MODUL PRAKTIKUM
ALGORITMA DAN STRUKTUR DATA

Oleh:

Fidi Wincoko Putro, S.ST., M.Kom.
Pangestu Widodo, S.T., M.Kom.
Bernadus Anggo Seno Aji, S.Kom, M.Kom.
Mohammad Sholik, S.Kom, M.Kom.

FAKULTAS TEKNOLOGI INFORMASI DAN INDUSTRI

INSTITUT TEKNOLOGI TELKOM SURABAYA

SURABAYA

2019

DAFTAR ISI

BAB 1 TIPE DATA 4
1.1. Tujuan Praktikum 4
1.2. Latihan 4
1.2. Tugas 1 4
BAB 2 ABSTRACT DATA TYPE 5
2.1. Tujuan Praktikum 5
2.2. Teori Singkat 5
2.3. Latihan 6
2.4. Tugas 6
BAB 3 File Header dan File Implementasi di C++ 7
3.1. Tujuan Praktikum 7
3.2. Teori Singkat 7
BAB 4 STACK 9
4.1. Tujuan Praktikum 9
4.2. Teori Singkat 9
4.3. Latihan 11
BAB 5 QUEUE 12
5.1. Tujuan Praktikum 12
5.2. Teori Singkat 12
5.3. Latihan 16
5.4. Tugas 16
BAB 6 SINGLE LINKED LIST 17
6.1. Tujuan Praktikum 17
6.2. Teori Singkat 17
6.3. Latihan 23
6.4. Tugas 23
BAB 7 DOUBLE LINKED LIST 24
7.1. Tujuan Praktikum 24
7.2. Teori Singkat 24
7.3. Latihan 27

7.4. Tugas 27
BAB 8 CIRCULAR DOUBLE LINKED LIST 28
8.1. Tujuan Praktikum 28
8.2. Teori Singkat 28
8.3. Latihan 31
8.4. Tugas 31
BAB 9 STACK dan QUEUE dengan LINKED LIST 32
9.1. Tujuan Praktikum 32
9.2. Teori Singkat 32
9.2.1. STACK 32
9.2.2. QUEUE 34
9.3. Latihan 37
9.4. Tugas 37
BAB 10 Binary Search 38
10.1. Tujuan Praktikum 38
10.2. Teori Singkat 38
10.3. Latihan 39
10.4. Tugas 39
BAB 11 Interpolation Search 40
11.1. Tujuan Praktikum 40
11.2. Teori Singkat 40
11.3. Latihan 42
11.4. Tugas 42
BAB 12 Sorting Algorithm 43
12.1. Tujuan Praktikum 43
12.2. Teori Singkat 43
12.2.1. Bubble Sort 43
12.2.2. Selection Sort 44
12.2.3. Insertion Sort 45
12.3. Latihan 47
12.4. Tugas 47

BAB 1
TIPE DATA

1.1. Tujuan Praktikum

Setelah melakukan praktikum ini, diharapkan mahasiswa mampu:

1. Memahami struktur data yang digunakan pada antrian.
2. Memahami karakteristik antrian dan kondisi antrian baik kosong ataupun penuh.

1.2. Latihan

Mahasiswa melakukan percobaan input dan output menggunakan printf dan scanf untuk beberapa tipe data di bawah ini:

Tipe Data	Penentu Format
Integer	%d
Floating Point Bentuk Decimal	%f
Floating Point Bentuk Berpangkat	%e
Floating Point yang lebih pendek antara Decimal dan Berpangkat	%g
Double Precision	%lf
Character	%c
String	%s
Unsigned Integer	%u
Long Integer	%ld
Long Unsigned Integer	%lu
Unsigned hexadecimal integer	%x
Unsigned octal integer	%o

1.2. Tugas 1

Mahasiswa membuat program dengan bahasa C/C++ dengan ketentuan sebagai berikut:

- Buatlah program input dalam tipe data char array
- Berisi nama, tanggal lahir, bulan lahir, tahun lahir, jenis kelamin
- Kemudian tampilkan dalam bentuk tipe data string
- Kumpulkan source-code.cpp nya
- Screenshot hasilnya
- Beri penjelasan analisanya

BAB 2
ABSTRACT DATA TYPE

2.1. Tujuan Praktikum

- Mahasiswa memahami tipe data bentukan di C++
- Mahasiswa memahami penggunaan tipe data bentukan

2.2. Teori Singkat

Abstract Data Type adalah tipe data bentukan atau campuran (composite))yang terdiri dari type dan sekumpulan operasi dasar terhadap type tersebut. Definisi type dari sebuat ADT dapat mengandung sebuah definisi ADT lain. Misalnya:

2.1.1.1. ADT tanggal_lahir yang terdiri dari ADT tanggal, bulan dan tahun.
2.1.1.2. ADT titik terdiri dari dua buah nilai integer x dan y.
2.1.1.3. ADT garis yang terdiri dari dua buah ADT titik.
2.1.1.4. ADT segi_empat terdiri dari empat buah ADT garis.

Bentuk umum deklarasi suatu ADT adalah sebagai berikut:

```
struct nama_tipe_data_bentukan()
{
   Tipe nama_variabel_1;
   . . .
   . . .
   Tipe nama_variabel_n;
} variabel_bentukan1,. . ., variabel_bentukanM;
```

Atribut dari tipe_data_bentukan dapat diakses dengan menggunakan bentuk:

```
Variabel_bentukan.nama_variabel
```

Contoh program:

```
1.  #include <iostream>
2.  using namespace std;
3.
4.  struct tanggal_lahir{
5.      int tgl;
6.      int bln;
7.      int thn;
```

```
8.  };
9.
10. int main()
11. {
12.     tanggal_lahir tgl_lhr;
13.     cout << "Masukkan Tanggal Lahir!" << endl;
14.     cout << "Masukkan tanggalnya:" << endl;
15.     cin >> tgl_lhr.tgl;
16.     cout << "Masukkan bulannya:" << endl;
17.     cin >> tgl_lhr.bln;
18.     cout << "Masukkan tahunnya:" << endl;
19.     cin >> tgl_lhr.thn;
20.     cout << "Jadi Tanggal Lahir Anda, Tanggal:" << tgl_lhr.tgl
     << " bulan:" << tgl_lhr.bln << " tahun:" <<tgl_lhr.thn;
21.     return 0;
22. }
```

2.3. Latihan

1. Buatlah program input dan output dengan menggunakan Abstract Data Type dengan nama ‘mahasiswa’ yang memiliki atribut: nama (char[20]), usia (int), mata_kuliah (char[50]) dan nilai (float).
2. Berdasarkan program pada praktikum no. 1, tambahkan ADT ‘tanggal_lahir’ dengan atribut sebagai berikut: tanggal (int), bulan (char[10]), tahun (int).
3. Berdasarkan program pada praktikum no. 1 dan 2, tambahkan atribut ‘tgl_lhr’ (tanggal_lahir) pada ADT ‘mahasiswa’.

2.4. Tugas

1. Buatlah program ‘jasa_kurir’ berupa input dan output yang menggunakan ADT ‘pengirim’, ADT ‘alamat’, ADT ‘barang’, dan ADT ‘penerima’. Silahkan tentukan sendiri atribut yang sesuai.

BAB 3
File Header dan File Implementasi di C++

3.1. Tujuan Praktikum

- Mahasiswa memahami pemanfaatan file header dan file implementasi di C++
- Mahasiswa mampu menuliskan program menggunakan file header dan file implementasi

3.2. Teori Singkat

Program yang ditulis menggunakan C++ umumnya terbagi menjadi dua kelompok file: file header dan file implementasi. File header berisi deklarasi kelas (anda akan mempelajari tentang kelas di mata kuliah Pemrograman Berorientasi Objek), tipe data bentukan, fungsi/prosedur, dan konstanta global. File implementasi berisi implementasi fungsi dan variabel global.

Contoh file header mahasiswa.h

```
1.  #ifndef MAHASISWA_H_INCLUDED
2.  #define MAHASISWA_H_INCLUDED
3.
4.  struct mahasiswa {
5.      char name[30];
6.      int NIM;
7.      char department[20];
8.  };
9.
10. void inputDataMahasiswa(mahasiswa& dataMahasiswa);
11.
12. void printDataMahasiswa(mahasiswa dataMahasiswa);
13.
14. #endif // MAHASISWA_H_INCLUDED
```

Pada file header diatas dideklarasikan satu buah tipe data bentukan yaitu `mahasiswa` dan dua prosedur yaitu `inputDataMahasiswa` dan `printDataMahasiswa`. Di bagian atas file dituliskan

```
#ifndef MAHASISWA_H_INCLUDED
#define MAHASISWA_H_INCLUDED
```

beserta baris berikut di bagian akhir file

```
#endif
```

yang berguna untuk menghindari error yang dapat terjadi bila file header tersebut digunakan di lebih dari satu file implementasi. Kata `MAHASISWA_H_INCLUDED` dapat diubah sesuai kebutuhan (tidak boleh kembar dengan file header yang lain).

Contoh file-file implementasi yang memanfaatkan file header diatas:

File mahasiswa.cpp

```
1.  #include <iostream>
2.  #include "mahasiswa.h"
3.
4.  using namespace std;
5.
6.  void inputDataMahasiswa(mahasiswa &dataMahasiswa)
7.  {
8.      cout << "Masukkan nama mahasiswa: ";
9.      cin >> dataMahasiswa.name;
10.
11.     cout << "Masukkan NIM mahasiswa: ";
12.     cin >> dataMahasiswa.NIM;
13.
14.     cout << "Masukkan nama program studi mahasiswa: ";
15.     cin >> dataMahasiswa.department;
16. }
17.
18. void printDataMahasiswa(mahasiswa dataMahasiswa)
19. {
20.     cout << "Nama mahasiswa  : " << dataMahasiswa.name << endl;
21.     cout << "NIM             : " << dataMahasiswa.NIM << endl;
22.     cout << "Program Studi   : " << dataMahasiswa.department <<
    endl;
23. }
```

File main.cpp

```
1.  #include <iostream>
2.  #include "mahasiswa.h"
3.
4.  using namespace std;
5.
6.  int main()
7.  {
8.      mahasiswa m;
9.
10.     inputDataMahasiswa(m);
11.
12.     printDataMahasiswa(m);
13.
14.     return 0;
15. }
```

Dalam contoh diatas, file header mahasiswa.h digunakan di dua file implementasi yaitu mahasiswa.cpp dan main.cpp. File implementasi mahasiswa.cpp berisi implementasi fungsi-fungsi yang dideklarasikan di dalam file header mahasiswa.h, sementara file main.cpp berisi fungsi main() yang memanfaatkan fungsi-fungsi tersebut.

