondisi){
      case kondisi_1:
            kode kondisi_1
            break;
      case kondisi_2:
            kode kondisi_2
            break;

      ...

      default:
            kode default
}
```

Seperti pada *if statement*, *syntax switch statement* dapat diubah dan disesuaikan sesuai kebutuhan. *syntax* tersebut juga dapat diubah menjadi *nested switch*. Berikut merupakan *syntax* ketika menggunakan *nested switch*.

```
switch(kondisi_a){
      case kondisi_a_1:
            switch(kondisi_b){
                  case kondisi_b_1:
                        kode kondisi_b_1
                        break;

                  case kondisi_b_2:
                        kode kondisi_b_2
                        break;

                  ...
                  default:
                        kode default
            }
            break;

      case kondisi_a_2:
            kode kondisi_2
            break;

      ...

      default:
            kode default
```

```
}
```

Referensi:

[1] https://www.tutorialspoint.com/cprogramming/c_decision_making.htm

[2] https://www.tutorialspoint.com/cprogramming/if_statement_in_c.htm

[3] https://www.tutorialspoint.com/cprogramming/if_else_statement_in_c.htm

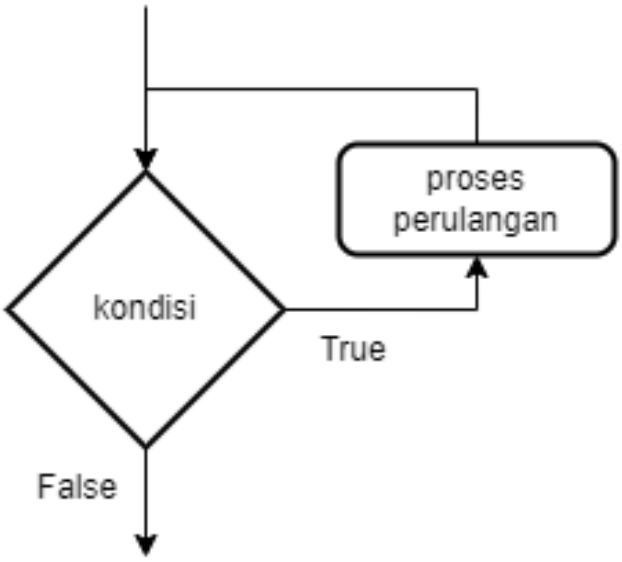
[4] https://www.tutorialspoint.com/cprogramming/nested_if_statements_in_c.htm

[5] https://www.tutorialspoint.com/cprogramming/switch_statement_in_c.htm

[6] https://www.tutorialspoint.com/cprogramming/nested_switch_statements_in_c.htm

II.V. Perulangan

Perulangan merupakan salah satu struktur yang sering digunakan dalam program selain kondisional. Sesuai namanya, struktur perulangan akan mengulangi bagian program tertentu hingga dihentikan oleh kondisi tertentu. Secara umum struktur perulangan akan mengikuti *flowchart* berikut.



Gambar 15 Struktur Perulangan

Dalam Bahasa C terdapat beberapa jenis perulangan yang dapat digunakan yaitu *while loop*, *for loop*, dan *do while loop*. Masing-masing jenis perulangan akan dijelaskan lebih lanjut pada poin berikut ini.

1. While loop

While loop akan melakukan perulangan secara terus menerus selama kondisi yang diberikan bernilai benar. Perulangan akan berhenti ketika kondisi yang diberikan bernilai salah. *While loop* akan memiliki *syntax* seperti berikut.

```
while(kondisi){
      kode perulangan
}
```

2. For loop

   For loop merupakan perulangan yang digunakan untuk mengulangi kode program apabila jumlah perulangannya sudah diketahui. Perulangan ini merupakan jenis yang sering digunakan dalam menyusun suatu program. *Syntax* dari struktur *for loop* dapat dilihat di bawah ini.

```
for(inisialisasi; kondisi; increment){
      kode perulangan
}
```

   Dari *syntax* di atas dapat dilihat bahwa terdapat tiga bagian yang menyusun *for loop* yaitu inisialisasi, kondisi, dan juga *increment*. Inisialisasi merupakan bagian untuk mendeklarasikan variabel kontrol dan nilainya. Bagian kondisi digunakan untuk mengevaluasi variabel kontrol apakah bernilai benar atau bernilai salah. *Increment* merupakan bagian untuk mengubah variabel kontrol yang digunakan sebelum dievaluasi kembali pada bagian kondisi.

3. Do while loop

   Do while loop merupakan struktur perulangan yang mirip dengan *while loop*. Pembeda dari *do while loop* dari *while loop* adalah proses evaluasi kondisi dilakukan di akhir setiap setelah kode perulangan dieksekusi. Apabila evaluasi kondisi bernilai benar maka kode perulangan akan dieksekusi kembali hingga evaluasi kondisi bernilai salah. *Syntax* berikut dapat digunakan untuk menyusun *do while loop*.

```
do{
```

```
      kode perulangan
}while (kondisi)
```

Referensi:

[1] https://www.tutorialspoint.com/cprogramming/c_loops.htm

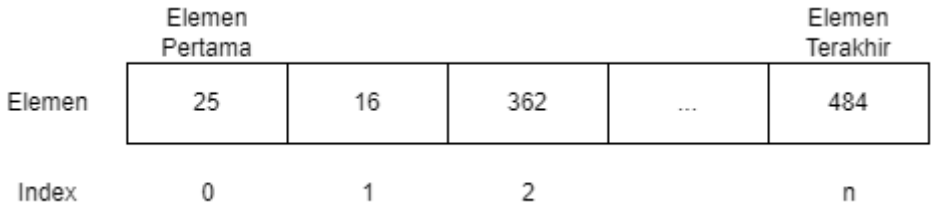
[2] https://www.tutorialspoint.com/cprogramming/c_while_loop.htm

[3] https://www.tutorialspoint.com/cprogramming/c_for_loop.htm

[4] https://www.tutorialspoint.com/cprogramming/c_do_while_loop.htm

II.VI. Array

Array adalah salah satu struktur data yang dapat menyimpan beberapa buah nilai dengan tipe data yang sama. Setiap nilai yang disimpan dalam sebuah *array* dapat diakses menggunakan *index*. Sebagai ilustrasi, gambar berikut menunjukkan bagaimana sebuah *array* menyimpan data.



Gambar 16 Ilustrasi Array

Dari gambar tersebut terlihat bahwa *array* akan memiliki banyak nilai/elemen sesuai dengan deklarasi yang diberikan. Masing-masing nilai akan menempati ruang memori secara terurut dari elemen pertama pada alamat memori yang paling kecil hingga elemen terakhir pada alamat memori yang paling besar. Masing-masing nilai dapat diakses menggunakan *index* yang berurut dari *index* 0 hingga *index* terakhir.

Deklarasi *array* dalam Bahasa C dapat dilakukan dengan mengikuti *syntax* berikut

```
type arrayName [ arraySize ];
```

dengan `type` merupakan tipe data dari *array*, `arrayName` merupakan variabel *array* yang digunakan, dan `arraySize` merupakan ukuran *array* yang dibuat. Seperti deklarasi variabel, deklarasi *array* juga dapat langsung diinisialisasi dengan menggunakan *syntax* berikut

```
type arrayName [ arraySize ] = {value_1, value_2, …, value_n};
```

Perlu diperhatikan bahwa banyaknya nilai yang diinisialisasi tidak boleh lebih dari ukuran *array*. Contoh terkait pendefinisian *array* dapat Anda lihat pada kode berikut.

```
double balance[5];

double balance[5] = {1000.0, 2.0, 3.4, 7.0, 50.0};
```

Setelah mempelajari proses mendefinisikan *array* maka berikutnya akan dipelajari proses untuk mengakses *array*. Sebuah *array* dapat diakses elemennya dengan menggunakan *index* yang sesuai. Seperti diilustrasikan pada Gambar 16, setiap elemen memiliki *index* yang sesuai. *Syntax* yang digunakan dalam mengakses *array* adalah sebagai berikut.

```
arrayName [ index ]
```

Dengan menggunakan *syntax* tersebut kita dapat mengakses salah satu elemen pada *array* dan melakukan operasi. Operasi tersebut dapat berupa menyimpan nilai atau mengambil nilai. Sebagai contoh, terdapat *array* `balance` dengan isi seperti contoh sebelumnya. *Array* tersebut dapat diambil nilainya dan diubah nilainya.

```
double balance[5] = {1000.0, 2.0, 3.4, 7.0, 50.0};

double number = balance[3] //number bernilai 7.0

balance[2] = 4.8 //balance berisi {1000.0, 2.0, 4.8, 7.0, 50.0}
```

Dari contoh di atas jelas bahwa dengan mengakses *index* dari sebuah *array* dapat dilakukan manipulasi elemen di dalamnya. Perlu diperhatikan bahwa angka di dalam kurung siku pada baris pertama dan dua baris lainnya merupakan hal yang berbeda. Pada baris pertama, yang merupakan inisialisasi, merupakan ukuran dari *array*, sedangkan pada baris kedua dan ketiga merupakan *index* dari elemen yang ingin diakses. Anda dapat menggunakan konsep perulangan untuk mengakses *array* secara lebih mudah ketimbang menuliskan *index*-nya secara manual.

Referensi

[1] https://www.tutorialspoint.com/cprogramming/c_arrays.htm

III. Tutorial

Di bawah ini tersedia dua buah kode tutorial yang terkait dengan materi pada modul ini. Silakan Anda menyalin kode berikut dan menjalankannya pada komputer. Cermati proses eksekusi program sambil membaca dan memahami komentar serta maksud dari kode yang diberikan.

Source Code 3 Kode Tutorial 1 Modul 1

```
/** EL2208 Praktikum Pemecahan Masalah dengan C 2021/2022
 *  Modul               : 1 - Overview of C Language
 *  Percobaan           : Tutorial
 *  Hari dan Tanggal    : Jumat, 11 Februari 2022
 *  Pembuat             : Irfan Tito Kurniawan
 *  Nama File           : tutorial-m01-01.c
 *  Deskripsi           : Source code tutorial untuk modul 01.
 */

/** Perintah Kompilasi
 *
 * Linux/MacOS  - gcc -o tutorial-m01-01 tutorial-m01-01.c -lm
 * Windows      - gcc -o tutorial-m01-01.exe tutorial-m01-01.c
 *
 * atau dengan Makefile:
 *
 * Linux/MacOS  - make 01
 */

/** Catatan
 *
 * Sebuah kode tidak seharusnya memiliki komentar yang terlalu
 * banyak seperti kode ini. Kode ini sengaja dibuat dengan
 * komentar yang rinci untuk membantu pemahaman.
 */
```

```c
// Preprocessor Directives
// Files inclusion
#include <stdio.h>
#include <math.h>

// Macro definition
// Bilangan Euler
#define EULER 2.71828

int main() {
    // Deklarasi variabel
    int i;
    float f;
    char c;

    /** Blok input nilai variabel
     *
     * Catatan:
     * Saat input, gunakan tanda ampersand (&) di depan nama variabel.
     * Hal ini ditujukan untuk merujuk alamat memori dari variabel,
     * alih-alih nilai variabel.
     */
    printf("-- Fase Masukan\n");
    printf("Masukkan nilai bilangan bulat i: ");
    scanf("%d", &i);
    // Input char
    /** Trik:
     *  Tambahkan spasi di depan placeholder untuk mengabaikan karakter
     *  newline dari input sebelumnya.
     */
    printf("Masukkan nilai karakter c: ");
    scanf(" %c", &c);
    printf("Masukkan nilai bilangan real f: ");
    scanf("%f", &f);

    /** Blok echo nilai variabel
     *
     * Print nilai variabel yang telah di-input.
     */
    printf("\n-- Fase Keluaran\n");
    printf("Nilai bilangan bulat i: %d\n", i);
    printf("Nilai karakter c: %c\n", c);
    // Print float dengan formatting 3 angka di belakang koma
    printf("Nilai bilangan real f: %.3f\n", f);

    /** Blok assignment dan ekspresi
     *
     */
    // Contoh ekspresi perkalian
    i = i * 2;
    printf("Nilai variabel i setelah dikali 2: %d\n", i);
    // Contoh assignment
    i = 5;
    printf("Nilai variabel i setelah assignment: %d\n", i);
    // Contoh perbandingan
    /** Karena C tidak memiliki variabel boolean, boolean dapat diwakili
oleh int
     *  dengan false dinyatakan sebagai 0, dan true dengan bukan 0.
     *
```

```
     *  Bisa dicoba sendiri dengan kondisional:
     *      if (5) {...}, if (-2) {...}, atau if (0) {...}
     */
    printf("Apakah nilai i kurang dari 5: %d\n", i < 5);
    printf("Apakah nilai i sama dengan 5: %d\n", i == 5);

