Modul 9

RECURSIVE FUNCTION

TUJUAN

1. Memahami konsep rekursif langsung dan tidak langsung;
2. Memahami aplikasi dan fungsi rekursif dalam memecahkan masalah pemrograman;
3. Memahami kelebihan dan kekurangan fungsi rekursif, serta;
4. Memahami pemecahan kasus rekursif dan implementasinya di dalam program.

DASAR TEORI

Rekursi (bahasa Inggris: *recursion*) adalah suatu metode yang lazim digunakan di matematika dan pemrograman untuk menyederhanakan algoritma suatu proses. Di dalam pemrograman, fungsi rekursif merupakan fungsi yang mengandung fungsi itu sendiri, sehingga menimbulkan perulangan di dalam fungsi tersebut. Karena proses perulangan terjadi di dalam fungsi itu sendiri, kita tidak lagi menggunakan *for*, *while*, dan *do-while*.

Sebuah fungsi rekursif wajib memiliki definisi *base case* atau *termination point*. *Base case* adalah titik atau kondisi di mana fungsi tersebut akan berhenti. Tanpa adanya *base case*, maka akan terjadi *endless recursion*. Pada kasus terparah, endless recursion dapat memenuhi RAM PC (pada komputer tua), dan memaksa kita force shutdown/restart PC. Pada PC modern saat ini, *endless recursion* hanya akan mengakibatkan program crash.

EXAMPLE

Sebuah truk mengangkut sebuah truk yang kemudian mengangkut sebuah mobil. Ini merupakan sebuah fungsi rekursi dengan *base case* yang terdefinisi dengan benar.

Sebuah truk mengangkut sebuah truk yang mengangkut sebuah truk …. Ini merupakan sebuah fungsi rekursif dengan *base case* yang tidak terdefinisi dengan benar, sehingga menimbulkan *overflow*.

a. *Rekursif langsung dan tidak langsung*

Berdasarkan pemanggilannya, terdapat dua jenis fungsi rekursif yang umum digunakan, yakni :

- Fungsi rekursif langsung

  Fungsi rekursif langsung berarti di dalam fungsi tersebut, terjadi **pemanggilan terhadap dirinya sendiri.** Fungsi jenis ini merupakan fungsi terkursif yang paling sering digunakan.

CODE	OUTPUT

```c
void directRecursion(int n) {
    if (n > 0) {
        printf("%d ", n);
        directRecursion(n - 1);
    }
}

int main() {
    int n = 5;
    directRecursion(n);
    return 0;
}
```

```
5 4 3 2 1
```

- Fungsi rekursif tak langsung

  Fungsi rekursif tidak langsung berarti di dalam fungsi tersebut (sebut saja fungsi *A*) terjadi **pemanggilan terhadap fungsi lain** (sebut saja fungsi *B*), yang dapat saja memanggil fungsi lain juga (sebut saja fungsi *C*), namun pada akhirnya akan kembali memanggil fungsi *A*.

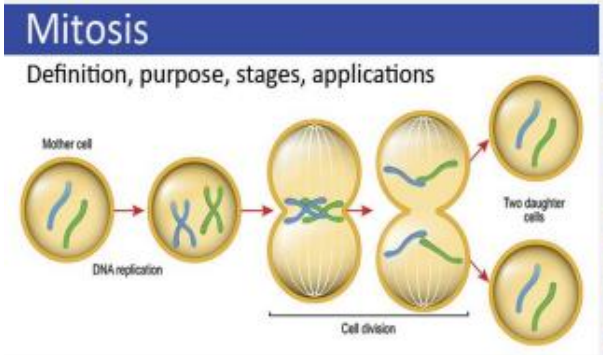
CODE	OUTPUT

```c
void indirectRecursionA(int n);
void indirectRecursionB(int n);

void indirectRecursionA(int n) {
    if (n > 0) {
        printf("%d ", n);
        indirectRecursionB(n - 1);
    }
}
```

```
20 19 9 8 4 3 1
```

```c
void indirectRecursionB(int n) {
    if (n > 1) {
        printf("%d ", n);
        indirectRecursionA(n / 2);
    }
}

int main() {
    int n = 20;
    indirectRecursionA(n);
    return 0;
}
```

EXAMPLE



Mitosis merupakan proses pembelahan sel yang akan menghasilkan sel baru yang secara genetik sama dengan sel inangnya, sehingga merupakan contoh rekursif langsung.

