MAKALAH REKRUSIF DALAM PEMROGRAMAN

BAHASA C



Oleh :

DIKI CANDRA

NIM 2022903430010

D4 TEKNOLOGI REKAYASA KOMPUTER JARINGAN

POLITEKNIK NEGERI LHOKSEMAWE

2022/2023

# DAFTAR ISI

[DAFTAR ISI 2](#_Toc119702421)

[BAB I PENDAHULUAN 1](#_Toc119702422)

[BAB II PEMBAHASAN 2](#_Toc119702423)

[A. Pengertian Rekrusif 2](#_Toc119702424)

[B. Proses Dalam Rekrusif 3](#_Toc119702425)

[C. Teknik dan Metode Rekrusif 4](#_Toc119702426)

[D. Menyelesaikan Masalah Rekrusif 5](#_Toc119702427)

[E. Fungsi Rekrusif 7](#_Toc119702428)

[F. Kelebihan dan Kekurangan Rekrusif 7](#_Toc119702429)

[G. Implementasi Rekrusif 8](#_Toc119702430)

[H. Contoh Program Penggunaan Rekrusif 10](#_Toc119702431)

[I. Menampilkam Deret Mengunakan Prrogram Rekrusif 11](#_Toc119702432)

[J. Aplikasi Sederhana Rekursi 14](#_Toc119702433)

[K. Program Sederhana Rekrusif 17](#_Toc119702434)

[L. Analisis Algoritma Rekrusif 33](#_Toc119702435)

[M. Tail Call 34](#_Toc119702436)

[N. Menyelesaikan Masalah Rekrusif 37](#_Toc119702437)

[O. Prosedur Rekrusif 39](#_Toc119702438)

[BAB III PENUTUP 41](#_Toc119702439)

[KESIMPULAN 41](#_Toc119702440)

[DAFTAR PUSTAKA 42](#_Toc119702441)

# BAB I PENDAHULUAN

Seiring dengan kebutuhan masyarakat yang dipengaruhi oleh perkembangan teknologi yang begitu pesat terutama di bidang komputer, hampir semua pengolahan data maupun transaksi diberbagai perusahaan diproses menggunakan komputer baik perusahaan swasta maupun instansi pemerintah. Penerapan sistem operasi telah berubah dari sistem manual menjadi sistem komputerisasi, hal itu cukup beralasan mengingat banyaknya kelebihan dari sistem komputerisasi antara lain adalah proses pembelajaran struktur data yang membahas tentang Fungsi dan Rekursif dalam Algoritma. Komputer adalah sistem elektronik untuk memanipulasi data yang cepat dan tepat serta dirancang dan diorganisasikan supaya secara otomatis menerima dan menyimpan data input memperoses dan menghasilkan output dibawah pengawasan instruksi-instruksi program yang tersimpan di memori. Adapun batasan masalah dari makalah ini adalah mengenai pengertian tentang Fungsi dan Rekursif, algoritma dan pemrograman Fungsi dan Rekursif bisa dijalankan di dalam pemrograman Bahasa Pascal.

Tujuan dibuatnya makalah ini yaitu untuk menyelesaikan salah satu tugas mata kuliah Algoritma Dasar Pemrograman. Selain itu, saya juga berharap dibuatnya makalah ini bisa membantu kita semua dalam hal mempelajari ilmu pemrograman. Semoga makalah ini sangat bermanfaat bagi saya khususnya dan kita semua amin amin YaRobbalAlamin.

# BAB II PEMBAHASAN

## Pengertian Rekrusif

Rekrusif adalah suatu proses dari fungsi yang memanggil dirinya sendiri secara berulang-ulang. Karena proses dilakukan berulang-ulang, maka harus ada suatu kondisi yang mengakhiri prosesnya. Jika tidak, maka proses tidak akan pernah berhenti sampai memori yang digunakan tidak dapat menampung lagi. Dalam [matematika](https://id.wikipedia.org/wiki/Matematika) dan [ilmu](https://id.wikipedia.org/wiki/Ilmu_komputer) [komputer](https://id.wikipedia.org/wiki/Ilmu_komputer), kelas dari objek atau metode memperlihatkan perilaku rekursif bila mereka dapat didefinisikan oleh dua properti berikut:

1. Sebuah kasus (atau beberapa kasus) dasar sederhana
2. Sejumlah aturan yang mengurangi kasus-kasus lainnya sampai ke kasus dasarnya.