    /** Blok perhitungan
     *
     * Print nilai eksponen dengan beberapa metode.
     */
    printf("\n-- Contoh Perhitungan\n");
    printf("Nilai bilangan Euler: %.3f\n", EULER);
    // Print nilai eksponen i dengan fungsi eksponensial exp()
    printf("Dengan math.h - Hasil dari e^(%d): %.3f\n", i, exp(i));
    // Print nilai eksponen i dengan fungsi pangkat pow()
    printf("Dengan macro - Hasil dari e^(%d): %.3f\n", i, pow(EULER, i));

    // Return 0 mengindikasikan tidak ada error yang terjadi
    return 0;
}
```

Source Code 4 Kode Tutorial 2 Modul 1

```
/** EL2208 Praktikum Pemecahan Masalah dengan C 2020/2021
 *  Modul               : 1 - Overview of C Language
 *  Percobaan           : Tutorial
 *  Hari dan Tanggal    : Sabtu, 13 Februari 2021
 *  Pembuat             : Irfan Tito Kurniawan
 *  Nama File           : tutorial-01-02.c
 *  Deskripsi           : Source code tutorial untuk modul 01.
 */

/** Perintah Kompilasi
 *
 * Linux/MacOS  - gcc -o tutorial-m01-02 tutorial-m01-02.c
 * Windows      - gcc -o tutorial-m01-02.exe tutorial-m01-02.c
 *
 * atau dengan Makefile:
 *
 * Linux/MacOS  - make 02
 */

/** Catatan
 *
 * Sebuah kode tidak seharusnya memiliki komentar yang terlalu
 * banyak seperti kode ini. Kode ini sengaja dibuat dengan
 * komentar yang rinci untuk membantu pemahaman.
 */

#include <stdio.h>
#define MAX_ENTRY 20

int main () {
    // Deklarasi array of int
    int arr[MAX_ENTRY];
    int entry_amt;

    // Input jumlah masukan
```

```
    printf("Input jumlah masukan (maksimum 20): ");
    scanf("%d", &entry_amt);

    /** Struktur kontrol kondisional.
     *
     * Pada kasus ini digunakan untuk validasi input.
     */
    if (entry_amt <= 0 || entry_amt > 20)
        printf("Error, jumlah masukan salah.\n");
    else {
        /** Struktur kontrol iteratif.
         *
         * Pada kasus ini digunakan untuk input dan output elemen array.
         */

        // Input elemen array
        printf("\nMasukkan %d bilangan bulat:\n", entry_amt);

        for (int i = 0; i < entry_amt; ++i)
            scanf("%d", &arr[i]);

        // Print elemen array
        printf("\nIsi array: ");
        for (int i = 0; i < entry_amt; ++i)
            printf("%d ", arr[i]);

        /** Struktur kontrol kondisional.
         *
         * Pada kasus ini digunakan untuk mencari elemen dari array.
         * Digunakan karena pencarian tidak perlu dilanjutkan ketika
         * bilangan sudah ditemukan.
         */

        // Input bilangan nyang ingin dicari dalam array
        int key;

        printf("\n\nInput bilangan yang ingin dicari dalam array: ");
        scanf("%d", &key);

        // Cari bilangan dalam array
        int idx = 0;
        int found = 0;

        // Cari selama indeks masih di dalam batas array dan bilangan
        // belum ditemukan
        while (idx < entry_amt && !found) {
            // Elemen ditemukan
            if (arr[idx] == key)
                found = 1;
            else
                // Iterasi
                ++idx;
        }

        // Cetak hasil pencarian
        if (found)
            printf("Bilangan %d ditemukan pada indeks %d.\n", key, idx);
        else
            printf("Bilangan %d tidak ada.", key);
```

```
    }

    return 0;
}
```

MODUL II
FILE EXTERNAL

I. Tujuan

1. Melatih kemampuan untuk melakukan dekomposisi masalah
2. Memberikan pemahaman terkait tipe data *string* dan pemrosesannya pada Bahasa C
3. Memberikan pemahaman terkait pengolahan *file* eksternal pada Bahasa C

II. Materi

II.I. String

String dalam Bahasa C merupakan sebuah *array* dengan tipe data *char*, artinya *string* merupakan kumpulan dari beberapa karakter. *String* dalam Bahasa C akan selalu diakhiri dengan sebuah *null* karakter '\0'. Untuk mendeklarasikan sebuah *string* dapat digunakan cara yang sama seperti saat mendeklarasikan *array*. Berikut diberikan dua contoh cara mendeklarasikan *string* dalam Bahasa C.

```
char greeting[6] = {'H', 'e', 'l', 'l', 'o', '\0'};

char greeting[] = "Hello";
```

Anda dapat mendeklarasikan ukuran *string* yang akan digunakan kemudian memasukkan karakter yang sesuai satu per satu. Selain itu, Anda juga bisa menggunakan cara tanpa memberikan ukuran *string* lalu langsung menuliskan kalimat yang diperlukan dan biarkan *compiler* yang mengatur ukurannya. Karakter *null* pada bagian akhir contoh baris satu sesungguhnya tidak perlu ditulis secara eksplisit karena *compiler* secara otomatis akan menambahkannya.

Pada umumnya untuk melakukan operasi pada *string* diperlukan sebuah *library* tambahan untuk mempermudah yaitu `string.h`. Dalam *library* tersebut terdapat berbagai macam fungsi yang dapat digunakan untuk memanipulasi *string*. Beberapa fungsi yang biasa digunakan ditampilkan pada tabel berikut.

Tabel 9 Contoh Fungsi dalam Library String.h

No.	Fungsi	Deskripsi
1	`char *strtok(char *str, const char *delim)`	Digunakan untuk memisahkan sebuah *string* menjadi beberapa buah *string* yang dipisahkan dengan *delimiter* tertentu.
2	`char *strcpy(char *dest, const char *src)`	Digunakan untuk menyalin sebuah *string*.
3	`int strcmp(const char *str1, const char *str2)`	Digunakan untuk membandingkan dua buah *string* apakah sama atau tidak.
4	`size_t strlen(const char *str)`	Digunakan untuk menghitung panjang dari suatu *string* tanpa melibatkan *null* karakter.
5	`char *strcat(char *dest, const char *src)`	Digunakan untuk menambahkan suatu *string* ke akhir dari *string* lainnya.

Tabel di atas memberikan beberapa contoh fungsi yang umum digunakan. Anda dapat membaca lebih lanjut fungsi-fungsi lainnya yang terdapat pada *library* `string.h` pada referensi di bawah. Pada referensi tersebut juga tersedia contoh penggunaan dari masing-masing fungsi.

Referensi

[1] https://www.tutorialspoint.com/cprogramming/c_strings.htm

[2] https://www.tutorialspoint.com/c_standard_library/string_h.htm

II.II. File Eksternal

Dalam Bahasa C tersedia fitur untuk mengolah sebuah *file* eksternal. Proses pengolahan meliputi membuka *file*, menulis *file*, membaca *file*, dan menutup *file*. Masing-masing operasi tersebut akan dibahas lebih detail pada bagian berikutnya.

1. Membuka File

   Proses membuka suatu *file* eksternal dapat dilakukan dengan menggunakan fungsi `fopen()`. Fungsi tersebut akan menerima parameter berupa nama *file* dan mode yang digunakan. Untuk lebih jelasnya berikut adalah *syntax* yang digunakan.

```
FILE *fopen( const char * filename, const char * mode );
```

   Dalam Bahasa C terdapat beberapa mode pembukaan yang dapat digunakan. Penjelasan untuk masing-masing mode ditampilkan pada tabel berikut.

Tabel 10 Mode Pembacaan File Eksternal dalam Bahasa C

No.	Mode	Deskripsi
1.	r	Membuka *file* yang sudah ada untuk dibaca isinya.
2.	w	Membuka *file* untuk dituliskan isinya. Jika *file* belum ada maka akan dibuat sebuah *file* baru.
3.	a	Membuka *file* untuk dituliskan isinya secara *appending*. Jika *file* belum ada maka akan dibuat *file* baru.
4.	r+	Membuka *file* untuk membaca dan menulis isinya.
5.	w+	Membuka *file* untuk membaca dan menulis isinya. Jika *file* sudah ada maka *file* akan dipotong hingga panjangnya nol. Jika *file* tidak ada maka akan dibuat sebuah *file* baru.
6.	a+	Membuka *file* untuk membaca dan menulis isinya. Jika *file* tidak ada maka akan dibuat sebuah *file* baru. Proses membaca bisa dilakukan dari awal *file*, namun proses menulis hanya bisa dilakukan secara *appending*.

2. Menulis File

   Proses menulis *file* dapat dilakukan menggunakan dua cara yaitu satu persatu tiap karakter atau secara langsung dalam bentuk *string*. *Syntax* untuk masing-masing metode tersebut dapat dilihat pada tabel berikut.

```
int fputc( int c, FILE *fp );
```

```
int fputs( const char *s, FILE *fp );
```

Kedua metode tersebut akan menerima dua buah parameter yaitu data yang akan ditulis dalam bentuk karakter atau *string* dan *file* yang akan digunakan untuk menulis.

3. Membaca File

   Mirip seperti proses menulis, pada proses membaca *file* juga terdapat dua buah metode yang dapat digunakan yaitu membaca karakter dan membaca satu baris *string*. *Syntax* yang digunakan untuk melakukan pembacaan *file* dapat dilihat pada tabel berikut.

```
int fgetc( FILE * fp );

char *fgets( char *buf, int n, FILE *fp );
```

Pembacaan *file* berdasarkan karakter, *syntax* baris pertama, hanya memerlukan sebuah parameter yaitu *file* yang akan dibaca. Berbeda dengan metode pembacaan karakter, pembacaan berdasarkan *string* memerlukan tiga buah parameter yaitu variabel untuk menyimpan *string* yang dibaca, panjang *string* yang bisa dibaca, dan nama *file* yang dibaca.

4. Menutup File

   Proses menutup suatu *file* dapat dilakukan dengan mudah menggunakan fungsi `fclose()`. Fungsi ini akan memiliki *syntax* sebagai berikut.

```
int fclose( FILE *fp );
```

Fungsi tersebut akan memberikan nilai **EOF** jika terjadi eror saat menutup sebuah *file*.

Referensi

[1] https://www.tutorialspoint.com/cprogramming/c_file_io.htm

III. Tutorial

Di bawah ini tersedia kode tutorial yang terkait dengan materi pada modul ini. Silakan Anda menyalin kode berikut dan menjalankannya pada komputer. Cermati proses eksekusi program sambil membaca dan memahami komentar serta maksud dari kode yang diberikan.

Source Code 5 Kode Tutorial Modul 2

```
/** EL2208 Praktikum Pemecahan Masalah dengan C 2020/2021
 *  Modul               : 2 - Strings and External Files
 *  Percobaan           : Tutorial
 *  Hari dan Tanggal    : Jumat, 18 Februari 2022
 *  Nama File           : tutorial-02.c
 *  Pembuat             : Tito Irfan
 *  Deskripsi           : Source code tutorial untuk modul 02.
 */

/** Perintah Kompilasi
 *
 * Linux/MacOS  - gcc -o tutorial-02 tutorial-02.c
 * Windows      - gcc -o tutorial-02.exe tutorial-02.c
 *
 * atau dengan Makefile:
 *
 * Linux/MacOS  - make 01
 */

#include <stdio.h>
#include <stdlib.h>
#include <string.h>

// Panjang maksimum entry
#define MAX_LEN 255

int main () {
    /** Print 'string' yang konstan.
     * Catatan: lebihkan ukuran variabel karena string
     *  diakhiri dengan karakter terminator '\0'.
     *
     * Misal: 'halo' membutuhkan ukuran variabel 5 karakter,
     *  karena program menyimpan 'halo' + '\0'.
     */
    const char dummy[7] = "Henlo!";

    printf("Test print string: %s\n", dummy);

    // Input nama file
    char filename[MAX_LEN];
    printf("\nMasukkan nama file: ");
    scanf("%s", &filename);

    // Buka file untuk dibaca
    FILE* stream = fopen(filename, "r");

    // Array mama mahasiswa
    char names[MAX_LEN][MAX_LEN];
    // Array nilai mahasiswa
    float score[MAX_LEN];
```

```
    // Variabel penyimpan baris dari file sementara
    char line[MAX_LEN];

    // Token penyimpan string sementara untuk parsing
    char* token;

    // Iterator penanda indeks array,
    // pada akhir iterasi nilai i akan sama dengan jumlah baris pada file
    int i = 0;

    // Baca file baris demi baris hingga habis
    while(fgets(line, MAX_LEN, stream)) {
        // Parse baris
        // Ambil string pada baris hingga tanda koma (koma tidak ikut
dibaca)
        token = strtok(line, ",");

        // Copy string dari token ke array nama
        // String pada C tidak bisa di-assign dengan operator =, gunakan
strcpy
        strcpy(names[i], token);

