MAKALAH
STACK

Disusun oleh :

Nama	: L Hafid Alkhair
NIM	: 2023903430060
Kelas	: TRKJ 1.C
Jurusan	: Teknologi Informasi dan Komputer
Program Studi	: Teknologi Rekayasa Komputer Jaringan
Dosen Pengajar	: Indrawati, SST. MT

JURUSAN TEKNOLOGI INFORMASI KOMPUTER

PRODI TEKNOLOGI REKAYASA KOMPUTER JARINGAN

POLITEKNIK NEGERI LHOKSEUMAWE

TAHUN AJARAN 2023/2024

DAFTAR ISI

BAB I
PEMBAHASAN

A. Pengertian Stack

Stack adalah struktur data yang penting dalam ilmu komputer dan pemrograman, termasuk dalam bahasa pemrograman C. Dalam konteks C, stack adalah tumpukan memori yang digunakan untuk menyimpan data secara terstruktur. Dalam tumpukan (stack), elemen-elemen data disusun dalam urutan linier, namun hanya elemen teratas yang dapat diakses atau dihapus. Konsep ini sering digunakan dalam pemrograman untuk mengelola pemanggilan fungsi, ekspresi matematika, pemrosesan bahasa alami, dan banyak lagi.

Stack dalam bahasa C adalah salah satu struktur data yang paling umum digunakan. Ini adalah tumpukan linier yang mengikuti prinsip "Last In, First Out" (LIFO), yang berarti elemen terakhir yang dimasukkan adalah elemen pertama yang keluar. Dalam bahasa C, stack biasanya diimplementasikan menggunakan larik (array) atau linked list. Stack menyediakan dua operasi utama: push, yang menambahkan elemen ke tumpukan, dan pop, yang menghapus elemen teratas dari tumpukan.

Keunggulan utama dari penggunaan stack adalah efisiensi waktu operasinya. Penambahan dan penghapusan elemen dari stack memerlukan waktu konstan, yaitu O(1). Namun, ukuran stack biasanya terbatas oleh ukuran memori yang dialokasikan untuknya.

Salah satu contoh penggunaan stack dalam bahasa C adalah evaluasi ekspresi aritmatika. Saat mengevaluasi ekspresi matematika kompleks, stack dapat digunakan untuk menyimpan operator dan operand sementara ekspresi dievaluasi. Jika ada tanda kurung dalam ekspresi, stack juga dapat digunakan untuk memastikan urutan evaluasi yang benar.

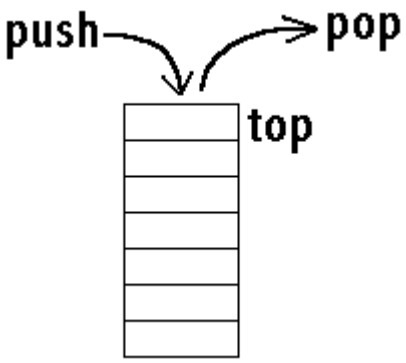
Selain itu, stack juga digunakan dalam pemrograman untuk mengelola pemanggilan fungsi. Saat sebuah fungsi dipanggil, alamat pengembalian dan variabel lokal disimpan di dalam stack. Setelah fungsi selesai dieksekusi, nilai-nilai ini diambil dari stack sehingga program dapat melanjutkan eksekusi dari titik yang benar.

Dalam pengembangan perangkat lunak, stack juga digunakan untuk melacak pemanggilan fungsi dan memastikan bahwa program berjalan dengan benar. Dengan memahami konsep stack, para pengembang dapat mengoptimalkan kode mereka dan memastikan bahwa aplikasi mereka bekerja secara efisien dan dapat diandalkan.

Dalam bahasa C, implementasi stack menggunakan array melibatkan deklarasi array, variabel indeks penunjuk atas stack (biasanya disebut "top"), dan fungsi-fungsi seperti push() untuk menambahkan elemen ke stack, pop() untuk menghapus elemen dari stack, dan isEmpty() untuk memeriksa apakah stack kosong atau tidak. Implementasi stack menggunakan linked list melibatkan deklarasi struktur node yang memiliki dua anggota: data dan penunjuk ke node berikutnya. Fungsi-fungsi yang sama juga dapat diterapkan pada implementasi linked list stack.

Dalam pengembangan perangkat lunak modern, stack juga digunakan dalam manajemen memori. Saat aplikasi berjalan, ia menggunakan tumpukan memori untuk menyimpan variabel lokal dan informasi pemanggilan fungsi. Melalui manajemen memori yang efisien menggunakan stack, aplikasi dapat mengoptimalkan penggunaan memori dan menghindari kebocoran memori yang dapat menyebabkan kinerja buruk atau crash aplikasi.

Dalam penutup, stack adalah struktur data yang sangat penting dalam pemrograman dan memainkan peran krusial dalam berbagai aspek pengembangan perangkat lunak. Dengan pemahaman yang baik tentang konsep stack dan implementasinya dalam bahasa C, para pengembang dapat menulis kode yang efisien, andal, dan mudah dimengerti. Dengan menguasai stack, para pengembang dapat meningkatkan keterampilan pemrograman mereka dan membangun aplikasi yang kompleks dan andal.



