MAKALAH

TENTANG REKURSI

Disusun oleh :

Nama	: Muhammad Ajra Zemima Muda
NIM	: 2023903430022
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 Rekursi

Rekursi adalah suatu konsep dalam pemrograman di mana sebuah fungsi dapat memanggil dirinya sendiri secara langsung atau tidak langsung. Dalam konteks pemrograman, rekursi digunakan untuk menyelesaikan suatu tugas atau pemecahan masalah dengan memecahkannya menjadi submasalah yang lebih kecil. Fungsi yang memanggil dirinya sendiri disebut fungsi rekursif, dan rekursi dapat memberikan solusi yang elegan untuk beberapa jenis permasalahan.

Pada umumnya, suatu fungsi rekursif memiliki dua bagian utama:

- Kasus Base (Base Case): Ini adalah kondisi khusus di mana rekursi berhenti dan tidak memanggil dirinya sendiri lagi. Tanpa kasus base, fungsi rekursif dapat memasuki loop tak terbatas dan menyebabkan stack overflow atau kegagalan program.
- Langkah Rekursif: Ini adalah langkah-langkah di mana fungsi memanggil dirinya sendiri dengan masukan yang lebih kecil atau lebih sederhana. Tujuannya adalah untuk mendekati kasus base, sehingga rekursi dapat berhenti setelah beberapa langkah.

Contoh sederhana penggunaan rekursi adalah perhitungan faktorial, di mana nilai faktorial dari suatu bilangan adalah hasil perkalian bilangan itu dengan faktorial dari bilangan yang lebih kecil. Jika kasus base adalah faktorial dari 0 (nol) sama dengan 1, maka langkah rekursifnya adalah faktorial dari n adalah n dikali faktorial dari n-1.

Pengertian rekursi ini menjadi dasar untuk pemahaman lebih lanjut tentang cara mengimplementasikannya dalam bahasa pemrograman C dan bagaimana rekursi dapat digunakan untuk menyelesaikan berbagai masalah pemrograman.

B. Cara Mendefinisikan Fungsi Rekursi

Mendefinisikan fungsi rekursif melibatkan dua elemen utama: kasus base dan langkah rekursif.

1. Kasus Base:

Kasus base adalah kondisi khusus di mana rekursi berhenti. Ini adalah titik akhir yang mencegah fungsi rekursif terus memanggil dirinya sendiri. Kasus base ditentukan agar rekursi dapat menghasilkan hasil yang benar.

Contoh kasus base untuk perhitungan faktorial dari suatu bilangan n:

```
if (n == 0 || n == 1) {
    return 1; // Kasus base: factorial(0) dan factorial(1) sama dengan 1
}
```

Gambar 1. 1.Kasus Base

2. Langkah Rekursif:

Langkah rekursif adalah tempat di mana fungsi memanggil dirinya sendiri dengan argumen yang berbeda. Tujuannya adalah mendekati kasus base agar rekursi dapat berhenti setelah beberapa panggilan.

Contoh langkah rekursif untuk perhitungan faktorial dari suatu bilangan n:

```
else {
    return n * factorial(n - 1); // Langkah rekursif: n dikali factorial dari (n-1)
}
```

Gambar 1. 2.Langkah Rekursif

Dengan menggabungkan kasus base dan langkah rekursif, definisi lengkap fungsi rekursif untuk perhitungan faktorial dalam bahasa C menjadi:

```c
int factorial(int n) {
    // Kasus base
    if (n == 0 || n == 1) {
        return 1;
    }
    // Langkah rekursif
    else {
        return n * factorial(n - 1);
    }
}
```

Gambar 1. 3.Menggabungkan Kasus Base dan Langkah Rekursif

Pada dasarnya, inilah cara mendefinisikan fungsi rekursif dalam bahasa pemrograman C. Perhatikan bahwa penting untuk merancang kasus base dengan benar agar rekursi tidak memasuki loop tak terbatas.