BAB 4
STACK

4.1. Tujuan Praktikum

- Mahasiswa memahami implementasi stack menggunakan C++
- Mahasiswa mampu memecahkan masalah menggunakan stack yang diimplementasikan dalam bahasa C++

4.2. Teori Singkat

Stack, sesuai artinya yaitu tumpukan, merupakan stuktur penyimpanan data dengan bentuk menyerupai tumpukan. Operasi pada Stack adalah LIFO (Last In First Out), yang artinya data yang terakhir masuk adalah data yang pertama dikeluarkan. Anda bisa membayangkan stack seperti sebuah tong, dimana barang yang terakhir dimasukkan adalah barang yang pertama dikeluarkan.

Dalam praktikum kali ini, stack diimplementasikan menggunakan array sebagai struktur dasar. Karena operasi LIFO yang telah disebutkan diatas, maka ada dua fungsi dasar yang harus diimplementasikan pada sebuah stack yaitu:

- Push: memasukkan data baru ke dalam stack
- Pop: mengambil data paling atas dari stack

Selain fungsi diatas, ada beberapa fungsi tambahan yang dapat diimplementasikan, misalnya:

- Peek: mengintip/melihat data yang berada di posisi paling atas (bukan mengambil)
- IsEmpty: melihat apakah stack kosong atau tidak
- IsFull: melihat apakah stack penuh atau tidak
- Dan lain-lain

Berikut adalah contoh deklarasi dan implementasi stack, serta pemanfaatan stack di dalam program:

File stack.h

```
1.  #ifndef STACK_H_INCLUDED
2.  #define STACK_H_INCLUDED
3.
4.  const int maxStackSize = 10;
5.
6.  struct stackOfInt
7.  {
8.      int data [maxStackSize];
9.      int top = -1;
```

```
10. };
11.
12. void push(int nomorISBN);
13. int pop();
14. int peek();
15. bool isEmpty();
16. bool isFull();
17.
18. #endif // STACK_H_INCLUDED
```

File stack.cpp

```
1.   #include "stack.h"
2.
3.   stackOfInt stackBuku;
4.
5.   void push(int nomorISBN)
6.   {
7.       stackBuku.top++;
8.       stackBuku.data[stackBuku.top] = nomorISBN;
9.   }
10.
11.  int pop()
12.  {
13.      int lastTop = stackBuku.top;
14.      stackBuku.top--;
15.      return stackBuku.data[lastTop];
16.  }
17.
18.  int peek()
19.  {
20.      return stackBuku.data[stackBuku.top];
21.  }
22.
23.  bool isEmpty()
24.  {
25.      return (stackBuku.top == -1);
26.  }
27.
28.  bool isFull()
29.  {
30.      return (stackBuku.top == maxStackSize-1);
31.  }
```

File main.cpp

```
1. #include <iostream>
2. #include "stack.h"
3.
4. using namespace std;
5.
6. void masukkanDataBukuKedalamStack()
7. {
```

```
8.      bool stop = false;
9.
10.     while (!stop)
11.     {
12.         cout << "Masukkan nomor ISBN : ";
13.         int nomorISBN;
14.         cin >> nomorISBN;
15.         push(nomorISBN);
16.
17.         cout << "Lagi ?";
18.         char jawaban;
19.         cin >> jawaban;
20.
21.         stop = ((jawaban == 'n') || (jawaban == 'N'));
22.     }
23. }
24.
25. void printDataBukuDiDalamStack()
26. {
27.     //tampilkan data buku di dalam stack
28.     while (!isEmpty())
29.     {
30.         int nomorISBN = pop();
31.         cout << "Nomor ISBN Buku : " << nomorISBN <<endl;
32.     }
33. }
34.
35. int main()
36. {
37.     masukkanDataBukuKedalamStack();
38.
39.     printDataBukuDiDalamStack();
40.
41.     return 0;
42. }
```

4.3. Latihan

1. Perhatikan bahwa fungsi pop() dan fungsi push() pada contoh diatas beresiko mengalami error apabila dipanggil terus menerus. Jelaskan mengapa dan perbaiki kode program kedua fungsi tersebut agar tidak terjadi error.

2. Tambahkan fungsi baru `int currentStackSize()` pada stack.h dan stack.cpp diatas yang menghasilkan nilai kembalian (*return value*) berupa jumlah elemen di dalam stack. Manfaatkan fungsi baru tersebut di dalam fungsi `printDataBukuDiDalamStack()` usehingga tampilan output menjadi seperti berikut:

```
Jumlah buku di dalam tumpukan: XX
Nomor ISBN buku ke-yy : AAAAAA
Nomor ISBN buku ke-zz : BBBBBB
.....
```

3. Pada contoh program diatas, isi data yang ditampung di dalam stack hanya berupa nomor ISBN (berupa *array of integer*). Ubahlah file stack.h, stack.cpp, dan main.cpp sehingga isi stack adalah struct yang berisi nomor ISBN dan judul buku.

BAB 5
QUEUE

5.1. Tujuan Praktikum

Setelah melakukan praktikum ini, diharapkan mahasiswa mampu:

1. Memahami struktur data yang digunakan pada antrian.
2. Memahami karakteristik antrian dan kondisi antrian baik kosong ataupun penuh.

5.2. Teori Singkat

Antrian atau disebut *queue* adalah konsep penyimpanan dari sekumpulan data yang sering digunakan. Antrian data bisa disimpan dalam bentuk *Array* atau *Linked list*. Elemen queue hanya diperbolehkan untuk dimasukkan dari belakang dan hanya diperbolehkan dikeluarkan dari depan. Aturan antrian tersebut disebut *FIFO* yang merupakan singkatan dari *First In First Out.* Adapun beberapa karakteristik penting dari antrian adalah sebagai berikut:

1. Elemen antrian yaitu data yang tersimpan didalam antrian.
2. Front/head (penunjuk elemen terdepan dari antrian).
3. Rear/tail (penunjuk elemen terakhir dari antrian).
4. Count (jumlah elemen pada antrian)
5. Kondisi penuh, yaitu kondisi dimana antrian mencapai kapasitas maksimum antrian. Pada kondisi ini tidak bisa menambahkan pada antrian lagi, karena akan menyebabkan *Overflow.*
6. Kondisi kosong, yaitu kondisi dimana tidak ada elemen pada antrian sehingga tidak memungkinkan dilakukan pengambilan elemen dari antrian. Bila dipaksakan untuk mengambil akan menyebabkan *overflow*.
7. Enqueue, yaitu proses menambahkan elemen ke dalam antrian.
8. Dequeue, yaitu proses pengambilan elemen dari antrian.

Pada praktikum ini antrian yang akan kita implementasikan adalah antrian dalam bentuk *array*. Antrian dalam implementasinya ada beberapa alternatif, di antaranya adalah sebagai berikut:

1. Linear Queue, yaitu antrian dalam implementasinya penambahan elemen antrian dilakukan di belakang (rear/tail). Kemudian setelah penghapusan atau pengambilan elemen antrian terdepan (front/head), maka seluruh elemen antrian setelah front

sampai rear akan bergeser maju satu langkah. Contohnya adalah antrian penjualan tiket.

2. Circular Queue, yaitu atrian yang dalam implementasinya penambahan elemen antrian dilakukan di belakang rear. Jika index elemen sudah mencapai batas count (MAX) tetapi masih ada ruang kosong di index sebelum rear, maka elemen antrian ditambahkan di index pertama dan seterusnya sampai sebelum rear. Sedangkan penghapusan elemen antrian dilakukan di front kemudian index front pindah ke elemen selanjutnya. Contohnya adalah Manajemen Memori, CPU scheduling, dsb.

Contoh program:

Linear Queue:

IntQueue.h

```
1.    #ifndef INTQUEUE_H_INCLUDED
2.    #define INTQUEUE_H_INCLUDED
3.
4.    #define MAX_QUEUE_SIZE 10
5.    #define EMPTY_QUEUE_HEAD_INDEX -1
6.    #define EMPTY_QUEUE_TAIL_INDEX -1
7.
8.    struct IntQueue {
9.        int head;
10.       int tail;
11.       int data[MAX_QUEUE_SIZE];
12.   };
13.
14.   void createEmpty(IntQueue &q);
15.   void enqueue(IntQueue &q, int input);
16.   void dequeue(IntQueue &q, int &output);
17.   bool isEmpty(IntQueue q);
18.   bool isFull(IntQueue q);
19.
20.   #endif // INTQUEUE_H_INCLUDED
```

IntQueue.cpp

```
1.    #include "IntQueue.h"
2.
3.
4.    bool isEmpty(IntQueue q)
5.    {
6.        return (q.tail == EMPTY_QUEUE_TAIL_INDEX);
7.    }
8.
9.    bool isFull(IntQueue q)
10.   {
11.       int arrayLength = sizeof(q.data) / sizeof(q.data[0]);
12.       return ( q.tail == (arrayLength -1) );
```

```
13.  }
14.
15.  void createEmpty(IntQueue &q)
16.  {
17.      q.head = EMPTY_QUEUE_HEAD_INDEX;
18.      q.tail = EMPTY_QUEUE_TAIL_INDEX;
19.  }
20.
21.  void enqueue(IntQueue &q, int input)
22.  {
23.      if (isEmpty(q))
24.      {
25.          q.head = 0;
26.          q.tail = q.head;
27.          q.data[q.tail] = input;
28.      } else
29.      {
30.          q.tail++;
31.          q.data[q.tail] = input;
32.      }
33.  }
34.
35.  void dequeue(IntQueue &q, int &output)
36.  {
37.      if  (!isEmpty(q))
38.      {
39.          output = q.data[q.head];
40.
41.          if (q.tail > q.head)
42.          {
43.              for (int i = q.head; i < q.tail; i++)
44.              {
45.                  q.data[i] = q.data[i+1];
46.              }
47.              q.tail--;
48.          } else
49.          {
50.              q.head = EMPTY_QUEUE_HEAD_INDEX;
51.              q.tail = EMPTY_QUEUE_TAIL_INDEX;
52.          }
53.      }
54.  }
```