        // Ambil string pada baris hingga karakter newline (newline tidak
ikut dibaca)
        token = strtok(NULL, "\n");
        // Convert string yang dibaca menjadi float dengan atof
        // Agar nilai yang dibaca bisa diperlakukan seperti bilangan
        score[i] = atof(token);

        // Iterasi
        ++i;
    }

    // Cari nilai maksimum, minimum, dan rata-rata nilai
    float max = score[0];
    float min = score[0];
    float sum = score[0];

    // Index nilai maksimum dan minimum pada array
    int max_idx = 0;
    int min_idx = 0;

    for (int j = 1; j < i; ++j) {
        // Cari nilai maksimum dan minimum
        if (score[j] > max) {
            max = score[j];
            max_idx = j;
        }
        if (score[j] < min) {
            min = score[j];
            min_idx = j;
        }

        // Jumlahkan nilai untuk menghitung rata-rata
        sum += score[j];
    }

    // Nilai rata-rata
```

```
    float mean = sum / i;

    // Print semua nama dan nilai
    for (int j = 0; j < i; ++j)
        printf("%s memperoleh nilai %.2f\n", names[j], score[j]);

    // Print statistik
    printf("\nStatistik:\n");
    printf("Nilai tertinggi: %.2f, atas nama %s\n", max, names[max_idx]);
    printf("Nilai terendah: %.2f, atas nama %s\n", min, names[min_idx]);
    printf("Nilai rata-rata: %.2f\n", mean);

    // Nama sebagai kunci pencarian
    char key[MAX_LEN];

    // Input kunci pencarian
    printf("\nCari nilai atas nama: ");
    scanf("%s", &key);

    // Hitung panjang nama dengan strlen dan tampilkan sebagai long int
    printf("%s terdiri atas %ld karakter.\n", key, strlen(key));

    // Cari nama hingga ditemukan atau hingga array habis
    int found = 0;
    int j = 0;

    while (j < i && !found) {
        // Perbandingan string, akan bernilai 0 bila kedua string sama
persis
        // maka dari itu, false berarti cocok.
        // String tidak bisa dikomparasi dengan operator == atau !=,
gunakan strcmp
        if (!strcmp(key, names[j])) {
            found = 1;
        }
        else
            ++j;
    }

    // Print nilai kunci pencarian bila ditemukan
    if (found)
        printf("%s memperoleh nilai: %.2f\n", key, score[j]);
    else
        printf("%s tidak ada.\n", key);

    // Tutup file
    fclose(stream);

    return 0;
}
```

Tabel 11 File Eksternal Tutorial Modul 2

```
Andi,70.5
Budi,40.5
Caca,20.9
Dedi,80.9
Eko,95.4
```

```
Fakhri,100.0
```

MODUL III
POINTERS

I. Tujuan

1. Melatih kemampuan untuk melakukan dekomposisi masalah
2. Memberikan pemahaman penggunaan *pointer* pada Bahasa C
3. Memberikan pemahaman terkait penggunaan fungsi pada Bahasa C

II. Materi

II.I. Pointer

Seperti yang sudah dibahas pada bagian sebelumnya, variabel merupakan sebuah alokasi memori yang punya alamat dan data yang disimpan di dalamnya. Alamat memori dari sebuah variabel dapat diakses menggunakan `&`. *Pointer* merupakan variabel yang menyimpan alamat memori dari variabel lainnya. Sama seperti variabel pada umumnya, *pointer* perlu dideklarasikan terlebih dahulu sebelum digunakan. *Syntax* yang digunakan untuk mendeklarasikan *pointer* adalah sebagai berikut,

```
type *var-name;
```

Bagian `type` merupakan tipe data dari variabel yang akan kita simpan alamat memorinya sedangkan `var-name` merupakan nama variabel *pointer* yang digunakan. Berikut diberikan beberapa contoh deklarasi *pointer*.

```
int    *ip;    /* pointer to an integer */
double *dp;    /* pointer to a double */
float  *fp;    /* pointer to a float */
char   *ch     /* pointer to a character */
```

Setelah mengetahui proses deklarasi dari *pointer* maka selanjutnya akan dibahas cara menggunakan *pointer*. Terdapat tiga buah tahap yang diperlukan dalam menggunakan *pointer* yaitu mendeklarasikan *pointer*, menyimpan alamat dari suatu variabel pada *pointer*, dan mengakses nilai dari alamat yang disimpan pada *pointer*. Proses menyimpan alamat memori dapat menggunakan operator `&` sedangkan untuk mengakses nilai dari alamat memori pada sebuah *pointer* dapat menggunakan operator `*`. Berikut diberikan contoh penggunaan operator tersebut.

```
/*assume address of var is 0x08bfacdf*/
int  var = 20;   /* actual variable declaration */
int  *ip;        /* pointer variable declaration */

ip = &var;  /* store address of var in pointer variable*/


printf("Address of var variable: %p\n", &var  );

/* address stored in pointer variable */
printf("Address stored in ip variable: %p\n", ip );

/* access the value using the pointer */
printf("Value of *ip variable: %d\n", *ip );
```

Asumsikan variabel `var` tersimpan pada alamat memori di atas, maka pada baris ke tiga alamat memori dari `var` akan disimpan pada *pointer* `ip`. Setelah alamat memori disimpan pada *pointer* maka selanjutnya *pointer* tersebut dapat digunakan untuk operasi. Sebagai contoh dilakukan operasi untuk mencetak pada layar, dua perintah cetak pertama akan menghasilkan nilai yang sama yaitu alamat memori dari `var` yakni `0x08bfacdf` karena pada baris sebelumnya sudah dilakukan proses penyimpanan alamat memori pada *pointer*. Perintah cetak pada baris terakhir akan menghasilkan nilai yang disimpan oleh variabel `var` yaitu 20. Perhatikan ketika akan mengakses nilai yang disimpan pada variabel maka digunakan operator `*`.

Selain penjelasan di atas, terdapat aplikasi dari *pointer* yang lebih dalam. Penjelasan lebih lanjut terkait *pointer* dapat dibaca pada referensi yang diberikan. Silakan lakukan eksplorasi secara mandiri.

Referensi

[1] https://www.tutorialspoint.com/cprogramming/c_pointers.htm

II.II. Fungsi

Fungsi merupakan sekelompok perintah yang memiliki tugas spesifik dalam bagian program. Fungsi berguna agar program lebih rapi dan mempermudah penulisan karena fungsi dapat digunakan secara terus menerus sesuai kebutuhan tanpa perlu menulis ulang.

Dalam membuat fungsi perlu diperhatikan bahwa terdapat deklarasi dan definisi fungsi. Deklarasi fungsi memuat informasi yaitu nama fungsi, tipe, dan parameter untuk digunakan oleh *compiler*. Definisi fungsi memuat keseluruhan fungsi termasuk perintah-perintah dalam fungsi tersebut. Secara umum fungsi dapat didefinisikan menggunakan format berikut

```
return_type function_name( parameter list ) {
   body of the function
}
```

dengan `return_type` merupakan tipe data yang dikembalikan oleh fungsi tersebut, `function_name` merupakan nama fungsi, parameter merupakan bagian yang digunakan untuk meneruskan nilai yang akan diproses oleh fungsi, dan *body* merupakan bagian fungsi yang berisi baris perintah.

Selain menggunakan definisi umum seperti atas, fungsi juga dapat dibuat dengan menggunakan deklarasi dan definisi secara terpisah. Deklarasi fungsi secara umum akan menggunakan format berikut

```
return_type function_name( parameter list );
```

Setelah membuat deklarasi selanjutnya dapat dilakukan definisi fungsi. Sebagai contoh diberikan fungsi untuk mengambil bilangan terbesar dari dua buah bilangan.

```
#include <stdio.h>
```

```c
/* function declaration */
int max(int num1, int num2);

int main () {

   /* local variable definition */
   int a = 100;
   int b = 200;
   int ret;

   /* calling a function to get max value */
   ret = max(a, b);

   printf( "Max value is : %d\n", ret );

   return 0;
}

/* function definiton */
/* function returning the max between two numbers */
int max(int num1, int num2) {

   /* local variable declaration */
   int result;

   if (num1 > num2)
      result = num1;
   else
      result = num2;

   return result;
}
```

Perhatikan bahwa contoh di atas memiliki bagian deklarasi dan bagian definisi. Bagian deklarasi akan diletakkan pada bagian atas program sedangkan definisi diletakkan setelah bagian `main`. Selain metode tersebut, fungsi dapat dibuat dengan menuliskan definisi secara langsung pada bagian deklarasinya.

Referensi

[1] https://www.tutorialspoint.com/cprogramming/c_functions.htm

III. Tutorial

Di bawah ini tersedia kode tutorial yang terkait dengan materi pada modul ini. Silakan Anda menyalin kode berikut dan menjalankannya pada komputer. Cermati proses eksekusi program sambil membaca dan memahami komentar serta maksud dari kode yang diberikan.

Source Code 6 Kode Tutorial Modul 3

```
/** EL2208 Praktikum Pemecahan Masalah dengan C 2021/2022
 *  Modul               : 3 - Pointers and Functions
 *  Percobaan           : Tutorial
 *  Hari dan Tanggal    : Selasa, 3 Januari 2023
 *  Nama File           : tutorial-03.c
 *  Pembuat             : Dismas Widyanto
 *  Deskripsi           : Source code tutorial untuk modul 03.
 */

/** Perintah Kompilasi
 *
 * Linux/MacOS  - gcc -o tutorial-03 tutorial-03.c
 * Windows      - gcc -o tutorial-03.exe tutorial-03.c
 *
 * atau dengan Makefile:
 *
 * Linux/MacOS  - make 01
 */

#include <stdio.h>
#include <stdlib.h>

/**
* Fungsi dengan definisi umum
*/
int add_int(int operand_1, int operand_2){
      return operand_1 + operand_2;
}

/**
* Fungsi dengan deklarasi
*/
void say_hi();

int main(){
      /**
      *Bagian ini menjelaskan pointer
      */

      //Deklarasi variabel
      int number = 24;
      int arr[5] = {10,20,30,40,50};
      int *ptr_to_number;

      //Simpan alamat variabel number pada pointer
      ptr_to_number = &number;

      //Cetak nilai alamat dan value
      printf("Nilai pada variabel number: %d\n",number);
      printf("Alamat pada variabel number: %p\n",&number);
```

```
	printf("Nilai pada pointer ptr_to_number: %p\n",ptr_to_number);
	printf("Nilai yang dirujuk pointer ptr_to_number:
%d\n",*ptr_to_number);
	printf("\n");

	//Pointer dan array
	//Memanggil nama array saja (tanpa index)
	//akan merujuk ke alamat pertama dari array tersebut
	printf("Alamat pertama dari sebuah array: %p\n",arr);
	printf("Elemen pertama dari sebuah array: %d\n",arr[0]);

	//Elemen dari array dapat dipanggil dengan
	//menggunakan metode yang sama dengan memanggil nilai yang dirujuk
pointer
	printf("Elemen pertama dari sebuah array: %d\n",*arr);
	printf("Elemen kedua dari sebuah array: %d\n",*(arr+1));
	printf("Elemen ketiga dari sebuah array: %d\n",*(arr+2));
	printf("Elemen keempat dari sebuah array: %d\n",*(arr+3));
	printf("Elemen kelima dari sebuah array: %d\n",*(arr+4));
	printf("\n");

	/**
	*Bagian ini menjelaskan function
	*/
	//Functions tipe Void
	say_hi();
	printf("\n");

	//Deklarasi Variabel
	int a = 31;
	int b = 14;

	//Functions tipe Int
	printf("Hasil penjumlahan dari variabel: %d
(31+14)\n",add_int(a,b));
	printf("Hasil penjumlahan dari operand langsung: %d
(12+8)\n",add_int(12,8));


	return 0;
}

/**
* Definisi dari fungsi sebelumnya
*/
void say_hi(){
	printf("Hello, Konichiwa, Bonjour\n");
}
```

MODUL IV
STRUCTURES AND DYNAMIC ARRAYS

I. Tujuan

1. Melatih kemampuan untuk melakukan dekomposisi masalah
2. Memberikan pemahaman pada praktikan terkait tipe data bentukan pada Bahasa C
3. Memberikan pemahaman pada praktikan terkait penggunaan *array* dinamis pada Bahasa C

II. Materi

II.I. Array Dinamis

Memory management atau sering disebut sebagai *array* dinamis merupakan metode dalam Bahasa C yang digunakan untuk mengatur ukuran dari sebuah *array*. Ukuran *array* pada umumnya ditulis secara eksplisit pada kode program dan tidak dapat diubah selama program berjalan. *Array* dinamis memberikan kemudahan untuk kondisi ketika ukuran *array* yang akan digunakan belum diketahui secara pasti. *Array* dinamis memungkinkan perubahan ukuran *array* ketika program sedang dijalankan.

Array dinamis dapat dijalankan dengan menggunakan fungsi yang terdapat dalam *library* `stdlib.h`. Terdapat empat buah fungsi yang secara umum digunakan untuk mengelola *array* dinamis. Masing-masing fungsi dijelaskan pada tabel berikut.

Tabel 12 Fungsi Pengolahan Array Dinamis dalam Bahasa C

No.	Fungsi	Deskripsi
1.	`void *calloc(int num, int size)`	Fungsi ini akan membuat *array* dengan ukuran sesuai parameternya. *Array* yang dibuat akan memiliki nilai awal yaitu 0.
2.	`void free(void *address)`	Fungsi ini digunakan untuk menghapus blok memori / *array* dengan alamat tertentu.
3.	`void *malloc(size_t size)`	Fungsi ini akan membuat *array* baru dengan ukuran tertentu.

4.	`void *realloc(void *address, int newsize)`	Fungsi ini digunakan untuk mengubah ukuran *array* yang sudah dibuat sebelumnya.

Referensi

[1] https://www.tutorialspoint.com/cprogramming/c_memory_management.htm

[2] https://www.tutorialspoint.com/c_standard_library/stdlib_h.htm

II.II. Structure

Sebelumnya sudah diajarkan bahwa *array* dapat menyimpan banyak data dengan tipe yang sama. Mirip dengan *array*, *structure* juga dapat menyimpan banyak data hanya saja tipe data yang dapat disimpan dapat berbeda-beda. *Structure* bekerja dengan cara mendefinisikan tipe data baru yang berisikan tipe data dasar. Untuk mendefinisikan *structure* dapat digunakan *syntax* berikut.