Gambar 1.1 Ilustrasi Stack

B. Deklarasi Stack

Dalam dunia pemrograman, stack adalah struktur data yang vital. Dalam bahasa C, deklarasi stack adalah pintu gerbang menuju berbagai aplikasi yang kompleks. Stack adalah kumpulan elemen yang mengikuti prinsip LIFO (Last In, First Out), yang berarti elemen terakhir yang dimasukkan adalah yang pertama kali diambil. Memahami cara mendeklarasikan stack dalam bahasa C adalah keterampilan yang esensial untuk pengembangan perangkat lunak yang efisien dan andal.

1. Deklarasi Dasar

Di bahasa C, stack bisa diimplementasikan menggunakan array atau linked list. Untuk mendeklarasikan stack menggunakan array, kita membutuhkan tiga elemen penting: larik untuk menyimpan data, variabel penunjuk atas (umumnya disebut 'top'), dan ukuran stack (maksimal elemen yang dapat ditampung). Deklarasi stack menggunakan array di C mungkin terlihat seperti ini:

```
1  #define MAX_SIZE 100 // Ukuran maksimal stack
2  int stack[MAX_SIZE]; // Deklarasi stack menggunakan array
3  int top = -1; // Variabel penunjuk atas stack
```

Gambar 1.2 Deklarasi Stack

Dalam contoh di atas, `MAX_SIZE` mendefinisikan ukuran maksimal stack yang dapat menampung 100 elemen. Variabel `stack` adalah larik integer yang menyimpan elemen-elemen stack. Variabel `top` adalah penunjuk atas stack yang awalnya diatur ke -1, menunjukkan stack kosong.

2. **Operasi Dasar:**

Operasi dasar pada stack melibatkan push (menambahkan elemen ke stack) dan pop (menghapus elemen dari stack). Berikut adalah contoh implementasi operasi push dan pop:

```
1  void push(int data) {
2      if (top == MAX_SIZE - 1) {
3          printf("Stack overflow! Tidak dapat menambahkan elemen.\n");
4      } else {
5          stack[++top] = data;
6          printf("%d ditambahkan ke stack.\n", data);
7      }
8  }
9
10 void pop() {
11     if (top == -1) {
12         printf("Stack kosong! Tidak dapat menghapus elemen.\n");
13     } else {
14         printf("%d dihapus dari stack.\n", stack[top--]);
15     }
16 }
```

Gambar 1.3 Implementasi Operasi Push

Fungsi `push` menambahkan elemen ke stack dan meningkatkan nilai `top` sebelum menyimpan data. Fungsi `pop` menghapus elemen dari stack dengan mengembalikan nilai stack pada posisi `top` sebelum mengurangkan nilai `top`.

3. **Implementasi Menggunakan Linked List:**

Selain menggunakan array, stack juga dapat diimplementasikan menggunakan linked list. Implementasi ini memungkinkan stack untuk dinamis memperluas ukurannya. Deklarasi stack menggunakan linked list mungkin terlihat seperti ini:

```
struct Node {
    int data;
    struct Node* next;
};

struct Node* top = NULL;
// Variabel penunjuk atas stack (head linked list)
```

Gambar 1.4 Deklarasi stack menggunakan linked list

Dalam implementasi ini, `struct Node` adalah struktur data yang berisi elemen stack dan penunjuk ke node berikutnya dalam linked list. Variabel `top` adalah penunjuk atas stack, mengacu pada node pertama dalam linked list.

4. **Kesimpulan:**

Deklarasi stack dalam bahasa C membuka pintu ke dunia pemrograman yang penuh dengan keajaiban struktur data. Dengan pemahaman mendalam tentang konsep stack dan penerapannya dalam bahasa C, pengembang dapat membuat aplikasi yang kompleks, efisien, dan andal. Dengan kemampuan untuk mendeklarasikan stack, para pengembang dapat memperoleh kontrol atas pemrosesan data dalam cara yang sistematis dan terorganisir. Dalam pemrograman yang cerdas, stack adalah teman setia yang membawa pengembang menuju pencapaian luar biasa.

C. Operasi Dasar pada Stack

Dalam dunia pemrograman, stack adalah struktur data yang sangat penting. Diimplementasikan dalam berbagai bahasa pemrograman, operasi dasar pada stack menjadi landasan untuk pemahaman struktur data yang efisien dan pengembangan aplikasi yang handal. Saat menggunakan bahasa C, pemahaman mendalam tentang operasi dasar pada stack adalah keterampilan yang esensial.

1. Push: Menambahkan Elemen ke Stack

Operasi push adalah cara untuk menambahkan elemen baru ke stack. Dalam bahasa C, push dilakukan dengan meningkatkan variabel penunjuk atas stack dan menetapkan nilai elemen ke posisi yang ditunjuk oleh variabel tersebut. Sebelum menambahkan elemen, penting untuk memeriksa apakah stack sudah penuh atau tidak. Ini memastikan bahwa tidak ada operasi push yang dilakukan ketika stack sudah mencapai kapasitas maksimalnya.