main.cpp

```
1.   #include <iostream>
2.   #include <stdlib.h>
3.   #include "IntQueue.h"
4.
5.   using namespace std;
6.
7.   void printSeluruhIsiArrayQueue(IntQueue q)
8.   {
9.       //tampilkan data buku di dalam stack
10.      cout << "Isi array: ";
11.      for (int i = 0; i < MAX_QUEUE_SIZE; i++)
12.      {
13.          if (q.head <= q.tail)
14.          {
```

```
15.                 if ((q.head <= i) && (i <= q.tail))
16.                 {
17.                     cout << " [" << q.data[i] << "]";
18.                 } else
19.                 {
20.                     cout << " [ ]";
21.                 }
22.             } else
23.             {
24.                 if ((i <= q.tail) || (i >= q.head))
25.                 {
26.                     cout << " [" << q.data[i] << "]";
27.                 } else
28.                 {
29.                     cout << " [ ]";
30.                 }
31.             }
32.         }
33.         cout << endl;
34.         cout << "q.head: " << q.head << endl;
35.         cout << "q.tail: " << q.tail << endl;
36.     }
37.
38.     void masukkanDataIntegerKeDalamQueue(IntQueue &q)
39.     {
40.         cout << "Masukkan angka yang ingin dimasukkan : ";
41.         int n;
42.         cin >> n;
43.         enqueue(q, n);
44.     }
45.
46.     void ambilDataIntegerDariDalamQueue(IntQueue &q)
47.     {
48.         int n;
49.         dequeue(q, n);
50.     }
51.
52.     int main()
53.     {
54.         IntQueue iq;
55.         createEmpty(iq);
56.
57.         bool stop = false;
58.
59.         while (!stop)
60.         {
61.             system("cls");
62.             printSeluruhIsiArrayQueue(iq);
63.
64.             cout << "Menu: " << endl;
65.             cout << "1. Enqueue" << endl;
66.             cout << "2. Dequeue" << endl;
67.             cout << "Pilihan anda: ";
68.             char jawaban;
69.             cin >> jawaban;
70.
71.             if (jawaban == '1')
72.             {
73.                 masukkanDataIntegerKeDalamQueue(iq);
74.             } else
```

```
75.             {
76.                 ambilDataIntegerDariDalamQueue(iq);
77.             }
78.
79.             stop = ((jawaban != '1') && (jawaban != '2'));
80.         }
81.
82.
83.
84.         return 0;
85.     }
```

5.3. Latihan

1. Perhatikan bahwa salah satu atau kedua fungsi enqueue() dan fungsi dequeue() pada kedua contoh diatas beresiko mengalami error apabila dipanggil terus menerus. Jelaskan mengapa dan perbaiki kode program kedua fungsi tersebut agar tidak terjadi error.
2. Kedua contoh diatas menggunakan model implementasi queue menggunakan Head dan Tail. Ubahlah kedua contoh diatas menjadi mempergunakan model Head dan Count.

5.4. Tugas

1. Pada kedua contoh program diatas, isi data yang ditampung di dalam stack hanya berupa angka (berupa array of integer). Ubahlah kedua contoh diatas sehingga isi queue adalah struct yang berisi nomor resi dan deskripsi isi paket (studi kasus jasa logistik).
2. Tambahkan fungsi search berdasarkan deskripsi isi paket menggunakan fungsi find().

BAB 6
SINGLE LINKED LIST

6.1. Tujuan Praktikum

Setelah melakukan praktikum ini, diharapkan mahasiswa mampu:

1. Memahami konsep struktur data *Single Linked List*.
2. Memahami konsep operasi-operasi dasar pada *Single Linked List.*
3. Membuat program dengan menggunakan *Single Linked List*.

6.2. Teori Singkat

Linked List atau biasa disebut List saja, adalah salah satu bentuk representasi penyimpanan data yang berupa rangkaian elemen data yang saling terhubung atau terkait dan bersifat dinamis. Dinamis disini yang dimaksud adalah kapasitasnya bisa berubah-ubah, baik membesar ataupun mengecil sesuai kebutuhan. Data yang disimpan dalam bentuk list bisa berupa data tunggal ataupun data majemuk. Data tunggal adalah data yang terdiri dari satu variable. Sedangkan data majemuk adalah data yang terdiri dari berbagai tipe data, misalnya data mahasiswa, terdiri dari variabel nama yang bertipe data string, NIM bertipe integer dan sebagainya. Linked List bisa diimplementasikan dengan menggunakan Array atau Pointer.

Pada praktikum ini akan menggunakan pointer untuk membentuk Linked List. Adapun kelebihannya adalah sebagai berikut:

1. Array bersifat statis, sedangkan pointer dinamis.
2. Data pada linked list saing terkait sehingga lebih mudah menggunakan pointer.
3. Linked List bersifat fleksibel sehingga lebih sesuai dengan sifat pointer yang bisa diatur sesuai kebutuhan.
4. Array relatif lebih susah untuk menangani linked list dari pada pointer.
5. Array lebih sesuai dengan data yang sudah diketahui jumlah elemen maksimumnya dari awal.

Secara umum operasi yang sering digunakan pada linked list, yaitu sebagai berikut:

1. Pembuatan dan inisialisasi list (Create List).
2. Penyisipan elemen list (insert).
3. Penghapusan elemen list (Delete).
4. Menampilkan elemen list (View).
5. Pencarian elemen list (searching).

6. Pengubahan isi elemen list (Update).

Contoh Program:

Pada program di bawah ini merupakan contoh linked list yang menerapkan operasi penyisipan (insert).

listexplicit.h

```
#ifndef LISTEXPLICIT_H_INCLUDED
#define LISTEXPLICIT_H_INCLUDED

struct listelement{
    int info;
    listelement* next;
};

struct listexplicit {
    listelement* first;
};

void createempty(listexplicit& l);
void insertfirst(listexplicit& l, listelement* inputelement);
void insertfirst(listexplicit& l, int inputinteger);

#endif // LISTEXPLICIT_H_INCLUDED
```

Listexplicit.cpp

```
#include <iostream>
#include "listexplicit.h"

void createempty(listexplicit& l)
{
    l.first = NULL;
}

void insertfirst(listexplicit& l, listelement* inputelement)
{
    //kondisi awal: l bisa kosong (artinya l.first == NULL)
    //atau tidak kosong (artinya l.first
    //berisi sebuah pointer ke sebuah listelement

    //tahap 1: memindahkan seluruh isi list sebagai elemen
sesudah input
    inputelement->next = l.first;

    //tahap 2: mengubah agar l.first menunjuk pada input
    l.first = inputelement;
}

void insertfirst(listexplicit& l, int inputinteger)
{
    //membuat elemen baru dengan parameter input sebagai isi
field info
    listelement* le = new listelement;
```

```
26.       le->info = inputinteger;
27.       le->next = NULL;
28.
29.       //memasukkan elemen yang telah dibuat ke dalam list
30.       insertfirst(l, le);
31.   }
```

main.cpp

```
1.    #include <iostream>
2.    #include "listexplicit.h"
3.
4.    using namespace std;
5.
6.    void printseluruhisilist(listexplicit l)
7.    {
8.        //tampilkan data buku di dalam stack
9.        cout << "Isi list: ";
10.       listelement* currentelement = l.first;
11.       while (currentelement != NULL)
12.       {
13.           cout << " [" << currentelement->info << "]";
14.           currentelement = currentelement->next;
15.       }
16.       cout << endl;
17.   }
18.
19.   void masukkandatasebagaielemenpertamalist(listexplicit& l)
20.   {
21.       cout << "Masukkan angka : ";
22.       int n;
23.       cin >> n;
24.       insertfirst(l, n);
25.   }
26.
27.   int main()
28.   {
29.       listexplicit l;
30.       createempty(l);
31.
32.       bool stop = false;
33.
34.       while (!stop)
35.       {
36.           system("clear");
37.           printseluruhisilist(l);
38.
39.           cout << "Menu: " << endl;
40.           cout << "1. Masukkan data sebagai elemen pertama"
<< endl;
41.           cout << "Selain itu: Keluar" << endl;
42.           cout << "Pilihan anda: ";
43.           char jawaban;
44.           cin >> jawaban;
45.
46.           if (jawaban == '1')
47.           {
48.               masukkandatasebagaielemenpertamalist(l);
49.           }
```

```
50.
51.             stop = (jawaban != '1');
52.         }
53.
54.
55.
56.         return 0;
57.     }
```

Contoh program antrian dengan linked list:

listqueue.h

```
1. #ifndef LISTANTRIAN_H
2. #define LISTANTRIAN_H
3.
4. #include <string.h>
5.
6. #define first(A) A.first
7. #define info(A) A->info
8. #define next(A) A->next
9.
10.    using namespace std;
11.
12.    struct elementdata {
13.        int queueno;
14.        string name;
15.    };
16.
17.    struct listelement{
18.        elementdata info;
19.        listelement* next;
20.    };
21.
22.    struct listexplicit {
23.        listelement* first;
24.    };
25.
26.    void createempty(listexplicit& l);
27.    bool isempty(listexplicit l);
28.
29.    void insertlast(listexplicit& l, listelement*
   inputelement);
30.    void insertlast(listexplicit& l, elementdata info);
31.
32.    void deletefirst(listexplicit& l, listelement*
   outputelement);
33.
34.    listelement* findelementbefore(listexplicit& l, int
   queueno);
35.    //fungsi findelementbefore mengembalikan elemen terakhir
   sebelum
36.    //elemen dengan nomor antrian queueno. apabila tidak
   ditemukan,
37.    //maka fungsi mengembalikan NULL
38.    void deleteafter(listelement* prec, listelement* p);
39.
40.    #endif // LISTANTRIAN_H
```

listqueue.cpp

```
1. #include <iostream>
2. #include "listqueue.h"
3.
4. void createempty(listexplicit& l)
5. {
6.     first(l) = NULL;
7. }
8.
9. bool isempty(listexplicit l)
10.    {
11.        return (first(l) == NULL);
12.    }
13.
14.    void insertlast(listexplicit& l, listelement* inputelement)
15.    {
16.        //kondisi awal: l bisa kosong (artinya l.first == NULL)
17.        //atau tidak kosong (artinya l.first
18.        //berisi sebuah pointer ke sebuah listelement
19.
20.        if (isempty(l))
21.        {
22.            first(l) = inputelement;
23.        } else
24.        {
25.            //tahap 1: menyusuri list hingga mencapai elemen
   akhir dimana next berisi NULL
26.            listelement* curr = first(l);
27.            while (next(curr) != NULL) {
28.                curr = next(curr);
29.            }
30.
31.            //tahap 2: mengubah agar next(curr) menunjuk pada
   inputelement
32.            next(curr) = inputelement;
33.        }
34.
35.
36.    }
37.
38.    void insertlast(listexplicit& l, elementdata info)
39.    {
40.        //membuat elemen baru
41.        listelement* newelement = new listelement;
42.        info(newelement) = info;
43.        next(newelement) = NULL;
44.
45.        //memasukkan elemen yang telah dibuat ke dalam list
46.        insertlast(l, newelement);
47.    }
```