```
struct [structure tag] {

   member definition;
   member definition;
   ...
   member definition;
} [one or more structure variables];
```

Dengan `structure tag` merupakan nama dari tipe data baru yang dibentuk, `member definition` merupakan deklarasi variabel, dan `structure variable` merupakan nama variabel yang dibentuk dengan tipe data bentukan tersebut (opsional). Sebagai contoh, diberikan *structure* untuk mendefinisikan tipe data buku.

```
struct Books {
   char  title[50];
   char  author[50];
   char  subject[100];
   int   book_id;
} book;
```

Contoh di atas dapat diartikan bahwa terdapat tipe data bentukan baru yang bernama `Books` dengan komponen penyusun yaitu `title`, `author`, `subject`, dan `book_id`. Terdapat juga sebuah variabel dengan tipe bentukan yang bernama `book`. Perhatikan bahwa komponen penyusun dari tipe data bentukan dapat berupa *array* maupun tipe data dasar.

Dari contoh di atas terlihat bahwa sudah terbentuk sebuah variabel bernama `book` yang memiliki tipe data `Books`. Untuk mendeklarasikan tipe data bentukan dapat dilakukan seperti variabel biasa yakni mengikuti *syntax* berikut.

```
struct [structure tag] nama_variabel


//contoh
struct Books Book1;
```

Komponen dari sebuah *structure* dapat diakses masing-masing menggunakan tanda titik (`.`). Menggunakan contoh di atas komponen dari tipe data `Books` dapat diakses menggunakan *syntax* berikut.

```
strcpy( book.title, "C Programming");
strcpy( book.author, "Nuha Ali");
strcpy( book.subject, "C Programming Tutorial");
book.book_id = 6495407;


//atau
strcpy( Book1.title, "C Programming");
strcpy( Book1.author, "Nuha Ali");
strcpy( Book1.subject, "C Programming Tutorial");
Book1.book_id = 6495407;
```

Perhatikan bahwa diperlukan nama dari variabel yang digunakan yaitu `book` atau `Book1` kemudian diikuti dengan nama komponen yang ingin diakses.

Sebuah *structure* dapat diperlakukan seperti variabel pada umumnya. *Structure* dapat dijadikan sebuah *pointer* dan bisa dijadikan sebagai parameter sebuah fungsi. Penggunaan *structure* sebagai sebuah *pointer* tetap menggunakan tanda (*) untuk deklarasi. Untuk mengakses elemennya menggunakan tanda (->). Perhatikan bahwa untuk mengakses elemen *structure* biasa dengan *structure pointer* memiliki cara yang berbeda. Perhatikan contoh berikut untuk melakukan deklarasi dan mengakses *structure pointer*.

```
struct Books *struct_pointer;
struct_pointer = &Book1;

printf( "Book title : %s\n", book->title);
printf( "Book author : %s\n", book->author);
printf( "Book subject : %s\n", book->subject);
printf( "Book book_id : %d\n", book->book_id);
```

Referensi

[1] https://www.tutorialspoint.com/cprogramming/c_structures.htm

III. Tutorial

Di bawah ini tersedia dua buah kode tutorial yang terkait dengan materi pada modul ini. Silakan Anda menyalin kode berikut dan menjalankannya pada komputer. Cermati proses eksekusi program sambil membaca dan memahami komentar serta maksud dari kode yang diberikan.

Source Code 7 Kode Tutorial 1 Modul 4

```
/** EL2208 Praktikum Pemecahan Masalah dengan C 2020/2021
 *  Modul               : 4 - Structures and Dynamic Arrays
 *  Percobaan           : Tutorial
 *  Hari dan Tanggal    : Jumat, 12 Maret 2021
 *  Nama File           : tutorial-04-arr.c
 *  Pembuat             : Tito Irfan
 *  Deskripsi           : Source code tutorial untuk modul 04, materi
array dinamis.
 */

/** Perintah Kompilasi
 *
 * Linux/MacOS  - gcc -o tutorial-04-arr tutorial-04-arr.c
 * Windows      - gcc -o tutorial-04-arr.exe tutorial-04-arr.c
 *
 * atau dengan Makefile:
```

```
 *
 * Linux/MacOS - make arr
 */

#include <stdio.h>
// stdlib dibutuhkan untuk manipulasi memori (i.e. malloc, calloc, etc)
#include <stdlib.h>

// Jumlahkan isi array dinamis arr dengan ukuran arr_size.
int sum_arr(int* arr, int arr_size) {
    int sum = 0;

    for (int i = 0; i < arr_size; ++i)
        sum += arr[i];

    return sum;
}

int main() {
    // Deklarasi array dinamis
    // Pointer to int yang akan menyimpan address elemen pertama array
    int* arr = NULL;
    // Ukuran array dinamis sejauh ini
    int arr_size = 0;
    // Variabel penyimpan input sementara sebelum disimpan dalam array
    int temp;

    // Input array secara dinamis hingga input -1
    printf("Input elemen array (-1 untuk selesai): ");
    scanf("%d", &temp);

    // Loop selama input tidak sama dengan -1
    while (temp != -1) {
        // Catat penambahan ukuran array
        ++arr_size;

        // Alokasi memori tambahan untuk menampung input
        if (arr_size == 1)
            // Bila array belum diinisialisasi
            // Reservasi memori sebesar 1 buah integer dengan malloc
            // malloc me-return pointer ke alamat memori yang telah
direservasi
            arr = (int*) malloc(sizeof(int));
        else
            // Bila array sudah diinisialisasi
            // Alokasi ulang dengan ukuran sebesar arr_size buah integer
dengan realloc
            // realloc me-return pointer ke alamat memori yang telah
direservasi
            arr = (int*) realloc(arr, arr_size * sizeof(int));

        // Assign nilai elemen terakhir array dengan input
        arr[arr_size - 1] = temp;

        // Alternatif: bagaimana kalau assignment diganti (*arr +
arr_size - 1) = temp; ?

        // Input elemen berikutnya
        printf("Input elemen array (-1 untuk selesai): ");
```

```
        scanf("%d", &temp);
    }

    // Print seluruh isi array
    printf("\nIsi array:\n");
    for (int i = 0; i < arr_size; ++i)
        printf("%d\n", arr[i]);

    // Print jumlah isi array arr
    printf("Jumlah isi array: %d\n", sum_arr(arr, arr_size));

    // Reset isi array
    // Free memori yang digunakan
    free(arr);
    // Set ukuran array ke nol kembali
    arr_size = 0;

    // Input ukuran array
    printf("\nJumlah elemen array: ");
    scanf("%d", &arr_size);

    // Alokasi memori dan inisialisasi array sebesar arr_size buah
integer
    arr = (int*) calloc(arr_size, sizeof(int));

    // Print seluruh isi array sebelum input
    printf("\nIsi array sebelum input:\n");
    for (int i = 0; i < arr_size; ++i)
        printf("%d\n", arr[i]);

    // Input isi array
    printf("\nInput isi array:\n");
    for (int i = 0; i < arr_size; ++i) {
        printf("Input elemen ke-%d: ", i + 1);
        scanf("%d", &arr[i]);
    }

    // Print seluruh isi array
    printf("\nIsi array setelah input:\n");
    for (int i = 0; i < arr_size; ++i)
        printf("%d\n", arr[i]);

    // Free memori yang digunakan
    free(arr);

    return 0;
}
```

Source Code 8 Kode Tutorial 2 Modul 4

```
/** EL2208 Praktikum Pemecahan Masalah dengan C 2020/2021
 *  Modul               : 4 - Structures and Dynamic Arrays
 *  Percobaan           : Tutorial
 *  Hari dan Tanggal    : Jumat, 12 Maret 2021
 *  Nama File           : tutorial-04-struct.c
 *  Pembuat             : Tito Irfan
 *  Deskripsi           : Source code tutorial untuk modul 04, materi
struct.
```

```
 */

/** Perintah Kompilasi
 *
 * Linux/MacOS  - gcc -o tutorial-04-struct tutorial-04-struct.c
 * Windows      - gcc -o tutorial-04-struct.exe tutorial-04-struct.c
 *
 * atau dengan Makefile:
 *
 * Linux/MacOS - make struct
 */

#include <stdio.h>
#include <stdlib.h>

/** Struct Mahasiswa.
 *
 * Berisi NIM dan IPK mahasiswa.
 */
typedef struct Mahasiswa {
    char nim[10];
    float ipk;
} Mahasiswa;

// Bandingkan IPK dua mahasiswa, return pointer ke mahasiswa dengan IPK
lebih tinggi.
// Bila IPK sama, return NULL pointer.
Mahasiswa* sarip(Mahasiswa* mhs_1, Mahasiswa* mhs_2) {
    // Solusi, inisialisasi dengan NULL
    Mahasiswa* bang_jago = NULL;

    // Bandingkan IPK
    if (mhs_1->ipk > mhs_2->ipk)
        bang_jago = mhs_1;
    else if (mhs_1->ipk < mhs_2->ipk)
        bang_jago = mhs_2;

    return bang_jago;
}

// Bucin akan menyita waktu dan membuat IPK turun :(, kurangi IPK mhs
menjadi setengahnya.
void bucin(Mahasiswa* mhs) {
    mhs->ipk = mhs->ipk / 2;
}

int main() {
    // Deklarasi variabel dengan tipe Mahasiswa dan pointer to Mahasiswa
    Mahasiswa mhs, *mhs_pt;

    // Alokasi memori untuk pointer to Mahasiswa
    mhs_pt = (Mahasiswa*) malloc(sizeof(Mahasiswa));

    // Input data kedua mahasiswa
    // Perhatikan bahwa untuk mengakses member dari struct dapat
digunakan operator dot (.),
    // dan untuk mengakses member dari pointer to struct dapat digunakan
operator arrow (->).
    printf("Input NIM Mahasiswa 1: ");
```

```
    scanf("%s", &(mhs.nim));
    printf("Input IPK Mahasiswa 1: ");
    scanf("%f", &(mhs.ipk));

    printf("Input NIM Mahasiswa 2: ");
    scanf("%s", &(mhs_pt->nim));
    printf("Input IPK Mahasiswa 2: ");
    scanf("%f", &(mhs_pt->ipk));

    // Print data mahasiswa yang telah diinput
    printf("\nMahasiswa 1 - NIM: %s, IPK: %.2f\n", mhs.nim, mhs.ipk);
    printf("Mahasiswa 2 - NIM: %s, IPK: %.2f\n", mhs_pt->nim, mhs_pt-
>ipk);

    // Cari mahasiswa dengan IPK lebih tinggi
    Mahasiswa* bang_jago = sarip(&mhs, mhs_pt);
    if (bang_jago != NULL)
        printf("\nBang jago - NIM: %s\n", bang_jago->nim);
    else
        printf("\nTiada bang jago di antara kita.\n");

    // Bucin itu tidak baik.
    bucin(&mhs);
    printf("\nSetelah bucin, IPK Mahasiswa 1 anjlok menjadi %.2f.
Bersama, kita bisa hentikan ini.\n", mhs.ipk);

    return 0;
}
```

MODUL V
RECURSIONS

I. Tujuan

1. Melatih kemampuan untuk melakukan dekomposisi masalah
2. Memberikan pemahaman pada praktikan terkait fungsi *recursive* yang dibuat dengan Bahasa C

II. Materi

Fungsi refursif merupakan fungsi yang memanggil dirinya sendiri di dalam prosesnya. Fungsi yang sudah dipelajari sebelumnya hanya memiliki proses yang sesuai dengan kebutuhan fungsi namun tidak memanggil dirinya sendiri. Fungsi rekursif dapat digunakan untuk menyelesaikan berbagai persoalan yang mudah maupun kompleks. Secara umum fungsi rekursif dapat dibentuk dengan *syntax* berikut.