Pulau Samosir merupakan sebuah pulau di Danau Toba, di Pulau Sumatra, merupakan contoh rekursi tidak langsung. *Recursive islands and lakes: en.wikipedia.org/wiki/Recursive_islands_and_lakes*

b. Aplikasi fungsi rekursif

Salah satu contoh kasus fungsi rekursif yang paling sering digunakan adalah perhitungan factorial. Namun sebenarnya, berbagai jenis perulangan iterative dapat juga diekspresikan dalam bentu rekrusif.

1) Perhitungan faktorial (Konsep rekursif dalam bentuk fungsi)

Perhatikan contoh perhitungan factorial berikut :

- $5! = 5 \cdot 4 \cdot 3 \cdot 2 \cdot 1$, hasilnya adalah 120
- $4! = 4 \cdot 3 \cdot 2 \cdot 1$, hasilnya adalah 24
- $2! = 2 \cdot 1$, hasilnya adalah 2
- $1! = 1$, hasilnya adalah 1

Dari contoh tersebut, dapat terlihat suatu pola. Contohnya 5! Dapat dirumuskan menjadi :

$$5! = 5 \cdot 4!$$

Terlihat pula bahwa 4! Adalah $4 \cdot 3!$, sehingga terdapat rumus umum, yakni:

$$n! = n \cdot (n-1)!$$

Rumus umum perhitungan faktorial didefinisikan sebagai berikut:

$$n! = \begin{cases} 1 & \text{jika n} = 0 \\ n \cdot (n-1)!, & jika\ n > 0 \end{cases}$$

Dengan demikian, kita dapat membuat algoritmanya sebagai berikut:

Algoritma non-rekursif (iteratif):

CODE	OUTPUT

```c
int faktorialIterasi(int n) {
    int i, hasil = 1;
    for (i=1; i<=n; i++) {
        hasil = hasil * i;
    }
    return hasil;
}

int main() {
    int n = 5;
    printf("Faktorial dari %d
adalah %d", n,
faktorialIterasi(n));
    return 0;
}
```

```
Faktorial dari 5 adalah 120
```

Algoritma rekursif:

Coba bandingkan: apakah algoritma non-rekursif atau algoritma rekursif lebih mudah dipahami dan lebih sesuai dengan rumus umum perhitungan faktorial.

CODE	OUTPUT

```
int faktorialRekursif(int n) {
    if (n == 0) {
        return 1;
    } else {
        return n *
faktorialRekursif(n - 1);
    }
}
```

```
Faktorial dari 5 adalah 120
```

2) Mencetak (printf) sebuah daftar (konsep rekursif dalam bentu prosedur)

Rekursif tidak hanya dapat dilakukan dengan fungsi, tetapi dapat juga dilakukan dalam bentu prosedur.

Apabila kita memliki sebuah datar (berupa *array of string*), kita dapat mencetaknya menggunakan fungsi rekursif.