Contoh implementasi operasi push dalam bahasa C:

```c
void push(int stack[], int *top, int maxSize, int element) {
    if (*top == maxSize - 1) {
        printf("Stack penuh! Tidak dapat menambahkan elemen.\n");
    } else {
        stack[++(*top)] = element;
        printf("%d ditambahkan ke stack.\n", element);
    }
}
```

Gambar 1.5 Operasi Push

2. Pop: Menghapus Elemen dari Stack

Operasi pop adalah cara untuk menghapus elemen teratas dari stack. Ini melibatkan mengambil nilai elemen pada posisi yang ditunjuk oleh variabel penunjuk atas stack dan kemudian mengurangkan nilai variabel tersebut. Sebelum menghapus elemen, penting untuk memeriksa apakah stack sudah kosong atau tidak. Hal ini mencegah operasi pop yang dilakukan ketika stack sudah kosong.

Contoh implementasi operasi pop dalam bahasa C:

```c
int pop(int stack[], int *top) {
    if (*top == -1) {
        printf("Stack kosong! Tidak ada elemen yang dapat dihapus.\n");
        return -1; // Nilai khusus untuk menandai stack kosong
    } else {
        int element = stack[(*top)--];
        printf("%d dihapus dari stack.\n", element);
        return element;
    }
}
```

Gambar 1.6 Operasi Pop

3. Peek: Melihat Elemen Teratas Tanpa Menghapusnya

Operasi peek memungkinkan pengguna melihat nilai elemen teratas dari stack tanpa menghapusnya. Dalam bahasa C, ini dilakukan dengan mengembalikan nilai elemen pada posisi yang ditunjuk oleh variabel penunjuk atas stack.

Contoh implementasi operasi peek dalam bahasa C:

```c
int peek(int stack[], int top) {
    if (top == -1) {
        printf("Stack kosong! Tidak ada elemen yang dapat dilihat.\n");
        return -1; // Nilai khusus untuk menandai stack kosong
    } else {
        printf("Elemen teratas dari stack adalah: %d\n", stack[top]);
        return stack[top];
    }
}
```

Gambar 1.7 Operasi peek

4. Cek Kosong atau Penuh

Dalam beberapa situasi, penting untuk memeriksa apakah stack kosong atau sudah mencapai kapasitas maksimalnya. Fungsi ini membantu memastikan operasi push dan pop hanya dilakukan saat diperlukan.

Contoh implementasi pemeriksaan stack kosong dan penuh dalam bahasa C:

```c
int isEmpty(int top) {
    return (top == -1);
}

int isFull(int top, int maxSize) {
    return (top == maxSize - 1);
}
```

Gambar 1.8 pemeriksaan stack kosong dan penuh

5. Kesimpulan:

Operasi dasar pada stack adalah fondasi yang kuat dalam pengembangan perangkat lunak. Dengan memahami cara melakukan push, pop, peek, serta cara memeriksa apakah stack kosong atau penuh, pengembang dapat merancang aplikasi yang efisien dan andal. Pengetahuan tentang operasi dasar pada stack membuka pintu menuju berbagai aplikasi, termasuk evaluasi ekspresi matematika, penanganan fungsi dalam bahasa pemrograman, dan manajemen memori. Dalam pengembangan perangkat lunak yang kompleks, pemahaman yang baik tentang operasi dasar pada stack adalah kunci keberhasilan.

D. Notasi Aritmatika (Infex, Prefix, dan Positifx)

Notasi aritmatik adalah cara yang berbeda untuk mengekspresikan operasi matematika seperti penjumlahan, pengurangan, perkalian, dan pembagian. Di dalam dunia pemrograman, pemahaman tentang notasi aritmatik sangat penting, terutama ketika mengekspresikan ekspresi matematika dalam bentuk yang bisa dihitung oleh komputer. Di antara berbagai jenis notasi aritmatik, infiks, prefiks, dan postfiks adalah yang paling umum. Dalam bahasa C, memahami konsep dan implementasi notasi aritmatik ini adalah langkah pertama menuju pengembangan algoritma yang efisien dan andal.

1. Infiks: Notasi Aritmatik Konvensional

Infiks adalah notasi aritmatik konvensional yang digunakan sehari-hari, di mana operator ditempatkan di antara operandnya. Sebagai contoh, ekspresi

3	+	4	*	5

3+4∗5 adalah contoh infiks, di mana penjumlahan ditempatkan di antara 3 dan hasil perkalian 4 dan 5.

Pada umumnya, ketika kita mengevaluasi ekspresi infiks, kita membutuhkan aturan prioritas operator (seperti perkalian dilakukan sebelum penjumlahan) dan tanda kurung untuk mengontrol urutan evaluasi.

2. **Prefiks (Notasi Polandia Awal): Operator Didahulukan**

Prefiks, juga dikenal sebagai notasi Polandia Awal, adalah bentuk notasi aritmatik di mana operator ditempatkan sebelum operandnya. Dalam ekspresi prefiks, urutan operasi matematika diindikasikan dengan sangat jelas. Misalnya, ekspresi infiks

3	+	4	*	5

Dalam Prefiks Menjadi

+3	*	45

Untuk mengevaluasi ekspresi prefiks, pengembang sering menggunakan struktur data stack. Operand ditempatkan dalam stack, dan ketika operator ditemui, dua operand teratas diambil dari stack dan hasil operasi ditempatkan kembali ke stack.