Sebagai contoh, berikut ini adalah definisi rekursif dari leluhur seseorang:

* [Orang tua](https://id.wikipedia.org/wiki/Orang_tua) seseorang adalah [leluhur](https://id.wikipedia.org/wiki/Leluhur) seseorang (*kasus dasar*).
* Orang tua dari suatu leluhur juga merupakan leluhur-nya (*langkah rekursi*). [Bilangan Fibonacci](https://id.wikipedia.org/wiki/Bilangan_Fibonacci) adalah contoh klasik dari rekursi:
* Fib(0) adalah 0 [kasus dasar]
* Fib(1) adalah 1 [kasus dasar]
* Untuk semua integer n > 1: Fib(n) adalah (Fib(n-1) + Fib(n-2)) [definisi rekursif]

Banyak aksioma matematika berdasarkan aturan-aturan rekursif. Sebagai contohnya, definisi formal dari [bilangan asli](https://id.wikipedia.org/wiki/Bilangan_asli) pada [Aksioma Peano](https://id.wikipedia.org/w/index.php?title=Aksioma_Peano&action=edit&redlink=1) dapat dideskripsikan sebagai: 0 adalah bilangan asli, dan setiap bilangan asli memiliki sebuah suksesor, yang juga merupakan bilangan asli. Dengan kasus dasar ini dan aturan rekursif, seseorang dapat membuat himpunan dari semua bilangan asli.

Gambaran humornya berbunyi: "Untuk memahami rekursi, pertama anda harus memahami rekursi." Atau mungkin yang lebih akurat, dari [Andrew Plotkin](https://id.wikipedia.org/w/index.php?title=Andrew_Plotkin&action=edit&redlink=1): "Jika anda telah mengetahui apa itu rekursi, cukup ingat jawabannya. Kalau tidak, cari orang yang berdiri paling dekat dengan [Douglas Hofstadter](https://id.wikipedia.org/w/index.php?title=Douglas_Hofstadter&action=edit&redlink=1) selain anda; lalu tanya dia rekursi itu apa."Objek matematika yang didefinisikan secara rekursif termasuk [fungsi](https://id.wikipedia.org/wiki/Fungsi_(matematika)), [himpunan](https://id.wikipedia.org/wiki/Himpunan_(matematika)), dan terutama sekali [fraktal.](https://id.wikipedia.org/wiki/Fraktal)

## Proses Dalam Rekrusif

Proses rekursi di program ADP\_rekursi\_1.cpp mempunyai kondisi pengakhiran rekursi, yaitu jika nilai N sudah lebih kecil atau sama dengan nol. Setiap kali fungsi memanggil dirinya sendiri, nilai dari N dikurangi dengan nilai 1, sehingga nilai N akhirnya akan menjadi nol dan proses rekursi akan diakhiri, sehingga fungsi ini akan memanggil dirinya sendiri sebanyak N kali.

Proses rekursi di program ADP\_rekursi\_2.cpp dapat dijelaskan sebagai berikut:

1. Fungsi utama memanggil fungsi Fak\_Rekursi() dengan mengirimkan nilai 5 untuk parameter aktual N, yang maksudnya akan dilakukan perhitungan sebanyak 5 faktorial.
2. Jika nilai dari N pertama kali yang diberikan oleh fungsi utama bernilai kurang atau sama dengan satu, maka hasil faktorial yang akan diberikan adalah bernilai 1.
3. Jika nilai N pertama kali yang diberikan oleh fungsi utama lebih besar dari 1, maka proses rekursi yang pertama adalah:

* F = 5 \* Fak\_Rekursi(4);

Proses ini akan memanggil kembali fungsi dirinya sendiri dengan mengirimkan nilai sebagai nilai N yang baru. Karena nilai N masih lebih besar dari 1, maka proses rekursi ke dua akan dilakukan dengan hasilnya adalah 4 \* Fak\_Rekursi(3). Untuk N adalah 3, hasil yang diperoleh oleh rekursi adalah 3 \* Fak\_Rekursi(2) dan seterusnya sampai nilai N adalah 1. Untuk nilai N sama dengan 1 ini, statemen return(1) akan mengembalikan proses ke bagian yang memanggilnya, yaitu statemen setelah statemen F = 5 \* Fak\_Rekursi(N-1).