Algoritma non-rekursif (iteratif)

CODE	OUTPUT

```
#define MAX 5
typedef char string[100];

void printAllIterative(string
list[]) {
    int i;
    for (i = 0; i < MAX; i++) {
        printf("%s\n", list[i]);
    }
}

int main() {
    string list[MAX] = {
        "1. Tuangkan air",
        "2. Tuangkan susu",
        "3. Tambahkan 1 sdt kopi",
        "4. Aduk hingga merata",
        "5. Tuangkan air panas"
    };
    // Print semua item
    printAllIterative(list);
    return 0;
}
```

```
1. Tuangkan air
2. Tuangkan susu
3. Tambahkan 1 sdt kopi
4. Aduk hingga merata
5. Tuangkan air panas
```

Algoritma rekursif

CODE

```
#define MAX 5
typedef char string[100];

void printAllRecursive(string
list[], int index) {
    printf("%s\n", list[index]);

    if (index == MAX - 1) {
        // Base case: do nothing
    } else {
        // Lanjutkan, index + 1
        printAllRecursive(list,
index + 1);
    }
}

int main() {
    string list[MAX] = {
        "1. Tuangkan air",
        "2. Tuangkan susu",
        "3. Tambahkan 1 sdt kopi",
        "4. Aduk hingga merata",
        "5. Tuangkan air panas"
    };
    // Print semua item
    printAllRecursive(list, 0);
    return 0;
}
```

OUTPUT

```
1. Tuangkan air
2. Tuangkan susu
3. Tambahkan 1 sdt kopi
4. Aduk hingga merata
5. Tuangkan air panas
```

Apabila diamati, terdapat beberapa perbedaan pada algoritma rekursif dan non-rekursif.

Kriteria	Iteratif	Rekursif
Parameter prosedur	`void printAllIterative(string list[])` Hanya ada 1 parameter: list yang ingin dicetak	`void printAllRecursive(string list[], int index)` Ada 2 parameter: list yang ingin dicetak dan indeks saat ini.

Body prosedur	`int i;`
	`for (i = 0; i < MAX; i++) {`
	`  printf("%s\n", list[i]);`
	`}`
	Kita mendeklarasikan variabel ‘i’, sebagai penanda indeks, kemudianmenggunakan *for*
		`printf("%s\n", list[index]);`
		`if (index == MAX - 1) {`
		`  // Base case: do nothing`
		`} else {`
		`  // Lanjutkan, index + 1`
		`  printAllRecursive(list, index + 1);`
		`}`

	loop untu mencetak item di list	Kita terlebih dahulu mencetak item di list, kemudian memeriksa apaka kita sudah mencapai ujung list (*base case*). Apabila belum mencapai ujung list, lanjutkan algoritma rekursif.
Pemanggilan prosedur	`printAllIterative(list);` Sesuai dengan parameter yang ada, kita hanya perlu menyertakan list yang ingin dicetak.	`printAllRecursive(list, 0);` Karena ada parameter `index`, maka perlu juga kita sertakan indeks pertama dari list tersebut, yakni index 0.

CHALLENGE	EXPECTED OUTPUT
Coba balikkan urutan cetak dari daftar di atas menggunakan algoritma rekursif. Hasilnya harus sama dengan di samping →	5. Tuangkan air panas 4. Aduk hingga merata 3. Tambahkan 1 sdt kopi 2. Tuangkan susu 1. Tuangkan air

c. Kelebihan dan kekurangan fungsi rekursif

Fungsi rekursif tidak dapat digunakan untuk menyelesaikan semua masalah. Ada beberapa kelebihan dan kekurangan dari fungsi rekursif :

- Kelebihan
  1) Lebih sederhana dibandingkan perulangan iterative
  2) Mengurangi pemanggilan fungsi berlebihan
  3) Dalam beberapa kasus, lebih mudah digunakan untuk menyelesaikan suatu masalah (menghitung pangkat, factorial, Fibonacci, deret, dll).
- Kekurangan
  1) Membutuhkan memori yang lebih besar
  2) Kasus terburuk jika *endless recursion*: PC *crash*

Modul ini (fungsi rekursif) merupakan konsep dasar dari modul-modul selanjutnya, yakni Linked List Recursive, Binary Tree, Searching, dan Sorting. Pemahaman akan konsep algoritma rekursif penting bagi modul-modul ke depannya.