main.cpp

```
1. #include <iostream>
2. #include "listqueue.h"
3.
4. using namespace std;
5.
6. void printantrian(listexplicit l)
7. {
8.     //tampilkan data buku di dalam stack
9.     cout << "Antrian saat ini: " <<endl;
10.        listelement* currentelement = first(l);
11.        while (currentelement != NULL)
12.        {
13.            elementdata data = info(currentelement);
14.            cout << " [" << data.queueno << " : " << data.name
   << "]" <<endl;
15.            currentelement = next(currentelement);
16.        }
17.        cout << endl;
18.    }
19.
20.    void antrianbaru(listexplicit& l, int nomorantrian)
21.    {
22.        cout << "Masukkan nama : ";
23.        string name;
24.        cin >> name;
25.
26.        elementdata info;
27.        info.queueno = nomorantrian;
28.        info.name = name;
29.
30.        insertlast(l, info);
31.    }
32.
33.    int main()
34.    {
35.        int queueno = 0;
36.        listexplicit l;
37.        createempty(l);
38.
39.        bool stop = false;
40.
41.        while (!stop)
42.        {
43.            system("clear");
44.            printantrian(l);
45.
46.            cout << "Menu: " << endl;
47.            cout << "1. Antrian baru" << endl;
48.            cout << "2. Pelanggan selesai dilayani" << endl;
49.            cout << "3. Pelanggan tidak hadir" << endl;
50.            cout << "4. Pelanggan tidak jadi mengantri" <<
   endl;
51.            cout << "Selain itu: Keluar" << endl;
52.            cout << "Pilihan anda: ";
53.            char jawaban;
54.            cin >> jawaban;
55.
56.            switch(jawaban) {
57.                case '1':
58.                    queueno++;
```

```
59.                     antrianbaru(l, queueno);
60.                     break;
61.                 case '2':
62.
63.                     break;
64.                 case '3':
65.
66.                     break;
67.                 case '4':
68.
69.                     break;
70.                 default:
71.                     stop = true;
72.                     break;
73.             }
74.         }
75.
76.
77.
78.         return 0;
79.     }
```

6.3. Latihan

1. Pada contoh program diatas, data yang dikandung setiap elemen adalah sebuah angka. Ubahlah contoh program diatas supaya data yang dikandung di dalam elemen adalah ID dan name. Sesuaikan fungsi-fungsi yang terdapat di dalam file listexplicit.h dan listexplicit.cpp dengan kondisi baru ini.

6.4. Tugas

1. Tambahkan sebuah fungsi baru di file listexplicit.h untuk mencari sebuah elemen berdasarkan ID tertentu sebagai berikut:

   listelement* getelement(listexplicit l, int inputID);

   lalu implementasikan fungsi tersebut di listexplicit.cpp, dan gunakan di file main.cpp untuk mencari nama berdasarkan ID tertentu.

BAB 7
DOUBLE LINKED LIST

7.1. Tujuan Praktikum

Setelah praktikum ini mahasiswa diharapkan mampu:

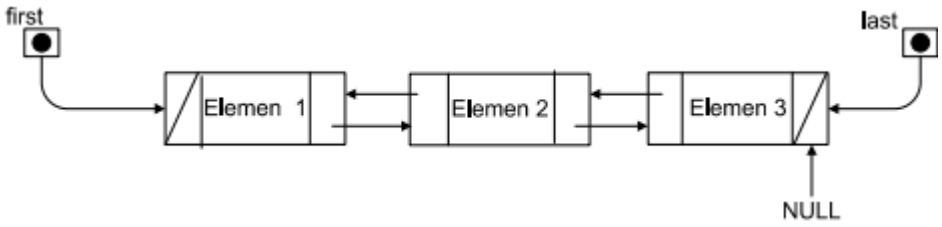
1. Memahami konsep Double Linked List.
2. Membuat program dengan mengimplementasikan konsep Double Linked List.

7.2. Teori Singkat

Double Linked List sebenarnya serupa dengan Single Linked List tetapi memiliki 2 pointer yang menunjuk pada elemen sebelumnya (prev) dan pointer yang menunjuk pada elemen sesudahnya (next). Double Linked List juga menggunakan dua buah pointer yang menunjukkan first (node pertama) dan pointer yang menunjukkan last (node terakhir) pada list.

Komponen – komponen yang terdapat pada Double Linked List adalah sebagai berikut:

1. First : Pointer pada list yang menunjuk pada elemen pertama list.
2. Last : Pointer pada list yang menunjuk pada lemen terakhir list.
3. Next : Pointer pada elemen yang menunjuk pada elemen di depannya.
4. Prev : Pointer pada elemen yang menunjuk pada elemen di belakangnya.



Contoh program:

```
1. #include <iostream>
2. #include <stdio.h>
3. #include <conio.h>
4. #include <ctype.h>
5. #include <stdlib.h>
6.
7. using namespace std;
8.
9. int x,y;
10.
11.    typedef struct node{
12.       int info;
13.       struct node *LEFT, *RIGHT;
14.    };
15.
16.    typedef struct node simpul;
17.    simpul *FIRST, *LAST, *P, *Q, *cetak;
18.
19.    void inisialisasi(){
20.       FIRST=NULL;
21.    }
22.
```

```
23.    void buatsimpul(int x){
24.       P=(simpul *)malloc(sizeof(simpul));
25.       if(P!=NULL){
26.          P->info=x;
27.       }
28.       else{
29.           cout<<"Gagal";
30.       }
31.    }
32.
33.    void simpulawal(){
34.       if(FIRST==NULL){
35.          FIRST=P;
36.          LAST=P;
37.          LAST->RIGHT=NULL;
38.          LAST->LEFT=NULL;
39.       }
40.    }
41.
42.    void insertkanan(){
43.      if(LAST!=NULL){
44.         LAST->RIGHT=P;
45.         P->LEFT=LAST;
46.         LAST=P;
47.         P->RIGHT=NULL;
48.      }
49.      else{
50.         cout<<"Isinya Kosong";
51.      }
52.    }
53.
54.    void insertkiri(){
55.      if(FIRST!=NULL){
56.         P->RIGHT=FIRST;
57.         FIRST->LEFT=P;
58.         FIRST=P;
59.         P->LEFT=NULL;
60.      }
61.      else{
62.         cout<<"Isinya Kosong";
63.      }
64.    }
65.
66.    void inserttengah(int y){
67.        Q=FIRST;
68.        while(Q->info!=y){
69.            Q=Q->RIGHT;
70.        }
71.        if(Q==LAST){
72.            LAST->RIGHT=P;
73.            P->LEFT=LAST;
74.            LAST=P;
75.            P->RIGHT=NULL;
76.        }
77.        else{
78.            P->RIGHT=Q->RIGHT;
79.            P->LEFT=Q->RIGHT;
80.            Q->RIGHT->LEFT=P;
81.            Q->RIGHT=P;
82.        }
```

```
83.    }
84.
85.    void deletekanan(){
86.      if(FIRST==LAST){
87.        FIRST=NULL;
88.        LAST=NULL;
89.      }
90.      else{
91.        LAST=LAST->LEFT;
92.        free(LAST->RIGHT);
93.        LAST->RIGHT=NULL;
94.      }
95.    }
96.
97.    void tampil(){
98.      cetak=FIRST;
99.      while(cetak!=NULL){
100.         cout<<cetak->info<<" ";
101.         cetak=cetak->RIGHT;
102.     }
103.   }
104.
105.   int main(){
106.       int pil;
107.       inisialisasi();
108.       do{
109.               //system("cls");
110.               cout<<endl<<endl<<"1. Insert Kanan"<<endl;
111.               cout<<"2. Insert Kiri"<<endl;
112.               cout<<"3. Insert Tengah"<<endl;
113.               cout<<"4. delete kanan"<<endl;
114.               cout<<"5. Tampil"<<endl;
115.               cout<<"Pilih (1-3):";cin>>pil;
116.               switch(pil){
117.                 case 1:
118.                     cout<<"Masukkan Nilai :";cin>>x;
119.                     buatsimpul(x);
120.                     simpulawal();
121.                     insertkanan();
122.                     break;
123.                 case 2 :
124.                     cout<<"Masukkan Nilai :";cin>>x;
125.                     buatsimpul(x);
126.                     simpulawal();
127.                     insertkiri();
128.                     break;
129.                 case 3 :
130.                     cout<<"Masukkan Nilai :";cin>>x;
131.                     buatsimpul(x);
132.                     cout<<"Masukkan Nilai dicari :";cin>>y;
133.                     simpulawal();
134.                     inserttengah(y);
135.                     break;
136.                 case 4 :
137.                     deletekanan();
138.                     break;
139.                 case 5:
140.                     tampil();
141.                     break;
142.                 default :
```

```
143.                    cout<<"Salah";
144.                }
145.        }while(pil!=6);
146.    }
```

7.3. Latihan

1. Buatlah Double Linked List dengan data info biodata dengan atribut ID, nama, usia, dan kota_asal.
2. Tambahkan pencarian data berdasarkan ID.

7.4. Tugas

Buatlah program Double Linked List dengan fungsi operasi:

1. Update data info
2. Delete node

BAB 8
CIRCULAR DOUBLE LINKED LIST

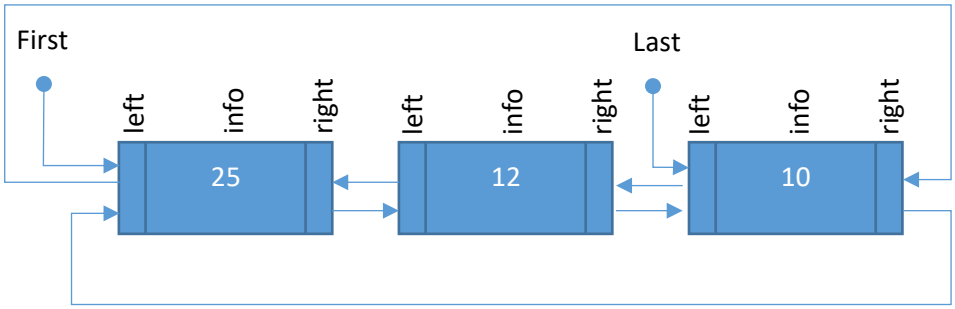
8.1. Tujuan Praktikum

Setelah melakukan praktikum ini, diharapkan mahasiswa mampu:

1. Memahami konsep struktur data *Circular Double Linked List*.
2. Memahami konsep operasi-operasi dasar pada *Circular Double Linked List.*
3. Membuat program dengan menggunakan *Circular Double Linked List*.