```
void recursion() {
   recursion(); /* function calls itself */
}

int main() {
   recursion();
}
```

Dalam menggunakan fungsi rekursif perlu diperhatikan dua hal penting yaitu kasus basis dan kasus rekursi agar tidak terjadi *infinite loop*. Kasus basis merupakan kasus paling dasar yang dapat menghentikan proses rekursi. Sedangkan kasus rekursi merupakan kondisi ketika program masih akan melakukan proses rekursi. Untuk memahami lebih lanjut silakan pelajari contoh dari referensi yang diberikan dan tutorial yang diberikan di bawah ini.

Referensi

[1] https://www.tutorialspoint.com/cprogramming/c_recursion.htm

III. Tutorial

Di bawah ini tersedia kode tutorial yang terkait dengan materi pada modul ini. Silakan Anda menyalin kode berikut dan menjalankannya pada komputer. Cermati proses eksekusi program sambil membaca dan memahami komentar serta maksud dari kode yang diberikan.

```
/** EL2208 Praktikum Pemecahan Masalah dengan C 2020/2021
 *  Modul               : 5 - Recursion
 *  Percobaan           : Tutorial
 *  Hari dan Tanggal    : Jumat, 19 Maret 2021
 *  Nama File           : tutorial-05.c
 *  Pembuat             : Tito Irfan
 *  Deskripsi           : Source code tutorial untuk modul 05, materi
rekursi.
 */

/** Perintah Kompilasi
 *
 * Linux/MacOS  - gcc -o tutorial-05 tutorial-05.c
 * Windows      - gcc -o tutorial-05.exe tutorial-05.c
 *
 * atau dengan Makefile:
 *
 * Linux/MacOS - make all
 */

#include <stdio.h>
#include <string.h>
#include <math.h>

/** recursive_power. Hitung pangkat bilangan bulat secara rekursif.
 *
 * @param[in] num Bilangan bulat yang akan dipangkatkan.
 * @param[in] pow Bilangan bulat positif pangkat.
 * @return Hasil dari num pangkat pow.
 */
int recursive_power(int num, int pow);

/** recursive_print. Print isi array secara rekursif.
 *
 * @param[in] arr Array of integer yang akan dicetak, diberikan sebagai
pointer.
 * @param[in] size Bilangan bulat ukuran arr.
 * @param[in] cur_idx Indeks yang sedang dicetak saat ini.
 * @result Seluruh isi arr tercetak ke layar.
 */
void recursive_print(int *arr, int size, int cur_idx);

/** check_palindrome. Cek apakah suatu string merupakan palindrom.
 *
 * @param[in] s String yang akan diperiksa sifat palindromnya.
 * @return Apakah s merupakan palindrom.
 */
int check_palindrome(char *s);

/** check_palindrome_helper. Fungsi pembantu rekursi check_palindrome.
 *
```

```
 * @param[in] s String yang akan diperiska sifat palindromnya.
 * @param[in] idx_front Bilangan bulat indeks karakter depan yang sedang
diperiksa.
 * @param[in] idx_rear Bilangan bulat indeks karakter belakang yang
sedang diperiksa.
 * @return Apakah s merupakan palindrom.
 */
int check_palindrome_helper(char *s, int idx_front, int idx_rear);

int main() {
    // Bagian 1 - Hitung pangkat bilangan bulat
    // Ubah dua variabel berikut untuk mencoba-coba.
    // num - bilangan bulat yang dipangkatkan
    int num = 2;
    // pow - bilangan bulat positif pangkatnya
    int pow = 10;

    printf("\nHitung pangkat: \n");
    printf("Hasil dari %d pangkat %d adalah: %d\n", num, pow,
recursive_power(num, pow));

    // Bagian 2 - Print array secara rekursif
    // Ubah arr untuk mencoba-coba.
    int arr[] = {0, 1, 2, 3, 4, 5};

    printf("\nPrint rekursif: \n");
    recursive_print(arr, sizeof(arr) / sizeof(arr[0]), 0);

    // Bagian 3 - Periksa palindrom
    // Ubah s untuk mencoba-coba.
    char* s = "kasurrusak";

    printf("\nCek palindrom: \n");
    if (check_palindrome(s))
        printf("'%s' adalah palindrome.\n", s);
    else
        printf("'%s' bukan palindrom.\n", s);

    return 0;
}

int recursive_power(int num, int pow) {
    // Kasus basis, apabila pangkat sudah sama dengan nol
    if (pow == 0)
        return 1;
    else
        // Rekuren, kalikan num dengan num^(pow -1)
        return num * recursive_power(num, pow - 1);
}

void recursive_print(int *arr, int size, int cur_idx) {
    // Rekuren, print elemen sekarang dan selanjutnya
    if (cur_idx < size) {
        printf("%d ", arr[cur_idx]);

        recursive_print(arr, size, cur_idx + 1);
    }
    else
        // Kasus basis, ketika cur_idx == size, print newline
```

```
        printf("\n");
}

int check_palindrome(char *s) {
    // Fungsi ini dibuat hanya agar kita bisa memanggil fungsi
    // periksa palindrom tanpa memberikan indeks-indeks.

    // Panggil fungsi check_palindrome_helper dengan idx_front 0 dan
    // idx_rear elemen paling belakang string.
    return check_palindrome_helper(s, 0, strlen(s) - 1);
}

int check_palindrome_helper(char *s, int idx_front, int idx_rear) {
    // Kasus basis bila panjang ganjil
    if (idx_front == idx_rear)
        return 1;

    // Kasus basis bila panjang genap
    if (idx_front == idx_rear - 1 && s[idx_front] == s[idx_rear])
        return 1;

    // Rekuren, bila belum sampai ke tengah, cek bagian tengahnya
    if (s[idx_front] != s[idx_rear])
        // Bila ditemukan pasangan karakter yang berbeda, langsung
        // nyatakan bahwa s bukan palindrom.
        return 0;
    else
        // Bila masih sama, cek pasangan karakter selanjutnya (ke tengah)
        return check_palindrome_helper(s, idx_front + 1, idx_rear - 1);
}
```

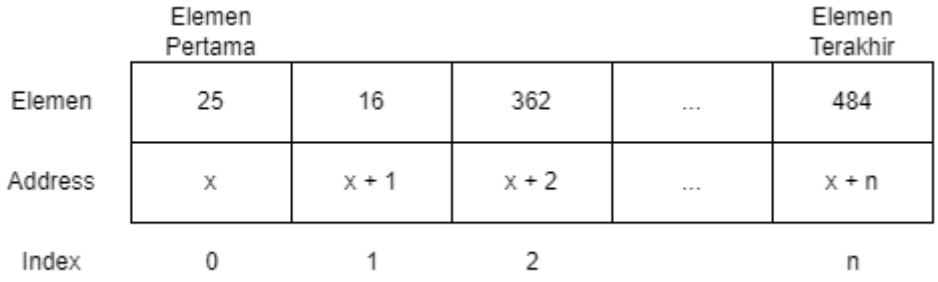
MODUL VI
LINKED LIST

I. Tujuan

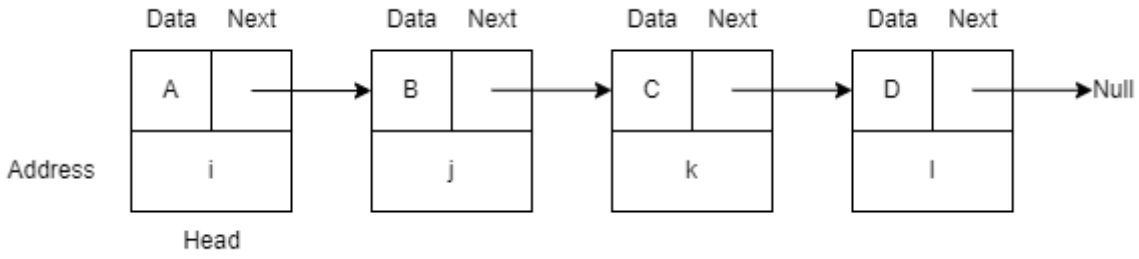
1. Melatih kemampuan untuk melakukan dekomposisi masalah
2. Memberikan pemahaman terkait struktur data *linked list* dalam Bahasa C

II. Materi

Linked list merupakan salah satu struktur data yang mampu menyimpan banyak data. Struktur data ini secara sekilas mirip dengan *array* namun terdapat perbedaan terkait bagaimana data disimpan dalam memori. Sebuah *array* akan menyimpan data pada memori dengan alamat yang berurutan sedangkan *linked list* akan menyimpan data pada alamat memori yang acak. Ilustrasi untuk kedua struktur data tersebut dapat dilihat pada Gambar 17 dan Gambar 18.



Gambar 17 Ilustrasi Array dengan Alamat Memori



Gambar 18 Ilustrasi Linked List dengan Alamat Memori

Perhatikan dari ilustrasi di atas terlihat bahwa struktur data *array* akan menyimpan masing-masing datanya secara urut pada alamat memori. Sebaliknya, struktur data *linked list* akan menyimpan data pada alamat memori yang acak namun tetap terkoneksi. Masing-masing blok data pada *linked list* sering disebut sebagai `node`. Sebuah `node` merupakan tipe data bentukan yang terdiri dari data dan *pointer* ke `node` berikutnya. `Node` pertama atau yang paling atas biasa disebut juga sebagai `head`.

Linked list memiliki beberapa tipe berdasarkan ikatan antar `node`. Dalam modul ini akan dijelaskan tiga jenis yang umum dijumpai yaitu *singly linked list*, *doubly linked list*, dan *circular linked list*. Penjelasan untuk masing-masing jenis *linked list* dapat dilihat pada poin berikut.

1. *Singly Linked List*

   Singly linked list merupakan tipe *linked list* yang hubungan antar `node` hanya berlaku satu arah dari `head` hingga akhir. Gambar 18 merupakan tipe *singly linked list*. *Linked list* tipe ini paling umum digunakan.

2. *Doubly Linked List*

   Doubly linked list merupakan tipe *linked list* yang mempunyai dua jalur hubungan `node`. Tipe *linked list* ini dapat diakses secara maju dari `head` hingga akhir ataupun sebaliknya.

3. *Circular Linked List*

   Circular linked list merupakan tipe *linked list* yang `node` akhirnya memiliki jalur untuk kembali ke `node head` sehingga akses *linked list* tipe ini bisa diibaratkan seperti sebuah lingkaran.

Silakan pelajari lebih lanjut terkait *linked list* dengan mengakses referensi di bawah. Referensi [2] menjelaskan terkait operasi dasar yang sering dilakukan pada *linked list*. Pelajari juga tutorial yang diberikan pada bagian selanjutnya.

Referensi:

[1] https://www.tutorialspoint.com/data_structures_algorithms/linked_list_algorithms.htm

[2] https://www.tutorialspoint.com/dsa_using_c/dsa_using_c_linked_list.htm

III. Tutorial

Di bawah ini tersedia kode tutorial dan dua buah kode pendukung yang terkait dengan materi pada modul ini. Silakan Anda menyalin kode berikut dan menjalankannya pada komputer. Cermati proses eksekusi program sambil membaca dan memahami komentar serta maksud dari kode yang diberikan.

Source Code 9 Kode Header Tutorial Modul 6

```
/** EL2208 Praktikum Pemecahan Masalah dengan C 2020/2021
 *  Modul               : 6 - Linked List
 *  Percobaan           : Tutorial
 *  Hari dan Tanggal    : Selasa, 23 Maret 2021
 *  Nama File           : lib.h
 *  Pembuat             : Tito Irfan
 *  Deskripsi           : Header file untuk library linked list.
 */

#include <stdio.h>
#include <stdlib.h>

/** Node. Sebuah node dari singly-linked list.
 *
 * @param data data int yang disimpan dalam node.
 * @param next pointer menuju node berikutnya.
 */
typedef struct Node {
    int data;
    struct Node* next;
} Node;

/** printLinkedList. Print seluruh isi linked list.
 *
 * @param current_node pointer menuju node dimulainya print.
 * @result seluruh data yang tersimpan dalam linked list tercetak.
 */
void printLinkedList(Node* current_node);

/** printLinkedListRecursive. Print seluruh isi linked list tapi
rekursif.
 *
 * @param current_node pointer menuju node dimulainya print.
 * @result seluruh data yang tersimpan dalam linked list tercetak.
 */
void printLinkedListRecursive(Node* current_node);