3. **Postfiks (Notasi Polandia Akhir): Operator Diposisikan Setelah Operand**

Postfiks, juga dikenal sebagai notasi Polandia Akhir, adalah bentuk notasi aritmatik di mana operator ditempatkan setelah operandnya. Dalam ekspresi postfiks, tidak ada ambiguitas dalam urutan operasi matematika. Contoh, ekspresi infiks

3	+	4	*	5

dalam postfiks menjadi

345	*	+

Evaluasi ekspresi postfiks juga melibatkan penggunaan stack. Saat operator ditemui, operand-operand yang sesuai diambil dari stack, hasil operasi ditempatkan kembali ke stack.

E. Kenapa Bisa Tersusun Menjadi Stack

Dalam bahasa pemrograman C, Anda bisa menggunakan array atau linked list untuk membuat dan mengelola stack. Alasannya adalah karena sifat dasar dari array dan linked list memungkinkan struktur data tumpukan (stack) dapat direpresentasikan dengan baik.

Berikut adalah alasan kenapa Anda bisa membuat stack menggunakan array atau linked list di bahasa C:

1. Array

Akses Langsung ke Memori:Array adalah struktur data yang menggunakan alokasi memori kontigus. Dalam stack, elemen-elemen disimpan dalam urutan yang sama dalam memori. Ini memungkinkan akses cepat ke elemen teratas stack dengan menggunakan indeks array.

- Efisiensi Penyimpanan:Penggunaan array memastikan penggunaan memori yang efisien karena setiap elemen membutuhkan ruang memori yang tetap.
- Operasi Cepat:Operasi push dan pop pada stack yang diimplementasikan menggunakan array memiliki kompleksitas waktu O(1) karena elemen-elemen stack dapat diakses langsung melalui indeks array.

2. Linked List

Alokasi Dinamis:Linked list menggunakan alokasi memori dinamis, yang berarti elemen-elemen dapat ditambahkan dan dihapus dari stack dengan mudah tanpa perlu menentukan ukuran stack secara tetap.

- Struktur Data Fleksibel: Linked list dapat diubah ukurannya sesuai kebutuhan. Ini memungkinkan stack untuk menyesuaikan diri dengan jumlah elemen yang harus disimpan, yang tidak harus diketahui sebelumnya.
- Operasi Fleksibel: Operasi push dan pop pada stack yang diimplementasikan menggunakan linked list membutuhkan kompleksitas waktu O(1). Pada linked list, operasi ini melibatkan penambahan atau penghapusan node di awal linked list, yang dapat dilakukan dengan cepat.

Kedua metode ini memberikan fleksibilitas dan kecepatan dalam mengelola stack, yang sangat penting dalam pengembangan perangkat lunak. Programmers dapat memilih metode yang paling cocok tergantung pada kebutuhan spesifik aplikasi mereka. Dengan menggunakan konsep dasar array atau linked list, stack dapat diimplementasikan secara efisien dan mudah dimengerti dalam bahasa pemrograman C.

F. Inisialisasi dan Penghapusan Stack

Dalam bahasa pemrograman C, stack dapat diimplementasikan menggunakan array atau linked list. Berikut adalah contoh cara menginisialisasi dan menghapus stack menggunakan array:

Menginisialisasi Stack menggunakan Array:

```c
#define MAX_SIZE 100 // Jumlah maksimum elemen dalam stack

struct Stack {
    int items[MAX_SIZE];
    int top; // Indeks elemen teratas stack
};

void initializeStack(struct Stack* stack) {
    stack->top = -1; // Menginisialisasi stack dengan top -1 menandakan stack kosong
}
```

Gambar 1.8 Inisialisasi Stack Menggunakan Array

Dalam contoh di atas, `struct Stack` digunakan untuk merepresentasikan stack menggunakan array. Fungsi `initializeStack` digunakan untuk menginisialisasi stack dengan mengatur nilai `top` menjadi -1, menandakan bahwa stack kosong.

Menghapus Stack menggunakan Array:

```c
void clearStack(struct Stack* stack) {
    stack->top = -1;
    // Mengatur top kembali ke -1 untuk menghapus stack
}
```

Gambar 1.9 Menghapus Stack Menggunakan Array

Fungsi `clearStack` digunakan untuk menghapus stack dengan mengatur nilai `top` kembali ke -1.