8.2. Teori Singkat

Circular Double Linked List adalah double linked list dimana pointer RIGHT simpul terakhir (paling kanan) menunjuk ke alamat simpul pertama (paling kiri), dan pointer LEFT simpul pertama (paling kiri) menunjuk ke alamat simpul terakhir (paling kanan), sehingga membuat efek melingkar baik searah jarum jam ataupun sebaliknya. Lebih jelasnya silahkan lihat gambar ilustrasi dibawah ini.



Contoh program:

```
1. #include <iostream>
2. #include <stdio.h>
3. #include <conio.h>
4. #include <ctype.h>
5. #include <stdlib.h>
6.
7. using namespace std;
8.
9. int x,y;
10.  int j=0;
11.
12.
13.  typedef struct node{
14.     int info;
15.     struct node *LEFT, *RIGHT;
16.  };
17.
18.  typedef struct node simpul;
19.  simpul *FIRST, *LAST, *P, *Q, *cetak;
20.
21.  void inisialisasi(){
```

```
22.     FIRST=NULL;
23.  }
24.
25.  void buatsimpul(int x){
26.     P=(simpul *)malloc(sizeof(simpul));
27.     if(P!=NULL){
28.        P->info=x;
29.     }
30.     else{
31.         cout<<"Gagal";
32.     }
33.  }
34.
35.  void simpulawal(){
36.     if(FIRST==NULL){
37.        FIRST=P;
38.        LAST=P;
39.        P->RIGHT=P;
40.        P->LEFT=P;
41.        j++;
42.     }
43.  }
44.
45.  void insertkanan(){
46.    if(LAST!=NULL){
47.       LAST->RIGHT=P;
48.       LAST=P;
49.       LAST->RIGHT=FIRST;
50.       FIRST->LEFT=LAST;
51.       j++;
52.    }
53.    else{
54.       cout<<"Isinya Kosong";
55.    }
56.  }
57.
58.  void insertkiri(){
59.    if(FIRST!=NULL){
60.       P->RIGHT=FIRST;
61.       FIRST=P;
62.       FIRST->LEFT=LAST;
63.       LAST->RIGHT=FIRST;
64.       j++;
65.    }
66.    else{
67.       cout<<"Isinya Kosong";
68.    }
69.  }
70.
71.  void inserttengah(int y){
72.      Q=FIRST;
73.      while(Q->info!=y){
74.          Q=Q->RIGHT;
75.      }
76.      if(Q==LAST){
77.          LAST->RIGHT=P;
78.          P->LEFT=LAST;
79.          LAST=P;
80.          P->RIGHT=NULL;
81.          j++;
```

```
82.       }
83.       else{
84.           P->RIGHT=Q->RIGHT;
85.           P->LEFT=Q->RIGHT;
86.           Q->RIGHT->LEFT=P;
87.           Q->RIGHT=P;
88.           j++;
89.       }
90.   }
91.
92.   void deletekanan(){
93.       LAST=LAST->LEFT;
94.       free(LAST->RIGHT);
95.       LAST->RIGHT=FIRST;
96.       FIRST->LEFT=LAST;
97.       j--;
98.   }
99.
100.  void deletekiri(){
101.      FIRST=FIRST->RIGHT;
102.      free(FIRST->LEFT);
103.      FIRST->LEFT=LAST;
104.      LAST->RIGHT=FIRST;
105.      j--;
106.  }
107.
108.  void deletetengah(int y){
109.      Q=FIRST;
110.      while(Q->info!=y){
111.          Q=Q->RIGHT;
112.      }
113.      Q->RIGHT=Q->RIGHT->RIGHT;
114.      free(Q->RIGHT->LEFT);
115.      Q->RIGHT->LEFT=Q;
116.      j--;
117.  }
118.
119.  void tampil(){
120.    cetak=FIRST;
121.    for(int k=0;k<j-1;k++){
122.        cout<<cetak->info<<" ";
123.        cetak=cetak->RIGHT;
124.    }
125.  }
126.
127.  int main(){
128.      int pil;
129.      inisialisasi();
130.      do{
131.              //system("cls");
132.              cout<<endl<<endl<<"1. Insert Kanan"<<endl;
133.              cout<<"2. Insert Kiri"<<endl;
134.              cout<<"3. Insert Tengah"<<endl;
135.              cout<<"4. delete kanan"<<endl;
136.              cout<<"5. delete kiri"<<endl;
137.              cout<<"6. delete tengah"<<endl;
138.              cout<<"7. Tampil"<<endl;
139.              cout<<"Pilih (1-3):";cin>>pil;
140.              switch(pil){
141.                case 1:
```

```
142.                    cout<<"Masukkan Nilai :";cin>>x;
143.                    buatsimpul(x);
144.                    simpulawal();
145.                    insertkanan();
146.                    break;
147.                case 2 :
148.                    cout<<"Masukkan Nilai :";cin>>x;
149.                    buatsimpul(x);
150.                    simpulawal();
151.                    insertkiri();
152.                    break;
153.                case 3 :
154.                    cout<<"Masukkan Nilai :";cin>>x;
155.                    buatsimpul(x);
156.                    cout<<"Masukkan Nilai dicari :";cin>>y;
157.                    simpulawal();
158.                    inserttengah(y);
159.                    break;
160.                case 4 :
161.                    deletekanan();
162.                    break;
163.                case 5 :
164.                    deletekiri();
165.                    break;
166.                case 6 :
167.                    cout<<"Cari Nilai Tertentu";cin>>y;
168.                    deletetengah(y);
169.                    break;
170.                case 7:
171.                    tampil();
172.                    break;
173.                default :
174.                  cout<<"Salah";
175.             }
176.     }while(pil!=6);
177. }
```

8.3. Latihan

Buatlah program dengan menggunakan circular double linked list yang mempunyai fungsi:

1. *Searching* (pencarian berdasarkan info yang unik)
2. *Update* (perubahan info berdasarkan info yang unik)
3. *insert before node* (penambahan simpul pada posisi sebelum simpul yang dicari).

8.4. Tugas

1. Buatlah program ‘toko buku’ sederhana dengan menggunakan circular double linked list dengan atribut sebagai berikut: no (int unik), judul (string), penulis (string), harga (float). Lengkapi dengan fitur insert, search, dan update.

BAB 9
STACK dan QUEUE dengan LINKED LIST

9.1. Tujuan Praktikum

Setelah melakukan praktikum ini, diharapkan mahasiswa mampu:

1. Menerapkan konsep tumpukan (*stack*) dengan Single Linked List.
2. Menerapkan konsep antrian (*queue*) dengan Double Linked List.

9.2. Teori Singkat

9.2.1. STACK

Konsep tumpukan (*stack*) sudah pernah dibahas pada BAB 4. Kalau pada praktikum sebelumnya menggunakan *Array*, maka dalam praktikum kali ini, stack diimplementasikan menggunakan *Single Linked List* sebagai struktur dasar. Ada dua fungsi dasar yang harus diimplementasikan pada sebuah stack yaitu:

- Push: memasukkan data baru ke dalam stack
- Pop: mengambil data paling atas dari stack

Selain fungsi diatas, ada beberapa fungsi tambahan yang dapat diimplementasikan, misalnya:

- Peek: mengintip/melihat data yang berada di posisi paling atas (bukan mengambil)
- IsEmpty: melihat apakah stack kosong atau tidak
- IsFull: melihat apakah stack penuh atau tidak
- Dan lain-lain

Berikut adalah contoh deklarasi dan implementasi stack, serta pemanfaatan stack di dalam program:

```
1. #include <stdio.h>
2. #include <stdlib.h>
3. #include <iostream>
4. using namespace std;
5. int isEmpty();
6.
7. struct Node {
8.   int data;
9.   struct Node* link;
10.  };
11.  struct Node* top;
12.
13.  void push(int data)
14.  {
15.          struct Node* temp;
16.          temp = (struct Node*)malloc(sizeof(struct Node));
17.          if (!temp) {
18.                  cout<<"Heap Underflow"<<endl;
```

```
19.                 exit(1);
20.         }
21.         temp->data = data;
22.         temp->link = top;
23.         top = temp;
24. }
25.
26. int isEmpty(Node* top)
27. {
28.         return top == NULL;
29. }
30.
31. int peek()
32. {
33.         if (!isEmpty(top))
34.                 return top->data;
35.         else
36.                 exit(EXIT_FAILURE);
37. }
38.
39. void pop()
40. {
41.         struct Node* temp;
42.         if (top == NULL) {
43.                 cout<<"Stack Underflow"<<endl;
44.                 exit(1);
45.         }
46.         else {
47.                 temp = top;
48.                 top = top->link;
49.                 temp->link = NULL;
50.                 free(temp);
51.         }
52. }
53.
54. void display()
55. {
56.         struct Node* temp;
57.         if (top == NULL) {
58.                 cout<<"Stack Underflow"<<endl;
59.                 exit(1);
60.         }
61.         else {
62.                 temp = top;
63.                 while (temp != NULL) {
64.                         cout<<"["<<temp->data<<"]";
65.                         temp = temp->link;
66.                 }
67.                 cout << endl;
68.         }
69. }
70.
71. int main(void)
72. {
73.
74.         int data,pilih;
75.         while(true){
76.          cout<<"Menu untuk program stack dengan single
    linkedlist"<<endl;
77.          cout<<"1 untuk memasukkan data(push)"<<endl;
```

```
78.           cout<<"2 untuk mengeluarkan data(pop)"<<endl;
79.           cout<<"3 untuk menampilkan semua data(display)"<<endl;
80.           cout<<"4 untuk menampilkan data teratas(peek)"<<endl;
81.           cout<<"0 untuk keluar program"<<endl;
82.           cin>>pilih;
83.
84.           switch(pilih){
85.           case 1:
86.               cout<<"Masukkan data (int):";
87.               cin>>data;
88.               push(data);
89.               break;
90.           case 2:
91.               cout<<"Data dikeluarkan"<<endl;
92.               pop();
93.               break;
94.           case 3:
95.               cout<<"Data:"<<endl;
96.               display();
97.               break;
98.           case 4:
99.               cout<<"Data teratas:"<<peek()<<endl;
100.              break;
101.          default:
102.              exit(0);
103.              break;
104.          }
105.            display();
106.          }
107.
108.          return 0;
109. }
```

9.2.2. QUEUE

Konsep Antrian (*queue*) telah dibahas pada BAB 5. Pada praktikum sebelumnya queue telah diimplementasikan dengan struktur data Array. Adapun beberapa karakteristik penting dari antrian adalah sebagai berikut:

1. Elemen antrian yaitu data yang tersimpan didalam antrian.
2. Front/head (penunjuk elemen terdepan dari antrian).
3. Rear/tail (penunjuk elemen terakhir dari antrian).
4. Count (jumlah elemen pada antrian)
5. Kondisi penuh, yaitu kondisi dimana antrian mencapai kapasitas maksimum antrian. Pada kondisi ini tidak bisa menambahkan pada antrian lagi, karena akan menyebabkan *Overflow.*
6. Kondisi kosong, yaitu kondisi dimana tidak ada elemen pada antrian sehingga tidak memungkinkan dilakukan pengambilan elemen dari antrian. Bila dipaksakan untuk mengambil akan menyebabkan *overflow*.