C. Keuntungan dan Kekurangan Rekursi

a. Keuntungan Rekursi:

- Kode Lebih Elegan: Penggunaan rekursi sering kali menghasilkan kode yang lebih singkat dan elegan dibandingkan dengan solusi non-rekursif. Ini dapat membuat kode lebih mudah dibaca dan dimengerti.
- Penyelesaian Masalah yang Terstruktur: Rekursi dapat digunakan untuk menyelesaikan masalah yang dapat dipecahkan secara terstruktur dengan memecahnya menjadi submasalah yang lebih kecil. Hal ini membuat pemecahan masalah menjadi lebih terorganisir.
- Fleksibilitas: Fungsi rekursif dapat dengan mudah diadaptasi untuk menangani masalah yang memiliki struktur serupa tetapi ukuran yang berbeda. Dengan merancang kasus base dengan benar, fungsi rekursif dapat digunakan untuk berbagai situasi.

b. Kekurangan Rekursi:

- Kinerja: Implementasi rekursi dapat memiliki kinerja yang kurang efisien dibandingkan dengan pendekatan iteratif. Hal ini disebabkan oleh overhead penumpukan (stack overhead) yang terjadi ketika fungsi rekursif memanggil dirinya sendiri.
- Pemahaman yang Sulit: Beberapa programmer mungkin kesulitan memahami rekursi karena sifatnya yang seringkali abstrak. Kesalahan dalam merancang atau memahami kasus base dapat menyebabkan bug sulit dilacak.
- Pemakaian Memori: Setiap panggilan rekursif menambahkan entri ke call stack, dan jika rekursi terlalu dalam, ini dapat mengakibatkan stack overflow (melebihi kapasitas stack).

Meskipun rekursi memiliki kelebihan dan kekurangan, keputusan untuk menggunakannya atau tidak tergantung pada sifat masalah yang dihadapi dan preferensi pemrogram. Dalam beberapa kasus, rekursi dapat menjadi alat yang sangat kuat, sementara dalam kasus lain, solusi iteratif mungkin lebih sesuai.

D. Kapan Menggunakan Rekursif

a. Kapan Menggunakan Rekursi:

1. Ketika Struktur Masalah Bersifat Rekursif:

Rekursi cocok digunakan ketika struktur masalah secara alami dapat dibagi menjadi submasalah yang serupa atau identik dengan masalah asalnya. Contohnya termasuk perhitungan faktorial, deret Fibonacci, atau pencarian dalam struktur data seperti pohon.

2. Pemecahan Masalah yang Terstruktur:

Rekursi adalah pilihan yang baik ketika masalah dapat dipecahkan secara terstruktur dengan mengurai masalah besar menjadi submasalah yang lebih kecil. Ini membuat desain solusi menjadi lebih bersih dan mudah dimengerti.

3. Kasus Base Jelas dan Terdefinisi Baik:

Keberhasilan penggunaan rekursi bergantung pada perancangan kasus base yang jelas dan terdefinisi dengan baik. Kasus base adalah kondisi di mana rekursi berhenti, dan jika kasus base tidak ditentukan dengan benar, rekursi dapat berlanjut secara tak terbatas.

4. Fleksibilitas Dalam Penanganan Data Dinamis:

Rekursi dapat digunakan secara efektif untuk menangani struktur data dinamis, seperti pohon atau graf, di mana kedalaman atau ukuran struktur tidak diketahui pada saat kompilasi.

5. Ketika Kode Lebih Membaca dan Mempertahankan:

Dalam beberapa kasus, rekursi dapat menghasilkan kode yang lebih singkat, elegan, dan mudah dipahami dibandingkan dengan solusi iteratif. Jika kejelasan dan keterbacaan kode adalah prioritas, rekursi mungkin menjadi pilihan yang baik.

Namun, perlu diingat bahwa tidak semua masalah cocok untuk pendekatan rekursif. Pemilihan antara rekursi dan iterasi bergantung pada sifat spesifik masalah dan keterbacaan serta kinerja solusi yang diinginkan.

E. Pemanggilan Fungsi Rekursif

Pemanggilan fungsi rekursif melibatkan langkah-langkah tertentu yang terjadi saat fungsi memanggil dirinya sendiri. Dalam konteks pemanggilan fungsi rekursif, konsep call stack menjadi penting.