Menginisialisasi dan Menghapus Stack menggunakan Linked List:

```c
struct Node {
    int data;
    struct Node* next;
};

struct Stack {
    struct Node* top;
    // Pointer ke node teratas stack
};

void initializeStack(struct Stack* stack) {
    stack->top = NULL;
    // Menginisialisasi stack dengan top NULL menandakan stack kosong
}

void clearStack(struct Stack* stack) {
    while (stack->top != NULL) {
        struct Node* temp = stack->top;
        stack->top = stack->top->next;
        free(temp);
        // Membebaskan memori untuk setiap node dalam stack
    }
}
```

Gambar 1.10 Menginisialisai dan Menghapus Stack Menggunakan Linked list

Dalam contoh menggunakan linked list, `struct Node` digunakan untuk merepresentasikan node dalam linked list. `struct Stack` memiliki pointer ke node teratas stack. Fungsi `initializeStack` menginisialisasi stack dengan mengatur nilai `top` menjadi NULL, menandakan bahwa stack kosong. Fungsi `clearStack` digunakan untuk menghapus stack dengan membebaskan memori untuk setiap node dalam stack menggunakan fungsi `free`.

Dengan cara ini, Anda dapat menginisialisasi dan menghapus stack dalam implementasi array atau linked list sesuai dengan kebutuhan program Anda. Pastikan untuk memahami dan mengelola dengan benar alokasi memori ketika menggunakan linked list untuk mengimplementasikan stack agar menghindari kebocoran memori (memory leaks).

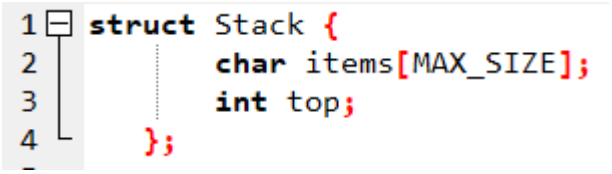
G. Penyusunan Ulang Kata Menggunakan Stack

Program "Penyusunan Ulang Kata Menggunakan Stack" adalah contoh implementasi struktur data stack untuk menyusun ulang sebuah kata. Dalam program ini, kita menggunakan stack untuk membalikkan urutan karakter-karakter dalam kata yang dimasukkan.

Berikut adalah penjelasan langkah-langkah utama dalam program tersebut:

Inisialisasi dan Operasi pada Stack:

1. **Inisialisasi Stack**

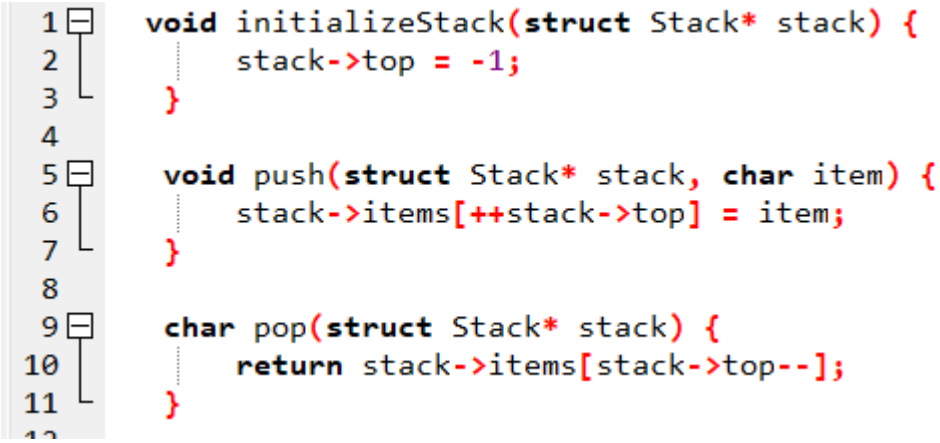


```
struct Stack {
    char items[MAX_SIZE];
    int top;
};
```

Gambar 1.11 Inisialisasi Stack

Kita mendefinisikan struktur `Stack` yang memiliki array karakter `items` untuk menyimpan elemen-elemen stack dan variabel `top` untuk menandakan indeks elemen teratas.

2. **Inisialisasi dan Operasi Stack**



```
void initializeStack(struct Stack* stack) {
    stack->top = -1;
}

void push(struct Stack* stack, char item) {
    stack->items[++stack->top] = item;
}

char pop(struct Stack* stack) {
    return stack->items[stack->top--];
}
```

Gambar 1.12 Inisialiasi dan Operasi Stack

- initializeStack: Digunakan untuk menginisialisasi stack dengan mengatur `top` menjadi -1, menunjukkan stack kosong.
- push: Menambahkan karakter ke dalam stack dengan memindahkan `top` terlebih dahulu dan kemudian menyimpan karakter di indeks `top`.
- pop: Mengambil dan menghapus karakter dari stack dengan mengembalikan karakter pada indeks `top` dan mengurangi `top`.

Penyusunan Ulang Kata

```
void rearrangeWord(char word[]) {
    struct Stack stack;
    initializeStack(&stack);

    // Mengisi stack dengan karakter-karakter kata
    for (int i = 0; word[i] != '\0'; ++i) {
        push(&stack, word[i]);
    }

    // Mengambil karakter-karakter dari stack untuk penyusunan ulang kata
    for (int i = 0; word[i] != '\0'; ++i) {
        word[i] = pop(&stack);
    }
}
```

Gambar 1.13 Penyusunan Kata

Fungsi `rearrangeWord` menerima kata sebagai parameter. Setiap karakter dari kata dimasukkan ke dalam stack menggunakan fungsi `push`. Kemudian, karakter-karakter diambil dari stack menggunakan fungsi `pop` dalam urutan terbalik dan ditempatkan kembali ke variabel `word`. Hasilnya adalah penyusunan ulang dari kata awal.