7. Enqueue, yaitu proses menambahkan elemen ke dalam antrian.
8. Dequeue, yaitu proses pengambilan elemen dari antrian.

Pada praktikum ini antrian yang akan kita implementasikan adalah antrian dalam bentuk *Double Linked List*.

Berikut adalah contoh deklarasi dan implementasi *queue*, serta pemanfaatan stack di dalam program:

```
1. #include <bits/stdc++.h>
2. using namespace std;
3.
4. class QNode
5. {
6.   public:
7.   int key;
8.   QNode *next;
9. };
10.
11.  class Queue
12.  {
13.          public:
14.          QNode *front, *rear;
15.  };
16.
17.  QNode* newNode(int k)
18.  {
19.          QNode *temp = new QNode();
20.          temp->key = k;
21.          temp->next = NULL;
22.          return temp;
23.  }
24.
25.  Queue *createQueue()
26.  {
27.          Queue *q = new Queue();
28.          q->front = q->rear = NULL;
29.          return q;
30.  }
31.
32.  void enQueue(Queue *q, int k)
33.  {
34.          QNode *temp = newNode(k);
35.
36.          if (q->rear == NULL)
37.          {
38.          q->front = q->rear = temp;
39.          return;
40.          }
41.
42.          q->rear->next = temp;
43.          q->rear = temp;
44.  }
45.
46.  QNode *deQueue(Queue *q)
47.  {
48.          if (q->front == NULL)
49.          return NULL;
```

```
50.
51.         QNode *temp = q->front;
52.         q->front = q->front->next;
53.
54.         if (q->front == NULL)
55.         q->rear = NULL;
56.         return temp;
57.  }
58.
59.  void display(Queue* q)
60.  {
61.      QNode* temp;
62.      if (q->front == NULL) {
63.                  cout<<"Stack Underflow"<<endl;
64.                  exit(1);
65.          }
66.          else {
67.                  temp = q->front;
68.                  while (temp != NULL) {
69.                          cout<<"["<<temp->key<<"]";
70.                          temp = temp->next;
71.                  }
72.                  cout << endl;
73.          }
74.  }
75.
76.  int main()
77.  {
78.      Queue *q = createQueue();
79.      int data,pilih;
80.          while(true){
81.          cout<<"Menu untuk program queue dengan double
   linkedlist"<<endl;
82.          cout<<"1 untuk memasukkan data(enQueue)"<<endl;
83.          cout<<"2 untuk mengeluarkan data(deQueue)"<<endl;
84.          cout<<"3 untuk menampilkan semua data(display)"<<endl;
85.          cout<<"0 untuk keluar program"<<endl;
86.          cin>>pilih;
87.
88.          switch(pilih){
89.          case 1:
90.              cout<<"Masukkan data (int):";
91.              cin>>data;
92.              enQueue(q,data);
93.              break;
94.          case 2:
95.              cout<<"Data dikeluarkan"<<endl;
96.              deQueue(q);
97.              break;
98.          case 3:
99.              cout<<"Data:"<<endl;
100.             display(q);
101.             break;
102.         default:
103.             exit(0);
104.             break;
105.         }
106.           display(q);
107.         }
108.
```

```
109.         return 0;
110. }
```

9.3. Latihan

Buatlah program stack dengan menggunakan single linked list dan queue dengan menggunakan double linked list yang mempunyai tambahan fungsi:

1. *Searching* (pencarian berdasarkan info terletak pada urutan data berapa)
2. *Update* (perubahan info berdasarkan info yang unik)

9.4. Tugas

1. Buatlah program stack dengan double linked list
2. Buatlah program queue dengan single linked list

Sumber: https://www.geeksforgeeks.org

BAB 10
Binary Search

10.1. Tujuan Praktikum

Setelah melakukan praktikum ini, diharapkan mahasiswa mampu:

1. Memahami penggunaan algoritma Binary Search.
2. Menerapkan algoritma Binary Search.

10.2. Teori Singkat

Algoritma Binary Search yaitu algoritma pencarian dengan cara membagi baris data array menjadi dua bagian setiap melakukan proses pencarian data. Algoritma ini memiliki karakteristik sebagai berikut:

1. Memperkecil proses pembandingan antara data yang dicari dengan tabel/baris data.
2. Beban komputasi lebih kecil karena pencarian dilakukan di depan, tengah dan belakang.
3. Prinsip dasarnya adalah melakukan pembagian ruang pencarian yang diulang terus menerus hingga data ditemukan atau sampai ruang pencarian tidak bisa dibagi lagi.
4. Syarat utama dari algoritma ini yaitu data harus sudah diurutkan lebih dahulu.

Adapun algoritma Binary Search adalah sebagai berikut:

1. Ambil posisi awal (0) dan akhir (N-1) dan tengah ((awal+akhir)/2).
2. Bandingkan data yang dicari dengan data tengah:
   a. Jika sama, maka data ditemukan
   b. Jika lebih kecil, maka proses tetap dilakukan tetapi dengan akhir=tengah-1.
   c. Jika lebih besar, maka proses tetap dilakukan tetapi dengan awal=tengah+1.
3. Ulang proses no. 2 sampai data ditemukan atau tidak ditemukan.
4. Jika data awal lebih besar dari pada data akhir maka proses pencarian dihentikan dan data tidak ketemu.

Contoh program:

```
1. //recursive Binary Search
2. #include <iostream>
3. using namespace std;
4.
5. /*fungsi binary search dengan parameter data arrary, index ke-1,
   index ke-n, data yg dicari */
6. int binarySearch(int arr[], int l, int r, int x)
7. {
8.   if (r >= l) {
9.           int mid = l + (r - l) / 2;
10.
11.                  // jika data tengah adalah data yang dicari
12.                  if (arr[mid] == x)
13.                          return mid;
14.
15.                  // jika data yang dicari adalah kurang dari data
   tengah
16.                  if (arr[mid] > x)
17.                          return binarySearch(arr, l, mid - 1, x);
```

```
18.
19.                 // lainnya jika data yang dicari lebih dari data
    tengah
20.                 return binarySearch(arr, mid + 1, r, x);
21.         }
22.
23.         // We reach here when element is not
24.         // present in array
25.         return -1;
26.  }
27.
28.  int main(void)
29.  {
30.         int arr[] = { 2, 3, 4, 10, 40 };
31.         int x = 10;
32.         int n = sizeof(arr) / sizeof(arr[0]);
33.         int result = binarySearch(arr, 0, n - 1, x);
34.         (result == -1) ? cout << "Data tidak ditemukan"
35.                                 : cout << "Data ditemukan di index
    ke-" << result;
36.         return 0;
37.  }
```

10.3. Latihan

1. Buatlah program binary search dengan menu input data array dan input data yang akan dicari.

10.4. Tugas

1. Tambahkan program pengurutan sederhana dibawah ini sebelum memanggil fungsi pencarian Binary:

```
1. for(i=0;i<sizeof(arr);i++)
2.     for(i=0;i<sizeof(arr);i++)
3.     {
4.         if(arr[i]>arr[i+1])
5.         {
6.             temp=arr[i];
7.             arr[i]=arr[i+1];
8.             arr[i+1]=temp;
9.         }
10.
11.     }
```

dan Cobalah dengan data yang tidak terurut.

BAB 11
Interpolation Search

11.1. Tujuan Praktikum

Dengan melakukan praktikum ini maka diharapkan mahasiswa mampu:

1. Memahami algoritma Interpolation Search.
2. Menerapkan algoritma Interpolation Search untuk pencarian data.

11.2. Teori Singkat

Algoritma Interpolation Search merupakan metode pencarian yang bisa dianalogikan sebagai cara kita mencari kata pada suatu kamus. Dimana ketika mulai mencari, kita tidak memulai pencarian dari halaman awal, tetapi membuka 1/3 atau 2/3 kamus terlebih dahulu kemudian melihat huruf awal dari data yang dicari dengan membandingkan dengan huruf awal pada halaman yang dibuka. Tentunya kamus berisi daftar kata yang sudah terurut sesuai alfabet. Metode pencarian tersebut dirumuskan dalam bentuk matematis sebagai berikut:

$$Position = low + \frac{Key - Data[low]}{Data[high] - Data[low]} x\ (high - low)$$

Jika data[posisi] > data yang dicari, high = pos – 1

Jika data[posisi] < data yang dicari, high = pos + 1

Keterangan:

Position = posisi data yang dicari.

Key = Data yang dicari.

Low = index awal data.

High = index akhir data.

Data[] = array data.

Algoritma:

1. Dalam iterasi (loop), lakukan perhitungan dengan rumus interpolasi di atas untuk menentukan posisi.
2. Jika data yang dicari (key) sama dengan data pada posisi (Data[position]), maka kembalikan index posisi dan keluar iterasi.
3. Jika data yang dicari (key) lebih kecil dari data pada posisi (Data[position]), maka hitung dengan rumus interpolasi dengan index akhir data adalah posisi – 1.
4. Jika data yang dicari (key) lebih besar dari data pada posisi (Data[position]), maka hitung dengan rumus interpolasi dengan index awal data adalah posisi + 1.