1. Langkah-langkah dalam Pemanggilan Rekursif:

a. Pemanggilan Fungsi:

Saat fungsi rekursif dipanggil, program menyimpan konteks saat ini (nilai-nilai variabel lokal dan alamat instruksi saat ini) di call stack.

b. Eksekusi Fungsi:

Fungsi rekursif dieksekusi dengan argumen yang diberikan. Selama eksekusi, fungsi dapat memanggil dirinya sendiri untuk menyelesaikan submasalah.

c. Pemanggilan Rekursif:

Pemanggilan rekursif memicu pembentukan instance fungsi baru di call stack. Setiap panggilan rekursif memiliki lingkup variabel lokalnya sendiri.

d. Kasus Base:

Saat mencapai kasus base, eksekusi berhenti, dan nilai dikembalikan. Nilai-nilai kembali melalui seluruh rantai pemanggilan rekursif di call stack.

e. Pengembalian Nilai:

Nilai-nilai dikembalikan ke pemanggil sebelumnya. Pemanggil rekursif selesai dieksekusi, dan kontrol kembali ke pemanggil sebelumnya.

f. Penyelesaian:

Proses berlanjut hingga semua pemanggilan rekursif selesai dieksekusi, dan program mencapai titik di mana fungsi awal dipanggil. Hasil akhir dikembalikan ke pemanggil fungsi awal.

g. Stack dalam Pemanggilan Rekursif:

Call stack berperan penting dalam pemanggilan fungsi rekursif. Setiap panggilan fungsi dan variabel lokalnya disimpan dalam

bentuk tumpukan (stack). Ketika fungsi rekursif memanggil dirinya sendiri, stack bertambah, dan ketika kasus base tercapai, stack menyusut.

Pemahaman tentang bagaimana call stack berperan membantu dalam mencegah stack overflow dan memastikan bahwa rekursi berhenti dengan benar setelah mencapai kasus base.

Dengan memahami proses pemanggilan rekursif dan peran call stack, programmer dapat mengoptimalkan dan memahami lebih baik implementasi fungsi rekursif dalam bahasa pemrograman C.

F. Stack dalam Pemanggilan Rekursif

Pada implementasi fungsi rekursif dalam bahasa C, stack berperan penting dalam menyimpan informasi tentang pemanggilan fungsi dan memastikan pemulihan yang benar setelah pemanggilan fungsi selesai dieksekusi.

- **Pemanggilan Fungsi dan Stack Frames:**

  Setiap kali fungsi dipanggil, suatu stack frame dibuat dan ditambahkan ke call stack. Stack frame ini berisi informasi seperti variabel lokal, parameter, dan alamat instruksi saat ini.

- **Pemanggilan Rekursif dan Call Stack:**

  Dalam pemanggilan rekursif, setiap panggilan rekursif membuat stack frame baru di atas stack. Ini berarti setiap instans rekursi memiliki lingkup variabel lokalnya sendiri, terpisah dari instans rekursi sebelumnya.

- **Kasus Base dan Penghapusan Stack Frames:**

  Saat mencapai kasus base, eksekusi berhenti, dan stack frame terkait dengan pemanggilan rekursif tersebut dihapus dari call stack. Ini disebut dengan proses "unwinding the stack" atau "pemulihan tumpukan."

- **Manajemen Pemanggilan dan Pengembalian Nilai:**

  Stack memastikan bahwa pemanggilan dan pengembalian nilai dilakukan dengan benar. Nilai-nilai yang dikembalikan dari kasus base hingga ke pemanggilan asli diatur oleh stack.

- **Pentingnya Kasus Base:**

  Kasus base berperan penting dalam mencegah stack overflow. Jika kasus base tidak tercapai dengan benar, stack dapat terus bertambah tanpa batas, menyebabkan program berhenti karena stack overflow.

- **Memahami Stack Overflow dan Pengoptimalan:**

  Memahami batasan kapasitas stack membantu dalam mencegah stack overflow. Pengoptimalan dalam rekursi melibatkan pemikiran tentang seberapa dalam rekursi dapat masuk tanpa mengakibatkan kegagalan stack.

Dengan memahami peran stack dalam pemanggilan rekursif, seorang programmer dapat mengoptimalkan kode untuk menghindari masalah stack overflow dan memastikan rekursi bekerja secara efisien.

G. Memahami Kasus Base dalam Rekursif

1. Arti Kasus Base

Kasus base dalam rekursi adalah kondisi khusus di mana fungsi rekursif berhenti memanggil dirinya sendiri dan mengembalikan nilai tanpa melakukan pemanggilan rekursif lebih lanjut. Kasus base menentukan kondisi terminasi yang diperlukan agar rekursi tidak memasuki loop tak terbatas.