Fungsi Utama (Main):

```c
int main() {
    char word[MAX_SIZE];

    // Meminta pengguna memasukkan kata
    printf("Masukkan kata: ");
    gets(word);

    // Memanggil fungsi untuk menyusun ulang kata
    rearrangeWord(word);

    // Menampilkan hasil penyusunan ulang kata
    printf("Kata setelah penyusunan ulang: %s\n", word);

    return 0;
}
```

Gambar 1.14 Fungsi Utama (Main)

Fungsi `main` bertanggung jawab untuk berinteraksi dengan pengguna. Program meminta masukan kata dari pengguna, kemudian memanggil fungsi `rearrangeWord` untuk menyusun ulang kata tersebut menggunakan stack. Hasil penyusunan ulang kata kemudian ditampilkan ke layar.

Program ini menggambarkan konsep Last-In-First-Out (LIFO) dari stack, di mana karakter terakhir yang dimasukkan menjadi karakter pertama yang diambil, menghasilkan kata yang terbalik dari input asli. Namun, harap diingat bahwa penggunaan `gets` dalam contoh ini tidak aman dan sebaiknya diganti dengan fungsi input yang lebih aman, terutama dalam pengembangan aplikasi nyata.

H. Pengguna Linked List dalam implementasi Stack

Linked list adalah struktur data linier yang terdiri dari sejumlah simpul (nodes), di mana setiap simpul menyimpan sebuah elemen data dan sebuah referensi (pointer) ke simpul berikutnya dalam urutan. Implementasi stack menggunakan linked list memungkinkan kita untuk membuat stack dengan ukuran dinamis, artinya stack dapat tumbuh atau menyusut sesuai kebutuhan tanpa perlu mendefinisikan ukuran tetap seperti dalam implementasi menggunakan array.

Berikut adalah cara penggunaan linked list dalam implementasi stack:

Struktur Linked List (Node):

```
1  struct Node {
2      int data; // Elemen data
3      struct Node* next; // Pointer ke simpul berikutnya
4  };
```

Gambar 1.15 Struktur Linked List

Setiap simpul dalam linked list menyimpan data (dalam contoh ini, bilangan bulat) dan pointer ke simpul berikutnya.

Struktur Stack menggunakan Linked List:

```
1  struct Stack {
2      struct Node* top; // Pointer ke simpul teratas stack
3  };
```

Gambar 1.16 Struktur Stack Menggunakan Linked List

Struktur stack menggunakan linked list memiliki satu atribut, yaitu `top`, yang merupakan pointer ke simpul teratas dalam linked list. Saat stack kosong, `top` akan menunjuk ke `NULL`.

Operasi Dasar pada Stack menggunakan Linked List:

1. **Inisialisasi Stack:**

```
struct Stack* createStack() {
        struct Stack* stack = (struct Stack*)malloc(sizeof(struct Stack));
        stack->top = NULL;
        return stack;
    }
```

Gambar 1.17 Inisialisai Stack

Fungsi ini membuat dan mengembalikan stack kosong menggunakan alokasi dinamis.

2. **Push:**

```
void push(struct Stack* stack, int data) {
        struct Node* newNode = (struct Node*)malloc(sizeof(struct Node));
        newNode->data = data;
        newNode->next = stack->top;
        stack->top = newNode;
    }
```

Gambar 1.18 Push

Operasi `push` membuat simpul baru, mengatur data, dan mengaitkannya dengan simpul teratas sebelumnya. Simpul baru ini kemudian diatur sebagai simpul teratas stack.

3. **Pop:**

```
int pop(struct Stack* stack) {
        if (isEmpty(stack)) {
            printf("Stack underflow\n");
            return -1; // Nilai khusus untuk menandakan kesalahan
        }
        struct Node* temp = stack->top;
        stack->top = stack->top->next;
        int data = temp->data;
        free(temp); // Membebaskan memori simpul teratas
        return data;
    }
```

Gambar 1.19 Pop

Operasi `pop` mengambil nilai dari simpul teratas, menghapus simpul tersebut, mengganti `top` dengan simpul berikutnya, dan mengembalikan data dari simpul yang dihapus.

4. **Cek Apakah Stack Kosong:**

```
int isEmpty(struct Stack* stack) {
    return stack->top == NULL;
}
```

Gambar 1.20 Mengecek Apakah Stack kosong

Fungsi ini memeriksa apakah stack kosong dengan memeriksa apakah `top` menunjuk ke `NULL`.

5. **Hapus Stack:**

```
void deleteStack(struct Stack* stack) {
    while (!isEmpty(stack)) {
        pop(stack); // Menghapus semua elemen stack
    }
    free(stack); // Membebaskan memori stack
}
```

Gambar 1.21 Menghapus Stack

Fungsi ini menghapus semua elemen dari stack dan kemudian membebaskan memori yang digunakan oleh stack.