Bacaan lanjutan :

- https://www.geeksforgeeks.org/introduction-to-recursion-data-structure-and-algorithm-tutorials/
- https://www.geeksforgeeks.org/recursive-functions/

Sumber dan referensi :

- https://en.wikipedia.org/wiki/Recursion_(computer_science
- https://www.programiz.com/cpp-programming/recursion
- https://www.reddit.com/r/ProgrammerHumor/search/?q=recursion&source=recent&restrict_sr=1&include_over_18=1&sort=top&t=all

To understand recursion, one must first understand recursion.

STEPHEN HAWKING

GUIDED

Pada Header :

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

#define MAX 5

typedef char string[100];

typedef struct{
    string nama;
    string kodeBarang;
}Belanjaan;

void showMenu();
void initData(Belanjaan B[], int index1);
int isEmpty(Belanjaan B[], int index1);
Belanjaan makeBelanjaan(string nama, string kodeBarang);

void tampilData(Belanjaan B[], int index1);
void singlePrint(Belanjaan B[], int index2);

int isFound(Belanjaan B[], int index1, string target);
```

Pada Source :

```cpp
#include "header.h"

void showMenu(){
    system("cls");
    printf("\n\t--=== GUIDED ATBA MART ===---");
    printf("\n[1]. Basukkan Data Belanjaan");
    printf("\n[2]. Tampil Seluruh Belanjaan");
    printf("\n[3]. Tampil Spesifik Data Belanjaan");

    printf("\n\n[0]. Keluar\n");
    printf("\n>>> ");
}

Belanjaan makeBelanjaan(string nama, string kodeBarang){
    Belanjaan temp;

    strcpy(temp.nama, nama);
    strcpy(temp.kodeBarang, kodeBarang);

    return temp;
}

void initData(Belanjaan B[], int index1){
    if(index1 >= MAX){  // Pengecekan jika index nya melebihi atau sama dengan max
        return;         // maka tidak ada melakukan init
        // END REKURSIF
    }

    B[index1] = makeBelanjaan("-", "-");
    // Melakukan Inisialisasi untuk belanjaan berdasarkan index sekarang

    initData(B, index1 + 1);
    // Memanggil prosedur initData lagi akan tetapi index nya di tambahkan satu
    // jadi ketika masuk ke fungsi initData nya lagi, index nya sudah berubah
    // Jadi misal index1 sekarang itu 0, lalu ketika memanggil prosedur initData
       lagi dengan index1 +1
    // Maka prosedur initData selanjutnya akan memiliki index1 sebagai 1
}

int isEmpty(Belanjaan B[], int index1){
    if(index1 >= MAX){  // Pengecekan jika index nya melebihi atau sama dengan max
        return -1;      // maka akan mengembalikan nilai -1
        // END REKURSIF
    }

    if(strcmpi(B[index1].nama, "-") != 0){
      // Jika Nama Belanjaan dengan index sekarang ada isi maka akan mengembalikan
         nilai index1
        return index1;
    }

    isEmpty(B, index1 + 1);
    // Setelah melakukan pengecekan diatas jika belum ada nama belanjaan yang ada
       isinya
    // maka akan memanggil fungsi isEmpty lagi dengan index + 1 untuk mengecek
        apakah Nama Belanjaan
    // Dengan index selanjutnya ada isi atau tidak hinga kondisi index1 >= MAX
        terpenuhi
}
```

```
void tampilData(Belanjaan B[], int index1){
    if(index1 >= MAX){
        return;
        // END REKURSIF
    }

    if(strcmpi(B[index1].nama, "-") != 0){
        printf("\n\tNama belanjaan\t : %s", B[index1].nama);
        printf("\n\tKode barang\t : %s", B[index1].kodeBarang);

        printf("\n");
    }

    tampilData(B, index1 + 1);