Dalam banyak kasus, kasus base adalah solusi langsung untuk masalah yang dipecahkan oleh fungsi rekursif. Misalnya, dalam perhitungan faktorial, kasus base adalah ketika n sama dengan 0 atau 1, di mana faktorial adalah 1. Tanpa kasus base, rekursi akan terus berlanjut hingga *stack overflow* atau kegagalan program.

2. Pentingnya Kasus Base

Pentingnya kasus base dapat diilustrasikan dengan beberapa poin:

a. **Mencegah Loop Tak Terbatas:** Kasus base mencegah fungsi rekursif memanggil dirinya sendiri tanpa henti, yang dapat menyebabkan *stack overflow* atau kegagalan program. Tanpa kasus base, rekursi tidak akan pernah berhenti.

b. **Memberikan Hasil Akhir:** Kasus base memberikan hasil akhir atau solusi langsung untuk masalah. Ini adalah nilai yang dikembalikan saat rekursi mencapai kondisi terminasi. Dengan kata lain, kasus base memberikan fondasi untuk membangun solusi.

c. **Definisi yang Benar:** Kasus base harus dirancang sedemikian rupa sehingga merinci solusi untuk masalah paling sederhana atau paling dasar. Ini memastikan bahwa rekursi memiliki definisi yang benar dan dapat menghasilkan hasil yang diharapkan.

d. **Mencegah Kesalahan dan Bug:** Kasus base yang tidak tepat atau tidak ditentukan dengan benar dapat menyebabkan kesalahan atau bug dalam implementasi rekursif. Oleh karena itu, merancang kasus base dengan hati-hati penting untuk memastikan keberhasilan rekursi.

Pahami dengan baik kasus base dan pastikan bahwa itu terpenuhi dengan benar untuk setiap implementasi fungsi rekursif. Ini adalah langkah kritis dalam merancang rekursi yang efektif dan bebas dari masalah.

H. Kesalahan Umum Dan Cara Menghindari Infinite Recursion

a. Kesalahan Umum dalam Kasus Base:

Kesalahan umum dalam penggunaan rekursi sering terkait dengan kasus base. Beberapa kesalahan yang dapat terjadi melibatkan:

- Oversight Kasus Base: Terkadang, programmer lupa atau mengabaikan untuk menentukan kasus base, yang dapat menyebabkan rekursi tak terbatas.
- Pendefinisian Kasus Base yang Tidak Tepat: Merancang kasus base yang tidak tepat atau tidak memperhatikan semua kemungkinan kondisi dapat menghasilkan rekursi tak terbatas atau hasil yang tidak diinginkan.

b. Cara Menghindari Infinite Recursion:

Untuk menghindari rekursi tak terbatas, diperlukan perhatian khusus terhadap perancangan kasus base dan pengujian input. Beberapa cara untuk menghindari *infinite recursion* melibatkan:

- **Pastikan Kasus Base Tercapai:** Pastikan bahwa setiap jalur rekursi pada suatu titik mencapai kasus base. Kasus base harus merinci kondisi di mana rekursi berhenti.
- **Uji dengan Input yang Bervariasi:** Uji fungsi rekursif dengan input yang bervariasi untuk memastikan bahwa kasus base tercapai dalam berbagai kondisi. Input yang sama seharusnya tidak menyebabkan rekursi tak terbatas.
- **Pantau dan Debug Rekursi:** Gunakan alat pemecah masalah (*debugger*) atau tambahan output untuk memantau jalur rekursi dan nilai-nilai yang dikembalikan. Hal ini membantu mengidentifikasi di mana rekursi mungkin menjadi tak terbatas.
- **Periksa Kasus Base Secara Kritis:** Periksa kembali kasus base untuk memastikan bahwa itu memberikan hasil yang diharapkan dan benar-benar menghentikan rekursi. Kesalahan kecil dalam kasus base dapat menyebabkan masalah besar.

- **Pertimbangkan Batasan Kedalaman Rekursi:** Menetapkan batasan kedalaman rekursi atau jumlah panggilan rekursif dapat menjadi langkah pengaman untuk mencegah *stack overflow*.

Dengan memperhatikan dan mengatasi kesalahan umum ini, programmer dapat meminimalkan risiko rekursi tak terbatas dan memastikan keberhasilan implementasi rekursif.