/** push. Tambahkan node berisi new_data di depan linked list.
 *
 * @param head pointer to pointer to node head dari linked list.
 * @param new_data data yang akan disimpan dalam node baru.
 * @result head menyimpan pointer menuju node paling depan baru yang
```

```
berisi new_data.
 */
void push(Node** head, int new_data);

/** insert. Tambahkan node berisi new_data setelah prev_node.
 *
 * @param prev_node pointer menuju node pendahulu node baru.
 * @param new_data data yang akan disimpan dalam node baru.
 * @result node berisi new_data ditambahkan setelah prev_node.
 */
void insert(Node* prev_node, int new_data);

/** deleteNode. Hapus node yang ditunjuk del_node dari linked list.
 *
 * @param head pointer to pointer to node head dari linked list.
 * @param del_node pointer menuju node yang akan dihapus dari linked
list.
 * @result del_node terhapus dari linked list.
 */
void deleteNode(Node** head, Node* del_node);

/** deleteKey. Hapus node pertama yang menyimpan data dengan value key
dari linked list.
 *
 * @param head pointer to pointer to node head dari linked list.
 * @param key data int dari elemen yang ingin dihapus dari linked list.
 * @result elemen pertama yang menyimpan key terhapus dari linked list.
 */
void deleteKey(Node** head, int key);

/** destroy. Hapus seluruh isi linked list.
 *
 * @param head pointer to pointer to node head dari linked list.
 * @result memori seluruh isi linked list dibersihkan dan *head berisi
NULL.
 */
void destroy(Node** head);

/** destroyHelper. Pembantu rekursi untuk destroy.
 *
 * @param cur_node node yang sedang dihapus saat ini.
 * @result memori cur_node dan seluruh node setelahnya dibersihkan.
 */
void destroyHelper(Node* cur_node);
```

Source Code 10 Kode Definisi Fungsi Tutorial Modul 6

```
/** EL2208 Praktikum Pemecahan Masalah dengan C 2020/2021
 *  Modul               : 6 - Linked List
 *  Percobaan           : Tutorial
 *  Hari dan Tanggal    : Selasa, 23 Maret 2021
 *  Nama File           : lib.c
 *  Pembuat             : Tito Irfan
 *  Deskripsi           : Source file untuk library linked list.
 */

#include "lib.h"
```

```
void printLinkedList(Node* current_node) {
    while (current_node != NULL) {
        printf("%d ", current_node->data);
        current_node = current_node->next;
    }
    printf("\n");
}

void printLinkedListRecursive(Node* current_node) {
    if (current_node != NULL) {
        printf("%d ", current_node->data);
        printLinkedList(current_node->next);
    }
}

void push(Node** head, int new_data) {
    Node* new_node = (Node*) malloc(sizeof(Node));
    new_node->data = new_data;
    new_node->next = *head;
    *head = new_node;
}

void insert(Node* prev_node, int new_data) {
    if (prev_node != NULL) {
        Node* new_node = (Node*) malloc(sizeof(Node));
        new_node->data = new_data;
        new_node->next = prev_node->next;
        prev_node->next = new_node;
    }
}

void deleteNode(Node** head, Node* del_node) {
    Node* cur_node = *head;
    Node* prev_node;

    if (del_node == *head) {
        *head = (*head)->next;
        free(cur_node);
    }
    else {
        while (cur_node != del_node && cur_node != NULL) {
            prev_node = cur_node;
            cur_node = cur_node->next;
        }
        if (cur_node != NULL) {
            prev_node->next = cur_node->next;
            free(cur_node);
        }
    }
}

void deleteKey(Node** head, int key) {
    Node* cur_node = *head;
    Node* prev_node;

    if ((*head)->data == key) {
        *head = (*head)->next;
        free(cur_node);
    }
```

```
    else {
        while (cur_node->data != key && cur_node != NULL) {
            prev_node = cur_node;
            cur_node = cur_node->next;
        }
        if (cur_node != NULL) {
            prev_node->next = cur_node->next;
            free(cur_node);
        }
    }
}

void destroyHelper(Node* cur_node) {
    if (cur_node != NULL) {
        destroyHelper(cur_node->next);
        free(cur_node);
    }
}

void destroy(Node** head) {
    destroyHelper(*head);
    *head = NULL;
}
```

Source Code 11 Kode Tutorial Modul 6

```
/** EL2208 Praktikum Pemecahan Masalah dengan C 2020/2021
 *  Modul               : 6 - Linked List
 *  Percobaan           : Tutorial
 *  Hari dan Tanggal    : Selasa, 23 Maret 2021
 *  Nama File           : linked-list.c
 *  Pembuat             : Tito Irfan
 *  Deskripsi           : Main file untuk tutorial modul 06.
 */

/** Perintah Kompilasi
 *
 * Linux/MacOS  - gcc -o linked-list linked-list.c lib.c
 * Windows      - gcc -o linked-list.exe linked-list.c lib.c
 *
 * atau dengan Makefile:
 *
 * Linux/MacOS - make
 */

#include <stdio.h>
#include <stdlib.h>
#include "lib.h"

int main() {
    // Tambahkan tiga node pertama linked list
    Node* head = (Node*) malloc(sizeof(Node));
    Node* second_node = (Node*) malloc(sizeof(Node));
    Node* third_node = (Node*) malloc(sizeof(Node));

    head->data = 9;
    second_node->data = 6;
    third_node->data = 7;
```

```
    head->next = second_node;
    second_node->next = third_node;
    third_node->next = NULL;

    // Print isi linked list, tapi manual
    // Perhatikan bahwa seluruh node bisa diakses hanya dengan
    // mengetahui address dari node head
    printf("%d %d %d \n", head->data, head->next->data, head->next->next-
>data);

    // Print isi linked list saat ini
    printLinkedList(head);

    // Tambahkan node berisi 420 di depan linked list
    push(&head, 420);
    printLinkedList(head);

    // Tambahkan node berisi 1337 di depan linked list
    insert(second_node, 1337);
    printLinkedList(head);

    // Hapus second_node
    deleteNode(&head, second_node);
    printLinkedList(head);

    // Hapus node pertama yang berisi 9
    deleteKey(&head, 9);
    printLinkedList(head);

    // Hapus node pertama yang berisi 420
    deleteKey(&head, 420);
    printLinkedList(head);

    // Bebaskan memori (ekuivalen dengan free)
    destroy(&head);

    return 0;
}
```

MODUL VII
STACKS AND QUEUES

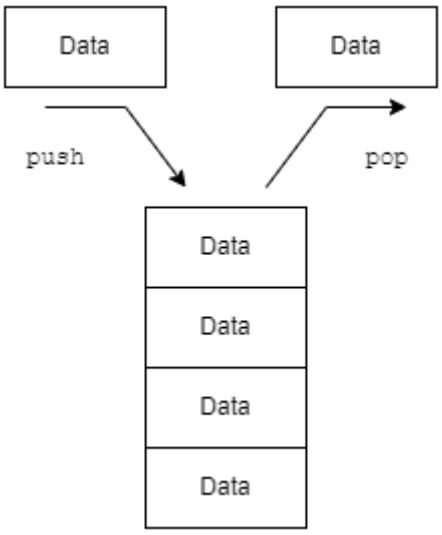
I. Tujuan

1. Melatih kemampuan untuk melakukan dekomposisi masalah
2. Memberikan pemahaman terkait struktur data *stack* dan *queue*
3. Memberikan pemahaman terkait algoritma pencarian dan pengurutan untuk struktur data *linked structure*

II. Materi

II.I. Stack

Stack adalah salah satu tipe data abstrak yang sering digunakan pada berbagai Bahasa pemrograman. *Stack* akan menyimpan data dalam sebuah "tumpukan" yang hanya dapat diakses dari satu buah sisi saja. *Stack* dapat diibaratkan seperti tumpukan buku dalam kardus. Buku tersebut hanya dapat diambil dari tumpukan teratas dan hanya dapat dimasukkan buku baru pada bagian teratas. Tipe data yang seperti ini masuk kategori LIFO atau *last in first out*.



Gambar 19 Ilustrasi Stack

Stack dapat diimplementasikan menggunakan *array*, *structure*, maupun *linked list*. Dalam modul ini akan digunakan implementasi *stack* dengan *linked list*. Terdapat beberapa operasi dasar yang sering digunakan dalam mengolah *stack* yaitu `push`, `pop`, `peek`, dan `isEmpty`. Masing-masing penjelasan dari operasi dasar diberikan pada bagian berikut.

1. `Push`

   Operasi ini digunakan untuk menambahkan data baru ke dalam *stack* yang sudah ada.

2. `Pop`

   Operasi ini digunakan untuk menghapus atau mengeluarkan data paling atas dari sebuah *stack*.

3. `Peek`

   Operasi ini digunakan untuk mendapatkan nilai dari data paling atas dalam sebuah *stack*.

4. `isEmpty`

   Operasi ini digunakan untuk melakukan pengecekan apakah suatu *stack* kosong.

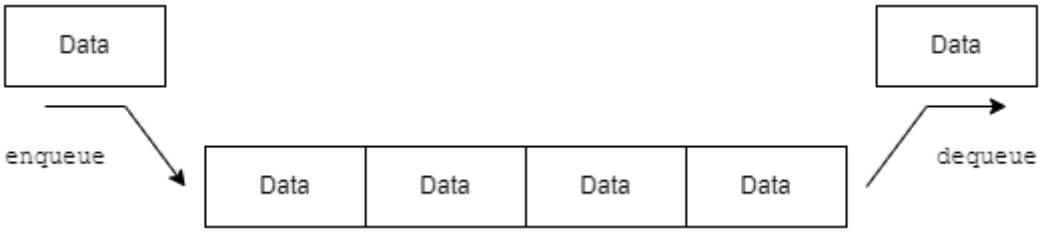
Silakan pelajari lebih lanjut terkait *stack* pada referensi yang diberikan. Operasi dasar *stack* yang diberikan dapat dipelajari lebih lanjut pada tutorial yang tersedia di bagian selanjutnya.

Referensi:

[1] https://www.tutorialspoint.com/data_structures_algorithms/stack_algorithm.htm

II.II. Queue

Queue merupakan salah satu jenis tipe data abstrak seperti *stack* yang sering dijumpai juga pada berbagai Bahasa pemrograman. Berbeda dengan *stack* yang berbentuk “tumpukan”, *queue* akan menyimpan datanya dalam bentuk “antrean”. *Queue* memiliki dua buah sisi yang dapat diakses yaitu `front` dan `rear`. Masing-masing sisi tersebut digunakan untuk jalur keluar dan masuknya data. Sebuah *queue* dapat diibaratkan sebagai antrean di kasir. Antrean tersebut akan memiliki dua buah sisi yaitu di dekat kasir untuk jalur keluar dan di sisi pelanggan untuk menerima antrean. Metode seperti ini sering disebut sebagai FIFO atau *first in first out*.



Gambar 20 Ilustrasi Queue

Seperti *stack*, *queue* dapat diimplementasikan menggunakan *array*, *structure*, dan *linked list*. Dalam pembahasan modul ini akan digunakan implementasi *queue* dengan *linked list*. *Queue* memiliki beberapa operasi dasar yang sering digunakan yaitu `enqueue`, `dequeue`, `peek` dan `isEmpty`. Penjelasan untuk masing-masing operasi dapat dilihat pada poin berikut.

1. `Enqueue`

   Operasi ini digunakan untuk menambahkan data baru ke dalam *queue* yang sudah ada.

2. `Dequeue`

   Operasi ini digunakan untuk menghapus atau mengeluarkan data dari sebuah *queue*.

3. `Peek`

   Operasi ini digunakan untuk mendapatkan nilai dari data paling atas dalam sebuah *queue*.

4. `isEmpty`

   Operasi ini digunakan untuk melakukan pengecekan apakah suatu *queue* kosong.

Silakan pelajari lebih lanjut terkait *queue* pada referensi yang diberikan. Operasi dasar *queue* yang diberikan dapat dipelajari lebih lanjut pada tutorial yang tersedia di bagian selanjutnya.

Referensi:

[1] https://www.tutorialspoint.com/data_structures_algorithms/dsa_queue.htm

III. Tutorial

Di bawah ini tersedia kode tutorial yang terkait dengan materi pada modul ini. Kode tutorial ini terdiri dari tujuh buah *file*. Silakan Anda menyalin kode berikut dan menjalankannya pada komputer. Cermati proses eksekusi program sambil membaca dan memahami komentar serta maksud dari kode yang diberikan.

Source Code 12 Kode Header Tutorial Modul 7

```
/** EL2208 Praktikum Pemecahan Masalah dengan C 2020/2021
 *  Modul               : 7 - Stacks and Queues
 *  Percobaan           : Tutorial
 *  Hari dan Tanggal    : Jumat, 2 April 2021
 *  Nama File           : common.h
 *  Pembuat             : Tito Irfan
 *  Deskripsi           : Common header file untuk library stack dan
queue.
 */