Implementasi stack menggunakan linked list memungkinkan operasi-operasi push dan pop untuk memiliki kompleksitas waktu yang konstan (\(O(1)\)), bahkan saat stack memiliki ukuran dinamis. Namun, perlu diingat bahwa perlu menangani dengan hati-hati pengelolaan memori, yaitu membebaskan memori yang dialokasikan untuk setiap simpul yang dihapus.

I. Penggunaan Stack dalam Algoritma Depth-First-Search(DFS)

Depth-First Search (DFS) adalah salah satu algoritma pencarian yang digunakan untuk menjelajahi atau mencari solusi dalam struktur data grafik. Algoritma ini menggunakan konsep stack atau rekursi untuk menjelajahi grafik.

Cara DFS Bekerja:

1. **Stack (atau Rekursi) untuk Melacak Jalur**

- Penggunaan Stack: Dalam pendekatan non-rekursif, kita menggunakan stack untuk melacak simpul yang akan dieksplorasi selanjutnya. Awalnya, simpul awal dimasukkan ke dalam stack. Ketika menjelajahi simpul, simpul yang terhubung dengannya dimasukkan ke dalam stack untuk dijelajahi nanti.
- Penggunaan Rekursi: Dalam pendekatan rekursif, setiap kali menjelajahi simpul, fungsi DFS dipanggil secara rekursif untuk setiap simpul tetangganya.

2. **Penandaan Simpul yang Sudah Dijelajahi:**

Untuk menghindari mengunjungi simpul yang sama berulang kali, simpul yang telah dieksplorasi biasanya ditandai atau dicatat. Ini dapat diimplementasikan sebagai array boolean yang menyatakan apakah simpul telah dikunjungi atau belum.

mplementasi DFS Menggunakan Stack (Non-Rekursif)

Berikut adalah contoh implementasi DFS menggunakan stack dalam bahasa C:

1. **Inisialisai Struktur data dan variable**

```
struct Stack {
    int items[MAX_SIZE];
    int top;
};

void initializeStack(struct Stack* stack) {
    stack->top = -1;
}
```

Gambar 1.22 Inisialisasi Struktur data dan variable

Struktur Stack: Program menggunakan struktur Stack yang memiliki array items untuk menyimpan elemen stack dan variabel top untuk melacak posisi teratas stack.

Fungsi initializeStack: Digunakan untuk menginisialisasi stack dengan mengatur top menjadi -1, menandakan stack kosong.

2. **Operasi Push dan Pop**

```
void push(struct Stack* stack, int item) {
    stack->items[++stack->top] = item;
}

int pop(struct Stack* stack) {
    return stack->items[stack->top--];
}
```

Gambar 1.23 Operasi Push dan Pop

Fungsi push: Digunakan untuk menambahkan elemen ke dalam stack. top ditingkatkan terlebih dahulu, lalu elemen ditambahkan ke posisi top.

Fungsi pop: Digunakan untuk mengambil dan menghapus elemen teratas dari stack. Mengembalikan nilai yang dihapus dari stack.

3. Fungsi DFS

```
void DFS(int graph[][MAX_SIZE], int numVertices, int startVertex) {
    struct Stack stack;
    bool visited[MAX_SIZE];

    initializeStack(&stack);

    // Inisialisasi array visited menjadi false (belum dikunjungi)
    for (int i = 0; i < numVertices; ++i) {
        visited[i] = false;
    }

    push(&stack, startVertex);
    visited[startVertex] = true;

    while (!isEmpty(&stack)) {
        int currentVertex = pop(&stack);
        printf("%d ", currentVertex);

        for (int i = 0; i < numVertices; ++i) {
            if (graph[currentVertex][i] == 1 && !visited[i]) {
                push(&stack, i);
                visited[i] = true;
            }
        }
    }
}
```

Gambar 1.24 Fungsi DFS

Fungsi DFS: Menerima matriks ketetanggaan graph, jumlah simpul numVertices, dan simpul awal startVertex.

Inisialisasi dan Persiapan: Stack diinisialisasi, dan array visited diatur menjadi false untuk menandai simpul-simpul yang belum dikunjungi.

Penelusuran Graf: Melakukan penelusuran dalam graf menggunakan DFS. Memulai dari startVertex, menambahkannya ke stack, dan menandainya sebagai dikunjungi. Selama stack tidak kosong, program melakukan langkah-langkah DFS:

- Mengambil simpul teratas dari stack.
- Menampilkan simpul tersebut.
- Menelusuri semua simpul tetangga yang belum dikunjungi dan memasukkannya ke dalam stack dan menandainya sebagai sudah dikunjungi

4. **Fungsi Utama (main)**

```
int main() {
    // Input: Jumlah simpul, matriks ketetanggaan, simpul awal
    // Output: Hasil DFS traversal

    return 0;
}
```

Gambar 1.25 Fungsi Utama

Fungsi Utama: Dalam fungsi main, biasanya input seperti jumlah simpul dan matriks ketetanggaan dimasukkan. Fungsi DFS kemudian dipanggil dengan parameter-parameter ini untuk menjalankan pencarian dalam graf menggunakan algoritma DFS.