Statemen return(F) kemudian baru mengembalikan proses ke fungsi utama. Hasil akhirs nilai F dari penjelasan di atas untuk 5 faktorial dapat dituliskan.

## Teknik dan Metode Rekrusif

1. Teknik Rekursi

Bentuk umum fungsi rekursi

return\_data\_type

function\_name(parameter\_list)

{...

function\_name(...);

... }

1. Metode Rekursi

* Cara lain untuk menyelesaikan permasalahan di atas adalah dengan cara rekursi, dimana *n!* adalah hasil kali dari n dengan *(n-1)!*.
* Untuk menyelesaikan *(n-1)!* adalah sama dengan *n!*, sehingga *(n-1)!* adalah *n- 1* dikalikan dengan *(n-2)!*, dan *(n-2)!* adalah *n-2* dikalikan dengan *(n-3)!* dan seterusnya sampai dengan *n = 1*, kita menghentikan penghitungan *n!*

## Menyelesaikan Masalah Rekrusif

1. Mengidentifikasi Masalah

Buatlah program memecahkan masalah faktorial

1. Menentukan Solusi

Pertama menentukan variabel yaitu menggunakan var n dengan type datanya integer kemudian menentukan prosesnya dengan memasukkan pernyataan jika nilai n>0 maka program akan menjalankan perintah faktorial:n\*faktorial(n-1);. Hal itu berarti jika nilai nyang diinputkan lebih dari 0 maka program akan menjalankan perintah untuk mengalikan n dengan nilai itu sendiri dikurang 1, maka akan menampilkan hasil/outputnya yang telah kita inputkan telebih dahulu**.**

1. Memilih Algoritma

Flowchart

|  |
| --- |
| Angka=Angka+1  Faktorial=Faktorial\*Angka |

1. Menulis Program

Program Rekursi;

Var n:integer;

Function faktorial (n:integer):integer; Begin

If (n=0) then Faktorial:=1

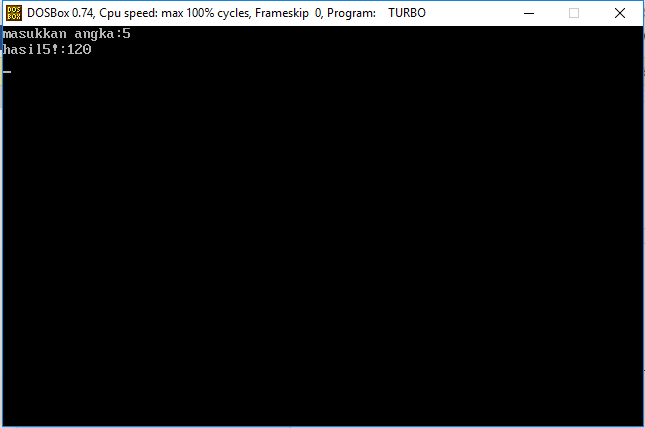
Else Faktorial:=n\*faktorial(n-1); End;

Begin Clrscr;

Write(‘masukkan angka :’);readln(n);

Writeln(‘hasil’,n,’!:’,faktorial(n)); Readln;

End.

1. Menampilkan Output

## Fungsi Rekrusif

Fungsi rekursi adalah suatu fungsi yang memanggil dirinya sendiri. Pada beberapa persoalan, fungsi rekursif sangat berguna karena mempermudah solusi. Namun demikian, fungsi rekursif juga memiliki kelemahan, yakni memungkinkan terjadinya overflow pada stack, yang berarti stack tidak lagi mampu menangani permintaan pemanggilan fungsi karena kehabisan memori( stack adalah area memori yang dipakai untuk variable lokal untuk mengalokasikan memori ketika suatu fungsi dipanggil. Oleh karena itu, jika bisa diselesaikan dengan metode iteratif, gunakanlah metode iteratif.