    // Semua logikanya hampir mirip, dengan melakukan pengecekan apakah index
       diluar dari jangakuan ( MAX )atau tidak
    // Lalu membuat logika sesuai nama prosedur/fungsi
    // Setelah itu memanggil prosedur/fungsi nya sendiri lagi dengan index + 1
}

void singlePrint(Belanjaan B[], int index2){

    printf("\n\tNama belanjaan\t : %s", B[index2].nama);
    printf("\n\tKode barang\t : %s", B[index2].kodeBarang);

}

int isFound(Belanjaan B[], int index1, string target){
    // Fungsi untuk mencari kode barang yang diinputkan menggunakan rekursif
    if(index1 >= MAX){
        return -1;
        // END REKURSIF
    }

    if(strcmpi(B[index1].kodeBarang, target) == 0){
        return index1;
    }

    return isFound(B, index1 + 1, target);
}
```

Pada Main :

```
#include "header.h"

int main(int argc, char *argv[]) {

    Belanjaan B[MAX], tempB;
    int menu, index1; srand(time(NULL));
    string cari;

    initData(B, 0);

    do{
        showMenu(); scanf("%d", &menu);

        switch(menu){
            case 1:
                index1 = isFound(B, 0, "-");
                if(index1 == -1){
                    printf("\n\t[!] Data belanjaan sudah penuh");
                    break;
                }

                system("cls");
                printf("\n\t\t-= Masukkan Data =-\n");
                printf("\tMasukkan nama Belanjaan\t : "); fflush(stdin); gets(tempB.nama);
                printf("\tMasukkan kode Kelas\t : "); fflush(stdin); gets(tempB.kodeBarang);
                while(strlen(tempB.kodeBarang) != 4){
                    printf("\n\t[!] Kode kelas hanya boleh 4 huruf\n");
                    printf("\tMasukkan kode Kelas\t : "); fflush(stdin); gets(tempB.kodeBarang);
                }
                B[index1] = tempB;
                printf("\n\t[!] Berhasil memasukkan data %s", tempB.nama);

                break;

            case 2:
                index1 = isEmpty(B, 0);
                if(index1 == -1){
                    printf("\n\t[!] Data belanjaan masih kosong");
                    break;
                }

                printf("\n\t-= Tampil Data Seluruh belanjaan =-\n");
                tampilData(B, 0);
                printf("\n\t[!] Berhasil tampil data");
                break;
```

```
            case 3:
                index1 = isEmpty(B, 0);
                if(index1 == -1){
                    printf("\n\t[!] Data belanjaan masih kosong");
                    break;
                }

                printf("\n\tMasukkan Kode Barang yang ingin ditampilkan : ");fflush(stdin);gets(cari);
                index1 = isFound(B, index1, cari);
                if(index1 == -1){
                    printf("\n\t[!] Kode Barang tidak ditemukan [!]");
                    break;
                }

                singlePrint(B, index1);
                break;

            case 0:
                printf("\n\tNAMA PRAKTIKKAN - NPM PRAKTIKKAN - KELAS PRAKTIKKAN");
                break;

            default:
                break;
        }
        getch();
    }while(menu != 0);
    return 0;
}
```

Hint dan Tips UGD :

1) Tidak diperkenankan menggunakan perulangan iterasi (*for…, while…, do…while*) dalam pembuatan berbagai prosedur dan fungsi kecuali untuk menu (di main.c) atau error handling.

```
1. printf("\n\tMasukkan Angka\t : "); scanf("%d", &angka);
2. while(angka < 0 || angka > 10){ // <<===== Perulangan untuk error handling
3.   printf("\n\t[!] Angka tidak boleh negatif ataupun lebih dari 10\n"); // <<===== Ini Error
         Handling Angka
4.   printf("\tMasukkan Angka\t : "); scanf("%d", &angka);
5. }
```

2) Pahamilah cara kerja algoritma rekursif dengan baik, karena akan ada fungsi/prosedur custom (yang belum ada penerapannya di guided) yang perlu dibuat nanti.

Format Pengumpulan:

GD9_X_YYYYY

X = kelas

Y = 5 digit terakhir NPM