5. Lakukan langkah 2 – 4 sampai ketemu data yang dicari. Jika sampai index akhir data sama dengan index awal data maka data tidak ditemukan dan program selesai.

Contoh Program:

```
1. // C++ program to implement interpolation search
2. #include<bits/stdc++.h>
3. using namespace std;
4.
5. /*
6. fungsi interpolation search
7. parameter:
8. arr = data
9. n = banyak data
10.  x = data yang dicari
11.  */
12.  int interpolationSearch(int arr[], int n, int x)
13.  {
14.          // menentukan index awal dan index akhir
15.          int lo = 0, hi = (n - 1);
16.
17.          // Selama data sudah diurutkan dan data yang dicari berada
   di antara index awal dan akhir
18.          while (lo <= hi && x >= arr[lo] && x <= arr[hi])
19.          {
20.              // Jika index awal sama dengan index akhir
21.                  if (lo == hi)
22.                  {
23.                      //jika data yang dicari berada pada index awal
   maka kembalikan index awal
24.                          if (arr[lo] == x) return lo;
25.                          //jika tidak maka tidak ketemu kembalikan -1
26.                          return -1;
27.                  }
28.                  // menentukan posisi dengan rumus interpolasi
29.                  int pos = lo + (((double)(hi - lo) /
30.                          (arr[hi] - arr[lo])) * (x - arr[lo]));
31.
32.                  // jika data ketemu
33.                  if (arr[pos] == x)
34.                          return pos;
35.
36.                  // jika data yang dicari lebih besar daripada data
   pada posisi tersebut
37.                  if (arr[pos] < x)
38.                          lo = pos + 1;
39.
40.                  // jika data yang dicari lebih kecil daripada data
   pada posisi tersebut
41.                  else
42.                          hi = pos - 1;
43.          }
44.          return -1;
45.  }
46.
47.  int main()
48.  {
```

```
49.         // Array Data
50.         int arr[] = {10, 12, 13, 16, 18, 19, 20, 21,
51.                                 22, 23, 24, 33, 35, 42, 47};
52.         int n = sizeof(arr)/sizeof(arr[0]);
53.
54.         int x = 18; // data yang dicari
55.         int index = interpolationSearch(arr, n, x);
56.
57.         // juka data ditemukan
58.         if (index != -1)
59.                 cout << "Data ditemukan pada index ke-" << index;
60.         else
61.                 cout << "Data tidak ditemukan.";
62.         return 0;
63.  }
```

Sumber: https://www.geeksforgeeks.org

11.3. Latihan

1. Buatlah program interpolation search dengan menu input data array dan input data yang akan dicari.

11.4. Tugas

1. Buatlah program pencarian dengan pilihan menu metode pencarian binary dan interpolation search dan tambahkan menu pengurutan data dengan menggunakan fungsi pengurutan sederhana (bublesort).

BAB 12
Sorting Algorithm

12.1. Tujuan Praktikum

Dengan melakukan praktikum ini maka diharapkan mahasiswa mampu:

1. Memahami algoritma Pengurutan.
2. Menerapkan algoritma Pengurutan untuk mengurutkan data.

12.2. Teori Singkat

12.2.1. Bubble Sort

Bubble Sort adalah algoritma pengurutan dengan cara melakukan penukaran data dengan tepat disebelahnya dan dilakukan terus-menerus hingga dalam satu iterasi tidak ada lagi data yang perlu diurutkan. Seperti namanya yaitu bubble (gelembung) yang berarti setiap data yang diperiksa akan menggelembung hingga berada pada tempat yang tepat.

Algoritma Bubble Sort ini merupakan salah satu algoritma pengurutan yang paling sederhana dalam penerapannya. Konsepnya adalah mengulang proses pembandingan antara tiap elemen dan menukarnya jika salah.

Contoh:

First Pass:
(**5 1** 4 2 8) –> (**1 5** 4 2 8), membandingkan dua elemen, dan tukar 5 > 1.
(1 **5 4** 2 8) –> (1 **4 5** 2 8), Tukar 5 > 4
(1 4 **5 2** 8) –> (1 4 **2 5** 8), Tukar 5 > 2
(1 4 2 **5 8**) –> (1 4 2 **5 8**), Tidak ditukar (8 > 5)
Second Pass:
(**1 4** 2 5 8) –> (**1 4** 2 5 8), Tidak ditukar
(1 **4 2** 5 8) –> (1 **2 4** 5 8), Tukar 4 > 2
(1 2 **4 5** 8) –> (1 2 **4 5** 8), Tidak ditukar
(1 2 4 **5 8**) –> (1 2 4 **5 8**), Tidak ditukar

Sekarang data sudah terurut, tetapi algoritma ini tidak tahu kalau sudah urut, maka perlu dilakukan iterasi lagi tanpa melakukan pertukaran untuk mengetahui bahwa data sudah terurut.
Third Pass:
(**1 2** 4 5 8) –> (**1 2** 4 5 8)
(1 **2 4** 5 8) –> (1 **2 4** 5 8)
(1 2 **4 5** 8) –> (1 2 **4 5** 8)
(1 2 4 **5 8**) –> (1 2 4 **5 8**)

Contoh Program:

```
1. #include <stdio.h>
2.
3. void swap(int *xp, int *yp)
4. {
5.   int temp = *xp;
```

```
6.    *xp = *yp;
7.    *yp = temp;
8.  }
9.
10.  //Fungsi bubble sort
11.  void bubbleSort(int arr[], int n)
12.  {
13.  int i, j;
14.  for (i = 0; i < n-1; i++)
15.
16.          for (j = 0; j < n-i-1; j++)
17.                  if (arr[j] > arr[j+1])
18.                          swap(&arr[j], &arr[j+1]);
19.  }
20.
21.  void printArray(int arr[], int size)
22.  {
23.          int i;
24.          for (i=0; i < size; i++)
25.                  printf("%d ", arr[i]);
26.          printf("n");
27.  }
28.
29.  int main()
30.  {
31.          int arr[] = {64, 34, 25, 12, 22, 11, 90};
32.          int n = sizeof(arr)/sizeof(arr[0]);
33.          bubbleSort(arr, n);
34.          printf("Sorted array: ");
35.          printArray(arr, n);
36.          return 0;
37.  }
```

12.2.2. Selection Sort

Algoritma pengurutan *Selection* adalah memilih elemen dengan nilai paling rendah dan menukar elemen yang dipilih dengan elemen ke-i. nilai dari I dimulai dari 1 ke n, dimana n adalah jumlah total elemen dikurangi 1.

Perhatikan contoh pengurutan dibawah ini:

Data	3	10	4	6	8	9	7	2	1	5
Pass 1	1	10	4	6	8	9	7	2	3	5
Pass 2	1	2	4	6	8	9	7	10	3	5
Pass 3	1	2	3	6	8	9	7	10	4	5
Pass 4	1	2	3	4	8	9	7	10	6	5
Pass 5	1	2	3	4	5	9	7	10	6	8
Pass 6	1	2	3	4	5	6	7	10	9	8
Pass 7	1	2	3	4	5	6	7	8	9	10

Contoh program:

```
1. #include <bits/stdc++.h>
2. using namespace std;
```

```
3.
4. void swap(int *xp, int *yp)
5. {
6.   int temp = *xp;
7.   *xp = *yp;
8.   *yp = temp;
9. }
10.
11.  void selectionSort(int arr[], int n)
12.  {
13.          int i, j, min_idx;
14.
15.          for (i = 0; i < n-1; i++)
16.          {
17.                  min_idx = i;
18.                  for (j = i+1; j < n; j++)
19.                  if (arr[j] < arr[min_idx])
20.                          min_idx = j;
21.
22.                  swap(&arr[min_idx], &arr[i]);
23.          }
24.  }
25.
26.  void printArray(int arr[], int size)
27.  {
28.          int i;
29.          for (i=0; i < size; i++)
30.                  cout << arr[i] << " ";
31.          cout << endl;
32.  }
33.
34.  int main()
35.  {
36.          int arr[] = {3, 10, 4, 6, 8, 9, 7, 2, 1, 5};
37.          int n = sizeof(arr)/sizeof(arr[0]);
38.          cout << "Unsorted array: \n";
39.          printArray(arr, n);
40.          cout << endl;
41.          selectionSort(arr, n);
42.          cout << "Sorted array: \n";
43.          printArray(arr, n);
44.          return 0;
45.  }
```

Sumber: https://www.geeksforgeeks.org

12.2.3. Insertion Sort

Satu lagi algoritma pengurutan yang sederhana adalah *insertion sort*. Algoritma ini dimulai dari data elemen paling kiri dibandingkan dengan elemen di sebelahnya. Kemudian elemen selanjutnya dibandingkan dengan elemen di sebelah kirinya apabila masih lebih kecil dari sebelah kirinya maka digeser. Begitu seterusnya ke kiri sampai tidak ada elemen yang lebih kecil lagi. Baru proses ke elemen selanjutnya.

Perhatikan contoh pengurutan di bawah ini:

Data	3	10	4	6	8	9	7	2	1	5
Pass 1	3	10	4	6	8	9	7	2	1	5
Pass 2	3	4	10	6	8	9	7	2	1	5
Pass 3	3	4	6	10	8	9	7	2	1	5
Pass 4	3	4	6	8	10	9	7	2	1	5
Pass 5	3	4	6	8	9	10	7	2	1	5
Pass 6	3	4	6	7	8	9	10	2	1	5
Pass 7	2	3	4	6	7	8	9	10	1	5
Pass 8	1	2	3	4	6	7	8	9	10	5
Pass 9	1	2	3	4	5	6	7	8	9	10

Contoh program:

```
1. #include <bits/stdc++.h>
2. using namespace std;
3.
4. void insertionSort(int arr[], int n)
5. {
6.   int i, key, j;
7.   for (i = 1; i < n; i++)
8.   {
9.           key = arr[i];
10.                  j = i - 1;
11.
12.                  while (j >= 0 && arr[j] > key)
13.                  {
14.                          arr[j + 1] = arr[j];
15.                          j = j - 1;
16.                  }
17.                  arr[j + 1] = key;
18.          }
19.  }
20.
21.  void printArray(int arr[], int n)
22.  {
23.          int i;
24.          for (i = 0; i < n; i++)
25.                  cout << arr[i] << " ";
26.          cout << endl;
27.  }
28.
29.  int main()
30.  {
31.          int arr[] = { 3, 10, 4, 6, 8, 9, 7, 2, 1, 5};
32.          int n = sizeof(arr) / sizeof(arr[0]);
33.          cout << "Sebelum diurutkan:" << endl;
34.      printArray(arr, n);
35.          insertionSort(arr, n);
36.          cout << "Setelah diurutkan:" << endl;
37.          printArray(arr, n);
38.
39.          return 0;
40.  }
```

Sumber: https://www.geeksforgeeks.org

12.3. Latihan

Buatlah program pengurutan dengan input data array dan input menu pilihan:

1. Data Sebelum Diurutkan
2. Bubble Sort
3. Selection Sort
4. Insertion Sort

12.4. Tugas

Buatlah program pengurutan seperti pada soal latihan, tetapi tampilkan proses pengurutannya setiap iterasi (pass). Seperti contoh pengurutan pada tabel di teori singkat.