/** Node. Sebuah node dari singly-linked list.
 * Digunakan sebagai elemen dari Stack dan Queue.
 *
 * @param data data int yang disimpan dalam node.
 * @param next pointer menuju node berikutnya.
 */
typedef struct Node {
    int data;
    struct Node* next;
} Node;
```

Source Code 13 Kode Header Stack Tutorial Modul 7

```
/** EL2208 Praktikum Pemecahan Masalah dengan C 2020/2021
 *  Modul               : 7 - Stacks and Queues
 *  Percobaan           : Tutorial
 *  Hari dan Tanggal    : Jumat, 2 April 2021
 *  Nama File           : stack_lib.h
 *  Pembuat             : Tito Irfan
 *  Deskripsi           : Header file untuk library stack.
 */

#include <stdio.h>
#include <stdlib.h>
#include "common.h"

/** Stack. Representasi dari sebuah stack.
 *
 * @param head pointer menuju node paling atas dalam stack.
 */
typedef struct Stack {
    Node* head;
} Stack;

/** push. Tambahkan new_data ke atas stack.
 *
 * @param stack pointer menuju stack.
 * @param new_data data baru untuk ditambahkan ke dalam stack.
 */
void push(Stack* stack, int new_data);

/** pop. Hapus elemen paling atas dari stack, ambil datanya.
 *
 * @param stack pointer menuju stack.
 * @return stack->head->data bila ada, 0 bila tidak ada.
 */
```

```
int pop(Stack* stack);

/** peek. Lihat data elemen paling atas dari stack.
 *
 * @param stack pointer menuju stack.
 * @return stack->head->data bila ada, 0 bila tidak ada.
 */
int peek(Stack* stack);

/** isStackEmpty. Cek apakah stack kosong.
 *
 * @param stack pointer menuju stack.
 * @return stack->head == NULL.
 */
int isStackEmpty(Stack* stack);

void printStack(Stack* stack);
```

Source Code 14 Kode Definisi Fungsi Stack Tutorial Modul 7

```
/** EL2208 Praktikum Pemecahan Masalah dengan C 2020/2021
 *  Modul               : 7 - Stacks and Queues
 *  Percobaan           : Tutorial
 *  Hari dan Tanggal    : Jumat, 2 April 2021
 *  Nama File           : stack_lib.c
 *  Pembuat             : Tito Irfan
 *  Deskripsi           : Implementation file untuk library stack.
 */

#include "stack_lib.h"

int isStackEmpty(Stack* stack) {
    return stack->head == NULL;
}

void push(Stack* stack, int new_data) {
    Node* new_node = (Node*) malloc(sizeof(Node));
    new_node->data = new_data;
    new_node->next = stack->head;

    stack->head = new_node;
}

int pop(Stack* stack) {
    int ret_val = 0;
    Node* temp = stack->head;

    if (temp != NULL) {
        ret_val = temp->data;
        stack->head = temp->next;

        free(temp);
    }

    return(ret_val);
}

int peek(Stack* stack) {
```

```
    if (stack->head != NULL)
        return stack->head->data;
    else
        return 0;
}

void printStack(Stack* stack) {
    Node* current = stack->head;

    while (current != NULL) {
        printf("%d ", current->data);
        current = current->next;
    }
    printf("\n");
}
```

Source Code 15 Kode Tutorial Stack Modul 7

```
/** EL2208 Praktikum Pemecahan Masalah dengan C 2020/2021
 *  Modul               : 7 - Stacks and Queues
 *  Percobaan           : Tutorial
 *  Hari dan Tanggal    : Jumat, 2 April 2021
 *  Nama File           : tutorial-07-stack.c
 *  Pembuat             : Tito Irfan
 *  Deskripsi           : Source code tutorial untuk modul 07, materi
stack.
 */

/** Perintah Kompilasi
 *
 * Linux/MacOS  - gcc -o tutorial-07-stack tutorial-07-stack.c
stack_lib.c
 * Windows      - gcc -o tutorial-07-stack.exe tutorial-07-stack.c
stack_lib.c
 *
 * atau dengan Makefile:
 *
 * Linux/MacOS - make stack
 */

#include <stdio.h>
#include <stdlib.h>
#include "stack_lib.h"

int main() {

    // Catatan:
    //  pop tidak harus return 0 ketika stack
    //  kosong. Pada contoh ini dibuat demikian
    //  agar lebih mudah dipahami.
    //  anda bisa mengganti 0 dengan INT_MIN misalnya.

    Stack* stack = (Stack*) malloc(sizeof(Stack));
    stack->head = NULL;

    push(stack, 1);
    push(stack, 2);
    push(stack, 3);
```

```
    printf("Current stack content:\n");
    printStack(stack);

    printf("\nPop, got: %d\n", pop(stack));
    printf("Front of stack: %d\n", peek(stack));
    printf("Current stack content:\n");
    printStack(stack);

    printf("\nPop, got: %d\n", pop(stack));
    printf("Front of stack: %d\n", peek(stack));
    printf("Current stack content:\n");
    printStack(stack);

    printf("\nPop, got: %d\n", pop(stack));
    printf("Front of stack: %d\n", peek(stack));
    printf("Current stack content:\n");
    printStack(stack);

    free(stack);

    return 0;
}
```

Source Code 16 Kode Header Queue Tutorial Modul 7

```
/** EL2208 Praktikum Pemecahan Masalah dengan C 2020/2021
 *  Modul               : 7 - Stacks and Queues
 *  Percobaan           : Tutorial
 *  Hari dan Tanggal    : Jumat, 2 April 2021
 *  Nama File           : queue_lib.h
 *  Pembuat             : Tito Irfan
 *  Deskripsi           : Header file untuk library queue.
 */

#include <stdio.h>
#include <stdlib.h>
#include "common.h"

/** Queue. Representasi dari sebuah Queue.
 *
 * @param head pointer menuju node paling depan dalam queue.
 * @param tail pointer menuju node paling belakang dalam queue.
 */
typedef struct Queue {
    Node* head;
    Node* tail;
} Queue;

/** enqueue. Tambahkan new_data ke belakang queue.
 *
 * @param queue pointer menuju queue.
 * @param new_data data baru untuk ditambahkan ke dalam queue.
 */
void enqueue(Queue* queue, int new_data);

/** Dequeue. Hapus elemen paling depan dari queue, ambil datanya.
 *
```

```
 * @param queue pointer menuju queue.
 * @return queue->head->data bila ada, 0 bila tidak.
 */
int dequeue(Queue* queue);

/** isQueueEmpty. Cek apakah queue kosong.
 *
 * @param queue pointer menuju queue.
 * @return queue->head == NULL.
 */
int isQueueEmpty(Queue* queue);

/** front. Ambil data elemen paling depan dari queue.
 *
 * @param queue pointer menuju queue.
 * @return queue->head->data bila ada, 0 bila tidak.
 */
int front(Queue* queue);

/** rear. Ambil data elemen paling belakang dari queue.
 *
 * @param queue pointer menuju queue.
 * @return queue->tail->data bila ada, 0 bila tidak.
 */
int rear(Queue* queue);

void printQueue(Queue* queue);
```

Source Code 17 Kode Definisi Fungsi Queue Tutorial Modul 7

```
/** EL2208 Praktikum Pemecahan Masalah dengan C 2020/2021
 *  Modul               : 7 - Stacks and Queues
 *  Percobaan           : Tutorial
 *  Hari dan Tanggal    : Jumat, 2 April 2021
 *  Nama File           : queue_lib.c
 *  Pembuat             : Tito Irfan
 *  Deskripsi           : Implementation file untuk library queue.
 */

#include "queue_lib.h"

int isQueueEmpty(Queue* queue) {
    return queue->head == NULL;
}

void enqueue(Queue* queue, int new_data) {
    Node* new_node = (Node*) malloc(sizeof(Node));
    new_node->data = new_data;
    new_node->next = NULL;

    if (!isQueueEmpty(queue)) {
        queue->tail->next = new_node;
        queue->tail = new_node;
    }
    else {
        queue->tail = new_node;
        queue->head = new_node;
    }
```

```
}

int dequeue(Queue* queue) {
    int ret_val = 0;
    Node* temp = queue->head;

    if (temp != NULL) {
        ret_val = temp->data;
        queue->head = temp->next;

        if (temp == queue->tail)
            queue->tail = NULL;

        free(temp);
    }

    return ret_val;
}

int front(Queue* queue) {
    if (!isQueueEmpty(queue))
        return queue->head->data;
    else
        return 0;
}

int rear(Queue* queue) {
    if (!isQueueEmpty(queue))
        return queue->tail->data;
    else
        return 0;
}

void printQueue(Queue* queue) {
    Node* current = queue->head;

    while (current != NULL) {
        printf("%d ", current->data);
        current = current->next;
    }
    printf("\n");
}
```

Source Code 18 Kode Tutorial Queue Modul 7

```
/** EL2208 Praktikum Pemecahan Masalah dengan C 2020/2021
 *  Modul               : 7 - Stacks and Queues
 *  Percobaan           : Tutorial
 *  Hari dan Tanggal    : Jumat, 2 April 2021
 *  Nama File           : tutorial-07-queue.c
 *  Pembuat             : Tito Irfan
 *  Deskripsi           : Source code tutorial untuk modul 07, materi
queue.
 */

/** Perintah Kompilasi
 *
 * Linux/MacOS  - gcc -o tutorial-07-queue tutorial-07-queue.c
```

```
queue_lib.c
 * Windows      - gcc -o tutorial-07-queue.exe tutorial-07-queue.c
queue_lib.c
 *
 * atau dengan Makefile:
 *
 * Linux/MacOS - make queue
 */

#include <stdio.h>
#include <stdlib.h>
#include "queue_lib.h"

int main() {

    // Catatan:
    //  dequeue tidak harus return 0 ketika queue
    //  kosong. Pada contoh ini dibuat demikian
    //  agar lebih mudah dipahami.
    //  anda bisa mengganti 0 dengan INT_MIN misalnya.

    Queue* queue = (Queue*) malloc(sizeof(Queue));
    queue->head = NULL;
    queue->tail = NULL;

    enqueue(queue, 1);
    enqueue(queue, 2);
    enqueue(queue, 3);

    printf("Front of queue: %d\n", front(queue));
    printf("Rear of queue: %d\n", rear(queue));
    printf("Current queue content:\n");
    printQueue(queue);

    printf("\nDequeue, got: %d\n", dequeue(queue));
    printf("Front of queue: %d\n", front(queue));
    printf("Rear of queue: %d\n", rear(queue));
    printf("Queue content post-dequeue:\n");
    printQueue(queue);

    printf("\nDequeue, got: %d\n", dequeue(queue));
    printf("Front of queue: %d\n", front(queue));
    printf("Rear of queue: %d\n", rear(queue));
    printf("Queue content post-dequeue:\n");
    printQueue(queue);

    printf("\nDequeue, got: %d\n", dequeue(queue));
    printf("Front of queue: %d\n", front(queue));
    printf("Rear of queue: %d\n", rear(queue));
    printf("Queue content post-dequeue:\n");
    printQueue(queue);

    free(queue);