J. Perbandingan antara Stack dan Struktur Data

Stack adalah salah satu struktur data yang penting dalam ilmu komputer. Dalam hal fungsionalitas, stack memiliki karakteristik khusus yang membedakannya dari struktur data lainnya. Di bawah ini adalah perbandingan stack dengan beberapa struktur data lainnya:

1. **Queue:**

- Stack:Menggunakan prinsip LIFO (Last-In-First-Out), artinya elemen yang terakhir dimasukkan adalah yang pertama keluar.
- Queue: Menggunakan prinsip FIFO (First-In-First-Out), artinya elemen yang pertama dimasukkan adalah yang pertama keluar.

2. **Linked List:**

- Stack:Menggunakan linked list untuk implementasinya, dengan operasi push dan pop yang memanipulasi simpul-simpul linked list.
- Linked List: Struktur data yang terdiri dari simpul-simpul yang saling terhubung, tanpa pembatasan urutan akses seperti di stack atau queue.

3. **Array:**

- Stack: Dapat diimplementasikan menggunakan array, dengan aturan akses dari satu ujung array (biasanya akhir) untuk operasi push dan pop.
- Array: Struktur data linear dengan elemen-elemen yang diakses menggunakan indeks. Tidak memiliki pembatasan khusus seperti stack atau queue.

4. **Tree:**

- Stack: Kadang-kadang digunakan dalam traversing tree (seperti dalam algoritma DFS) untuk menyimpan simpul yang harus dieksplorasi berikutnya.
- Tree: Struktur data hierarkis yang terdiri dari simpul-simpul yang terhubung, dimana setiap simpul memiliki nol atau lebih simpul anak.

5. **Heap:**

- Stack:Tidak langsung digunakan dalam implementasi heap, tetapi digunakan dalam beberapa operasi, terutama dalam algoritma pembangunan heap.
- Heap: Struktur data pohon biner yang memiliki sifat tertentu (min-heap atau max-heap) dan digunakan dalam algoritma pengurutan seperti Heap Sort serta dalam algoritma prioritas.

6. **Hash Table:**

- Stack:Tidak terlibat langsung dalam implementasi hash table, tetapi dapat digunakan dalam beberapa operasi yang melibatkan penyimpanan sementara.
- Hash Table: Struktur data yang mengimplementasikan tabel hash, memungkinkan pencarian, penambahan, dan penghapusan dengan waktu konstan dalam kasus terbaik.

7. **Graph**:

- Stack: Digunakan dalam algoritma pencarian grafik, seperti Depth-First Search (DFS), untuk melacak jalur yang telah dijelajahi.
- Graph: Struktur data yang terdiri dari simpul-simpul yang terhubung oleh sisi-sisi, memungkinkan model relasi antara objek-objek.

8. Queue Prioritas:

- Stack: Tidak digunakan dalam struktur data antrian prioritas, tetapi dapat digunakan dalam beberapa operasi internal untuk mengelola data.
- Queue Prioritas : Struktur data mirip dengan antrian, tetapi setiap elemen memiliki prioritas tertentu dan diurutkan berdasarkan prioritas tersebut.

Setiap struktur data memiliki penggunaan dan keunggulannya sendiri tergantung pada kasus penggunaannya. Pemilihan struktur data yang tepat sangat penting dalam merancang dan mengoptimalkan algoritma.

BAB II
PENUTUP

KESIMPULAN

Stack adalah struktur data Last-In-First-Out (LIFO), yang berarti elemen yang terakhir dimasukkan adalah yang pertama keluar. Operasi push digunakan untuk menambahkan elemen ke stack, sedangkan operasi pop digunakan untuk mengambil dan menghapus elemen teratas stack.

Stack dapat diimplementasikan menggunakan array atau linked list. Penggunaan array memungkinkan akses cepat ke elemen-elemen stack dengan indeks, sedangkan penggunaan linked list memberikan fleksibilitas ukuran dan memungkinkan stack tumbuh atau menyusut sesuai kebutuhan.

Stack digunakan dalam berbagai algoritma dan aplikasi, termasuk dalam algoritma pencarian grafik seperti Depth-First Search (DFS), dalam evaluasi ekspresi matematika postfix atau prefix, dan dalam membalik kata atau frasa.

Penting untuk memastikan manajemen memori yang baik ketika menggunakan stack, terutama jika stack diimplementasikan menggunakan alokasi dinamis (seperti linked list). Penggunaan yang tidak benar bisa menyebabkan memory leaks atau stack overflow.

DAFTAR PUSTAKA

https://rudidegil.wordpress.com/2013/02/10/stack-tumpukan-dengan-bahasa-c/

https://rudidegil.wordpress.com/2013/02/10/stack-tumpukan-dengan-bahasa-c/

https://www.belajarstatistik.com/blog/2022/02/09/contoh-progam-stack-dalam-bahasa-c/

http://yuliana.lecturer.pens.ac.id/Struktur%20Data%20C/Prak%20SD%20-%20pdf/Praktikum%204.pdf

https://medium.com/@noransaber685/understanding-the-stack-data-structure-in-c-introduction-implementation-and-examples-8d3fb03de809