BAB 13
Advanced Sorting Algorithm

13.1. Tujuan Praktikum

Dengan melakukan praktikum ini maka diharapkan mahasiswa mampu:

1. Memahami algoritma Pengurutan tingkat lanjut.
2. Menerapkan algoritma Pengurutan tingkat lanjut untuk mengurutkan data.

13.2. Teori Singkat

13.2.1. Shell Sort

Metode pengurutan ini dikembangkan oleh Donald L. Shell pada tahun 1959, oleh karena itu disebut Shell sort. Metode ini juga sering disebut dengan metode pertambahan menurun (*diminishing increment sort*). Pengurutan dilakukan dengan membandingkan suatu data dengan data lain yang memiliki jarak tertentu, sehingga membentuk sebuah sub-list. Kemudian dilakukan pertukaran posisi jika diperlukan. Jarak yang dipakai didasarkan pada *increment value* atau *sequence number k*. setiap sub-list berisi elemen ke-k dari kumpulan elemen yang asli.

Pemilihan squence number disarankan mula-mula data yang akan dibandingkan adalah N/2. Pada proses selanjutnya N/4 kemudian N/8 dan seterusnya hingga jarak yang digunakan adalah 1.

Contoh program:

```
1. #include <iostream>
2. using namespace std;
3.
4. /* fungsi shellSort */
5. int shellSort(int arr[], int n)
6. {
7.   // gap adalah jarak dimulai dari n/2
8.   for (int gap = n/2; gap > 0; gap /= 2)
9.   {
10.
11.                 for (int i = gap; i < n; i += 1)
12.                 {
13.                         int temp = arr[i];
14.
15.
16.                         int j;
17.                         for (j = i; j >= gap && arr[j - gap] > temp;
   j -= gap)
18.                                 arr[j] = arr[j - gap];
19.
20.
21.                         arr[j] = temp;
22.                 }
23.         }
24.         return 0;
25.  }
26.
27.  void printArray(int arr[], int n)
```

```
28.  {
29.          for (int i=0; i<n; i++)
30.                  cout << arr[i] << " ";
31.  }
32.
33.  int main()
34.  {
35.          int arr[] = {12, 34, 54, 2, 3}, i;
36.          int n = sizeof(arr)/sizeof(arr[0]);
37.
38.          cout << "Array before sorting: \n";
39.          printArray(arr, n);
40.
41.          shellSort(arr, n);
42.
43.          cout << "\nArray after sorting: \n";
44.          printArray(arr, n);
45.
46.          return 0;
47.  }
```

13.2.2. Quick Sort

Metode pengurutan Quick sort ini memiliki konsep membagi dan menyelesaikan atau sering disebut (devide and conquer). Proses metode ini dimulai dengan menentukan posisi pivot data, lalu membagi data array ke dalam dua sub-array berdasarkan pivot. Kemudian subarray diurutkan secara rekursif dengan memanggil fungsi quicksort.

Penentuan pivot ada beberapa cara yaitu:

1. Memilih pivot dari elemen pertama.
2. Memilih pivot dari elemen terakhir.
3. Memilih pivot secara acak.
4. Memilih pivot dari median (nilai tengah).

Fungsi yang utama pada metode ini adalah fungsi partition. Fungsi tersebut akan membagi array berdasarkan pivot yang telah ditentukan. Kemudian letakkan posisi pivot pada tempat yang tepat, yaitu data sebelah kiri pivot adalah data yang lebih kecil dari pivot dan sebelah kanan pivot adalah data yang lebih besar dari pivot. Berikut ini ditunjukkan pseudocode untuk quicksort:

```
/* low  --> Starting index,  high  --> Ending index */
quickSort(arr[], low, high)
{
    if (low < high)
    {
        /* pi is partitioning index, arr[pi] is now
           at right place */
        pi = partition(arr, low, high);

        quickSort(arr, low, pi - 1);  // Before pi
```

```
        quickSort(arr, pi + 1, high); // After pi
    }
}
```

```
partition (arr[], low, high)
{
    // pivot (Element to be placed at right position)
    pivot = arr[high];

    i = (low - 1)  // Index of smaller element

    for (j = low; j <= high- 1; j++)
    {
        // If current element is smaller than or
        // equal to pivot
        if (arr[j] <= pivot)
        {
            i++;    // increment index of smaller element
            swap arr[i] and arr[j]
        }
    }
    swap arr[i + 1] and arr[high])
    return (i + 1)
}
```

Contoh program:

```
1. #include <bits/stdc++.h>
2. using namespace std;
3.
4. // A utility function to swap two elements
5. void swap(int* a, int* b)
6. {
7.   int t = *a;
8.   *a = *b;
9.   *b = t;
10.  }
11.
12.  int partition (int arr[], int low, int high)
13.  {
```

```
14.         int pivot = arr[high]; // pivot
15.         int i = (low - 1); // Index of smaller element
16.
17.         for (int j = low; j <= high - 1; j++)
18.         {
19.                 // If current element is smaller than or
20.                 // equal to pivot
21.                 if (arr[j] <= pivot)
22.                 {
23.                         i++; // increment index of smaller element
24.                         swap(&arr[i], &arr[j]);
25.                 }
26.         }
27.         swap(&arr[i + 1], &arr[high]);
28.         return (i + 1);
29. }
30.
31. void quickSort(int arr[], int low, int high)
32. {
33.         if (low < high)
34.         {
35.                 /* pi is partitioning index, arr[p] is now
36.                 at right place */
37.                 int pi = partition(arr, low, high);
38.
39.                 // Separately sort elements before
40.                 // partition and after partition
41.                 quickSort(arr, low, pi - 1);
42.                 quickSort(arr, pi + 1, high);
43.         }
44. }
45.
46. /* Function to print an array */
47. void printArray(int arr[], int size)
48. {
49.         int i;
50.         for (i = 0; i < size; i++)
51.                 cout << arr[i] << " ";
52.         cout << endl;
53. }
54.
55. // Driver Code
56. int main()
57. {
58.         int arr[] = {10, 7, 8, 9, 1, 5};
59.         int n = sizeof(arr) / sizeof(arr[0]);
60.         cout << "Before sorted: \n" << endl;
61.         printArray(arr, n);
62.
63.         quickSort(arr, 0, n - 1);
64.         cout << "Sorted array: \n" << endl;
65.         printArray(arr, n);
66.         return 0;
67. }
```

13.2.3. Merge Sort

Metode pengurutan ini juga termasuk algoritma devide and conquer (membagi dan menyelesaikan). Metode ini dimulai dengan membagi array data menjadi dua hingga menjadi

subarray yang hanya berisi satu elemen. Menggabungkan solusi dari subarray dengan membandingkan terlebih dahulu dan meletakkan elemen yang terkecil di sebelah kiri.

Contoh program:

```
1. #include<stdlib.h>
2. #include<stdio.h>
3.
4. void merge(int arr[], int l, int m, int r)
5. {
6.   int i, j, k;
7.   int n1 = m - l + 1;
8.   int n2 = r - m;
9.
10.         /* create temp arrays */
11.         int L[n1], R[n2];
12.
13.         /* Copy data to temp arrays L[] and R[] */
14.         for (i = 0; i < n1; i++)
15.                 L[i] = arr[l + i];
16.         for (j = 0; j < n2; j++)
17.                 R[j] = arr[m + 1+ j];
18.
19.         /* Merge the temp arrays back into arr[l..r]*/
20.         i = 0; // Initial index of first subarray
21.         j = 0; // Initial index of second subarray
22.         k = l; // Initial index of merged subarray
23.         while (i < n1 && j < n2)
24.         {
25.                 if (L[i] <= R[j])
26.                 {
27.                         arr[k] = L[i];
28.                         i++;
29.                 }
30.                 else
31.                 {
32.                         arr[k] = R[j];
33.                         j++;
34.                 }
35.                 k++;
36.         }
37.
38.         while (i < n1)
39.         {
40.                 arr[k] = L[i];
41.                 i++;
42.                 k++;
43.         }
44.
45.         while (j < n2)
46.         {
47.                 arr[k] = R[j];
48.                 j++;
49.                 k++;
50.         }
51. }
52.
53. void mergeSort(int arr[], int l, int r)
54. {
```

```
55.         if (l < r)
56.         {
57.                 // Same as (l+r)/2, but avoids overflow for
58.                 // large l and h
59.                 int m = l+(r-l)/2;
60.
61.                 // Sort first and second halves
62.                 mergeSort(arr, l, m);
63.                 mergeSort(arr, m+1, r);
64.
65.                 merge(arr, l, m, r);
66.         }
67. }
68.
69. void printArray(int A[], int size)
70. {
71.         int i;
72.         for (i=0; i < size; i++)
73.                 printf("%d ", A[i]);
74.         printf("\n");
75. }
76.
77. /* Driver program to test above functions */
78. int main()
79. {
80.         int arr[] = {12, 11, 13, 5, 6, 7};
81.         int arr_size = sizeof(arr)/sizeof(arr[0]);
82.
83.         printf("Given array is \n");
84.         printArray(arr, arr_size);
85.
86.         mergeSort(arr, 0, arr_size - 1);
87.
88.         printf("\nSorted array is \n");
89.         printArray(arr, arr_size);
90.         return 0;
91. }
```

Sumber: https://www.geeksforgeeks.org

13.3. Latihan

Buatlah program pengurutan dengan input data array dan input menu pilihan:

1. Data Sebelum Diurutkan
2. Bubble Sort
3. Selection Sort
4. Insertion Sort
5. Shell Sort
6. Quick Sort
7. Merge Sort

13.4. Tugas

Buatlah program pengurutan seperti pada soal latihan, tetapi tampilkan proses pengurutannya setiap iterasi (pass).