I. Menghitung Faktorial

a. Implementasi Iterasi

Implementasi iteratif dari perhitungan faktorial dapat dilakukan dengan menggunakan perulangan, seperti for atau while loop. Berikut adalah contoh implementasi iteratif dalam bahasa C untuk menghitung faktorial:

```c
#include <stdio.h>

int factorialIterative(int n) {
    int result = 1;
    for (int i = 1; i <= n; ++i) {
        result *= i;
    }
    return result;
}

int main() {
    int number = 5;
    printf("Factorial of %d (iterative): %d\n", number, factorialIterative(number));
    return 0;
}
```

Gambar 1. 4.Implementasi Iterasi

b. Implementasi Rekursif

Implementasi rekursif dari perhitungan faktorial melibatkan pemanggilan fungsi rekursif dengan kasus base. Berikut adalah contoh implementasi rekursif dalam bahasa C untuk menghitung faktorial:

```c
#include <stdio.h>

int factorialRecursive(int n) {
    // Kasus base
    if (n == 0 || n == 1) {
        return 1;
    }
    // Langkah rekursif
    else {
        return n * factorialRecursive(n - 1);
    }
}

int main() {
    int number = 5;
    printf("Factorial of %d (recursive): %d\n", number, factorialRecursive(number));
    return 0;
}
```

Gambar 1. 5.Implementasi Rekursif

Dalam contoh ini, factorialRecursive adalah fungsi rekursif untuk menghitung faktorial. Kasus base diatur ketika n sama dengan 0 atau 1, di mana faktorialnya adalah 1. Langkah rekursif mengalikan n dengan faktorial dari n-1. Program utama kemudian memanggil fungsi ini dengan suatu angka dan menampilkan hasilnya.

Dengan memahami perbedaan antara implementasi iteratif dan rekursif, seorang programmer dapat memilih pendekatan yang sesuai dengan kebutuhan spesifik program atau masalah yang dihadapi.

BAB II

PENUTUP

A. Kesimpulan

Makalah ini membahas konsep rekursi dalam konteks bahasa pemrograman C. Rekursi adalah suatu teknik pemrograman di mana suatu fungsi memanggil dirinya sendiri. Pemakaian rekursi dapat memberikan solusi yang elegan untuk beberapa permasalahan yang kompleks dengan memecahnya menjadi submasalah yang lebih kecil.

Dalam bahasa C, implementasi rekursi melibatkan pemanggilan fungsi itu sendiri di dalam tubuh fungsi tersebut. Penggunaan fungsi rekursif memerlukan perhatian khusus terhadap kondisi terminasi agar tidak terjadi pemanggilan berulang tanpa henti (infinite recursion).

Selain itu, efisiensi dan pengelolaan memori juga menjadi pertimbangan penting dalam implementasi rekursi. Pemilihan pendekatan rekursif yang tepat dan pemahaman terhadap konsep tumpukan (stack) dapat membantu mengoptimalkan kinerja program.

Dalam rekursi, pemahaman tentang langkah base case (kasus dasar) dan langkah rekursif merupakan kunci untuk memahami dan mengimplementasikan algoritma dengan benar. Kasus dasar memberikan kondisi berhenti untuk rekursi, sementara langkah rekursif menentukan cara memecah masalah menjadi submasalah yang lebih kecil.

Pentingnya dokumentasi dan komentar dalam kode rekursif juga dibahas dalam makalah ini. Dokumentasi yang baik membantu pembaca untuk memahami logika dan tujuan dari fungsi rekursif, serta memfasilitasi pemeliharaan dan pengembangan kode di masa depan.

Secara keseluruhan, rekursi merupakan alat yang kuat dalam bahasa pemrograman C yang dapat digunakan untuk menyelesaikan masalah dengan cara yang lebih abstrak dan terstruktur. Namun, penggunaan rekursi juga memerlukan pemahaman yang mendalam tentang konsep dasar,

pengelolaan memori, dan kehati-hatian dalam menangani kasus dasar untuk menghasilkan solusi yang efisien dan tepat.

DAFTAR PUSTAKA

https://www.duniailkom.com/latihan-kode-program-bahasa-c-fungsi-rekursif-menghitung-faktorial/

https://kumparan.com/how-to-tekno/fungsi-rekursif-dan-contoh-nyatanya-1xqEEPPHXFg

https://www.petanikode.com/c-fungsi/