    return 0;
}
```

MODUL VIII
ALGORITHM

I. Tujuan

1. Melatih kemampuan untuk melakukan dekomposisi masalah
2. Memberikan pemahaman terkait strategi pembuatan algoritma untuk menyelesaikan masalah
3. Memberikan pemahaman terkait struktur data *graph* dan *tree* beserta algoritma umumnya

II. Materi

II.I. Strategi Algoritma

Algoritma merupakan prosedur sistematis yang digunakan untuk memecahkan suatu masalah. Dalam bahasan ini masalah yang diberikan akan diselesaikan menggunakan Bahasa C. Terdapat berbagai macam algoritma yang dapat digunakan untuk memecahkan berbagai masalah, namun pembahasan ini akan difokuskan pada strategi dasar yang mudah dipahami. Strategi dasar yang pertama adalah *backtracking* dan yang kedua adalah *divide and conquer*. Masing-masing strategi akan dibahas lebih lanjut pada poin berikut.

1. Backtracking

   Backtracking adalah salah satu strategi untuk memecahkan masalah dengan cara mencoba berbagai kombinasi yang menjanjikan memberikan solusi dengan cara rekursif. Cara ini merupakan cara yang paling mudah untuk dipahami dan digunakan, namun cara ini memiliki kekurangan yaitu kurang efektif karena harus melakukan pengecekan berulang.

2. Divide and Conquer

   Divide and conquer merupakan salah satu strategi yang mudah dipahami dan cukup banyak digunakan. Strategi ini akan membagi sebuah masalah yang besar menjadi beberapa masalah kecil. Kemudian masing-masing masalah kecil tersebut diselesaikan dan solusi yang dihasilkan disusun ulang hingga menjadi solusi utuh untuk permasalahan yang diberikan.

Silakan Anda membaca lebih lanjut terkait strategi algoritma dan pelajari juga beberapa contoh yang diberikan pada referensi.

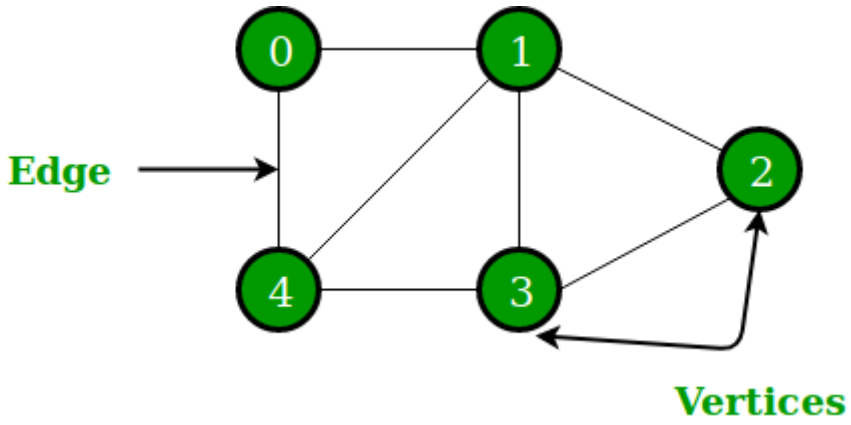
Referensi:

[1] https://www.geeksforgeeks.org/backtracking-algorithms/

[2] https://www.tutorialspoint.com/data_structures_algorithms/divide_and_conquer.htm

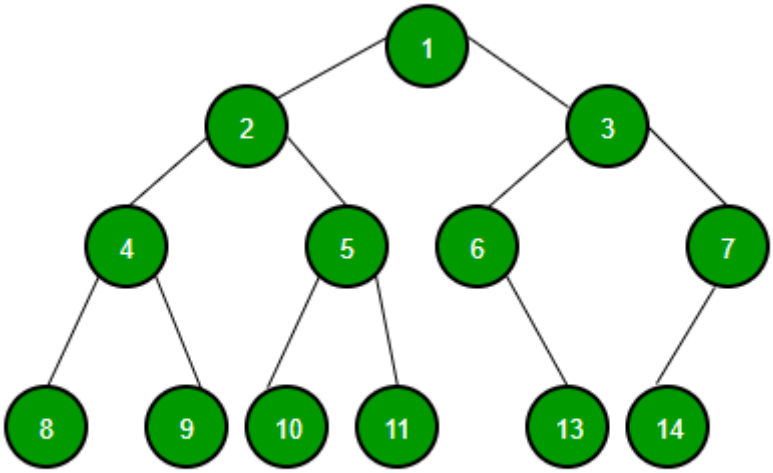
II.II. *Graph* dan *Tree*

Graph dan *Tree* merupakan struktur data non-linear yang dapat menyimpan data. *Graph* dan *Tree* cukup sering digunakan dalam merepresentasikan data. *Graph* merupakan kumpulan dari *vertices* (atau *node*) yang terhubung melalui *edges*. Hubungan antar *vertices* melalui *edges* tidak terikat oleh sebuah aturan. Gambar 21 merupakan contoh dari sebuah *graph*.



Gambar 21 Ilustrasi Graph

Tree merupakan kumpulan *nodes* yang saling terhubung. Berbeda dengan *Graph*, *Tree* memiliki aturan hubungan antar *nodes*. Tree memiliki sebuah *node* yang disebut *root* dan hubungan antar *node* hanya berjarak 1 level. Gambar 22 berikut merupakan contoh *tree*.



Gambar 22 Ilustrasi Tree

Untuk mengoperasikan *graph* dan *tree* terdapat dua algoritma yang sering digunakan yaitu *depth first search* (DFS) dan *breadth first search* (BFS). DFS, sesuai namanya, akan melakukan pencarian dengan cara masuk ke dalam sebuah cabang hingga ke *node* akhir kemudian berpindah ke cabang lainnya. Algoritma ini akan menggunakan *stack* untuk membantu mengingat *node* atau *vertex* mana yang akan dikunjungi selanjutnya apabila ditemukan jalan buntu. Khusus untuk struktur data *tree*, algoritma DFS masih dapat dibagi lagi ke dalam tiga jenis metode berdasarkan urutan aksesnya. Metode tersebut adalah *pre-order*, *in-order*, dan *post-order*. Berikut ini diberikan contoh alur pencarian menggunakan algoritma DFS untuk ilustrasi *graph* dan *tree* di atas.

```
Graph
0-1-2-3-4
0-1-4-3-2
0-4-3-2-1
0-4-3-1-2
0-4-1-2-3
0-4-1-3-2
```

Tree
1-2-4-8
1-2-4-9
1-2-5-10
1-2-5-11
1-3-6-13
1-3-7-14

Algoritma BFS akan melakukan pencarian dengan cara menjelajahi seluruh *node* atau *vertex* pada "level" yang sama kemudian melanjutkan pencarian ke "level" berikutnya. Algoritma BFS akan memanfaatkan *queue* untuk menentukan *node* atau *vertex* selanjutnya yang akan dikunjungi apabila ditemukan jalan buntu. Berikut ini merupakan contoh alur pencarian menggunakan algoritma BFS pada ilustrasi *graph* dan *tree* sebelumnya.

Graph
0
1-4
2-3

Tree
1
2-3
4-5-6-7
8-9-10-11-13-14

Secara umum algoritma DFS dan BFS diimplementasikan menggunakan strategi *backtracking* secara rekursif. Silakan pelajari lebih lanjut terkait *graph* dan *tree* secara khusus DFS dan BFS pada referensi yang diberikan.

Referensi:

[1] https://www.tutorialspoint.com/dsa_using_c/dsa_using_c_graph.htm
[2] https://www.tutorialspoint.com/dsa_using_c/dsa_using_c_tree.htm

III. Tutorial

Di bawah ini tersedia kode tutorial yang terkait dengan materi pada modul ini. Kode yang diberikan merupakan contoh salah satu masalah dan proses penyelesaiannya. Silakan Anda menyalin kode berikut dan menjalankannya pada komputer. Cermati alur berpikir yang digunakan pada program sambil membaca dan memahami komentar serta maksud dari kode yang diberikan.

Source Code 19 Kode Tutorial Modul 8

```
/** EL2208 Praktikum Pemecahan Masalah dengan C 2020/2021
 *  Modul               : 8 - Algorithms
 *  Percobaan           : Tutorial
 *  Hari dan Tanggal    : Rabu, 7 April 2021
 *  Nama File           : knapsack.c
 *  Pembuat             : Tito Irfan
 *  Deskripsi           : Penyelesaian 0/1 knapsack problem dengan
backtracking.
 */

/** Perintah Kompilasi
 *
 * Linux/MacOS  - gcc -o knapsack knapsack.c
 * Windows      - gcc -o knapsack.exe knapsack.c
 *
 * atau dengan Makefile:
 *
 * Linux/MacOS - make
 */

#include <stdio.h>
#include <stdlib.h>

// Struct daftar barang
typedef struct ItemList {
    int* value;
    int* weight;
    int data_amt;
} ItemList;

/** is_promising. Cek apakah solusi menjanjikan untuk ditelusuri lebih
jauh.
 *
 * Hitung batas atas keuntungan yang dapat diperoleh dengan menambah
barang
 * yang bisa ditambah secara greedy dan menambah fraksi harga barang yang
 * tersisa bila ada.
 *
 * Barang yang dicek untuk ditambah hanya barang yang memiliki indeks
setelah
 * indeks barang yang diperiksa saat ini.
 *
```

```
 * @param idx indeks barang yang diperiksa saat ini.
 * @param cur_value harga total barang saat ini.
 * @param cur_weight berat total barang saat ini.
 * @param best_value keuntungan terbaik yang telah ditemukan.
 * @param item_list list harga barang dan beratnya.
 * @param weight_limit batas berat barang.
 * @return 1 bila menjanjikan, 0 bila tidak.
 */
int is_promising(
    int idx,
    int cur_value,
    int cur_weight,
    int* best_value,
    ItemList* item_list,
    int weight_limit) {
    // Bila berat melebihi batas, solusi tidak menjanjikan
    if (cur_weight >= weight_limit)
        return 0;

    // Ambil barang dengan indeks setelah idx secara greedy, catat
keuntungan dan beratnya
    int it = idx + 1;
    float bound = cur_value;
    int weight_so_far = cur_weight;

    while (it < item_list->data_amt && weight_so_far + item_list-
>weight[it] <= weight_limit) {
        weight_so_far += item_list->weight[it];
        bound += item_list->value[it];
        ++it;
    }

    // Bila masih ada sisa barang, tambahkan fraksi harga barang tersebut
untuk
    // memenuhi ransel (seakan barang tersisa bisa dibagi)
    if (it < item_list->data_amt)
        bound += (weight_limit - weight_so_far) * item_list->value[it] /
(float) item_list->weight[it];

    // Bila batas atas keuntungan melebihi keuntungan terbesar saat ini,
solusi menjanjikan
    return bound > *best_value;
}

/** knapsack_recursive. Selesaikan masalah knapsack secara rekursif
 * dengan algoritma backtracking.
 *
 * @param idx indeks barang yang diperiksa saat ini.
 * @param cur_value harga barang total pada kombinasi ini.
 * @param cur_weight berat barang total pada kombinasi ini.
 * @param item_combination kombinasi barang.
 * @param best_value keuntungan terbaik yang telah ditemukan.
 * @param best_combination kombinasi yang menghasilkan keuntungan
terbaik.
 * @param item_list list harga barang dan beratnya.
 * @param weight_limit limit harga barang.
 * @result best_value dan best_combination menyimpan keuntungan terbaik
dan
 *  kombinasi yang menghasilkannya.
```

```
 */
void knapsack_recursive(
    int idx,
    int cur_value,
    int cur_weight,
    int* item_combination,
    int* best_value,
    int* best_combination,
    ItemList* item_list,
    int weight_limit) {
    // Cek apakah solusi saat ini merupakan terbaik
    if (cur_weight <= weight_limit && cur_value > *best_value) {
        // Set solusi saat ini sebagai solusi terbaik
        *best_value = cur_value;
        for (int i = 0; i < item_list->data_amt; ++i)
            best_combination[i] = item_combination[i];
    }

    // Bila solusi menjanjikan, lakukan rekursi penambahan barang
    if (is_promising(
            idx,
            cur_value,
            cur_weight,
            best_value,
            item_list,
            weight_limit)) {
        // Eksplor solusi dengan menyertakan barang pada idx
        item_combination[idx + 1] = 1;
        knapsack_recursive(
            idx + 1,
            cur_value + item_list->value[idx + 1],
            cur_weight + item_list->weight[idx + 1],
            item_combination,
            best_value,
            best_combination,
            item_list,
            weight_limit);

        // Backtrack, lanjutkan eksplor solusi tanpa menyertakan barang
pada idx
        item_combination[idx + 1] = 0;
        knapsack_recursive(
            idx + 1,
            cur_value,
            cur_weight,
            item_combination,
            best_value,
            best_combination,
            item_list,
            weight_limit);
    }
}

/** print_knapsack. Fungsi driver untuk menyelesaikan 0/1 knapsack
problem.
 *
 * @param item_list list harga dan berat barang.
 * @param weight_limit batas berat barang.
 * @result keuntungan terbanyak dan kombinasinya tercetak di layar.
```

```
 */
void print_knapsack(ItemList* item_list, int weight_limit) {
    int* item_combination = (int*) calloc(item_list->data_amt,
sizeof(int));
    int* best_combination = (int*) calloc(item_list->data_amt,
sizeof(int));
    int best_value = 0;

    knapsack_recursive(-1, 0, 0, item_combination, &best_value,
best_combination, item_list, weight_limit);

    printf("\nBest value: %d\n", best_value);

    printf("Best combination:\n");
    for (int i = 0; i < item_list->data_amt; ++i) {
        if (best_combination[i])
            printf("Item #%d, Value: %d, Weight: %d\n", i + 1, item_list-
>value[i], item_list->weight[i]);
    }
}

int main() {
    // Input barang dan batas berat secara dinamis
    ItemList item_list;
    int weight_limit;

    printf("Input data amount: ");
    scanf("%d", &(item_list.data_amt));

    item_list.value = (int*) malloc(item_list.data_amt * sizeof(int));
    item_list.weight = (int*) malloc(item_list.data_amt * sizeof(int));

    for (int i = 0; i < item_list.data_amt; ++i) {
        printf("\nInput item #%d value: ", i + 1);
        scanf("%d", &(item_list.value[i]));
        printf("Input item #%d weight: ", i + 1);
        scanf("%d", &(item_list.weight[i]));
    }

    printf("\nInput weight limit: ");
    scanf("%d", &weight_limit);

    // Selesaikan 0/1 knapsack problem
    print_knapsack(&item_list, weight_limit);

    return 0;
}
```