## Kelebihan dan Kekurangan Rekrusif

1. Kelebihan

* Sangat mudah untuk melakukan perulangan dengan batasan yang luas dalam artian melakukan perulangan dalam skala yang besar.
* Dapat melakukan perulangan dengan batasan fungsi.

1. Kekurangan

* Tidak bisa melakukan nested loop atau looping bersarang.
* Biasanya membuat fungsi sulit untuk dipahami, hanya cocok untuk persoalan tertentu saja.
* Trace error sulit.
* Memerlukan stack yang lebih besar, sebab setiap kali fungsi dipanggil, variabel lokal dan parameter formal akan ditempatkan ke stack dan ada kalanya akan menyebabkan stack tak cukup lagi (Stack Overrun).
* Proses agak berbelit-belit karena terdapat pemangilan fungsi yang berulang-ulang dan pemanggilan data yang ditumpuk.
* Memungkinkan terjadinya overflow pada stack (stack tak lagi mampu menangani permintaan pemanggilan subrutin karena kehabisan memori).

## Implementasi Rekrusif

Implementasi dari rekursi yang paling simple biasanya digunakan untuk menghitung faktorial dari sebuah bilangan atau menghitung bilangan fibonacci. Berikut adalah analogi untuk mencari faktorial dari 5.

faktorial(5)

-> 5 \* faktorial(4)

-> 5 \* (4 \* faktorial(3))

-> 5 \* (4 \* (3 \* faktorial(2)))

-> 5 \* (4 \* (3 \* (2 \* faktorial(1))))

-> 5 \* (4 \* (3 \* (2 \* 1)))

-> 5 \* (4 \* (3 \* 2))

-> 5 \* (4 \* 6)

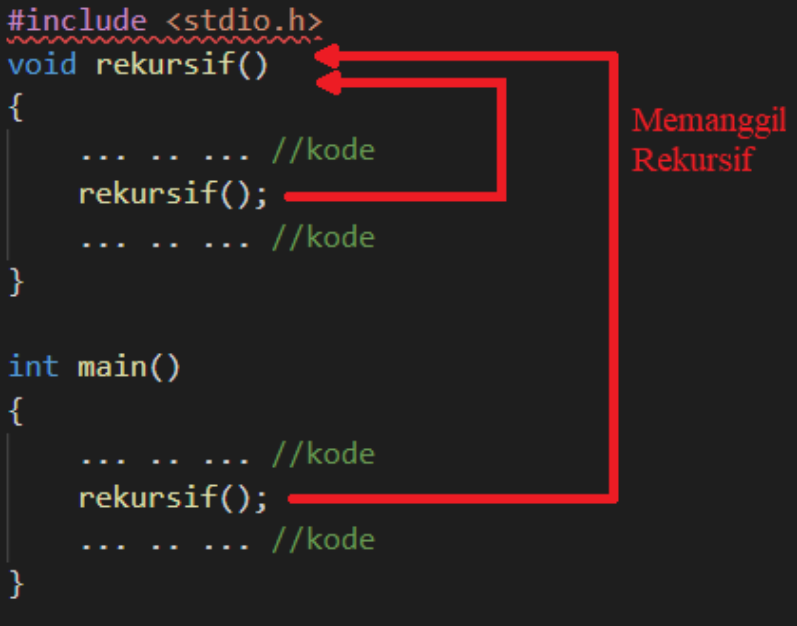
-> 5 \* 24

-> 120

**Dari contoh analogi diatas, kita dapat menarik kesimpulan :**

* Fungsi rekursi selalu memiliki kondisi yang menyatakan kapan fungsi tersebut berhenti. Kondisi ini harus dapat dibuktikan akan tercapai, karena jika tidak tercapai maka kita tidak dapat membuktikan bahwa fungsi akan berhenti, yang berarti algoritma kita tidak benar.
* Fungsi rekursi selalu memanggil dirinya sendiri sambil mengurangi atau memecahkan data masukan setiap panggilannya. Hal ini penting diingat, karena tujuan utama dari rekursif ialah memecahkan masalah dengan mengurangi masalah tersebut menjadi masalah-masalah kecil.

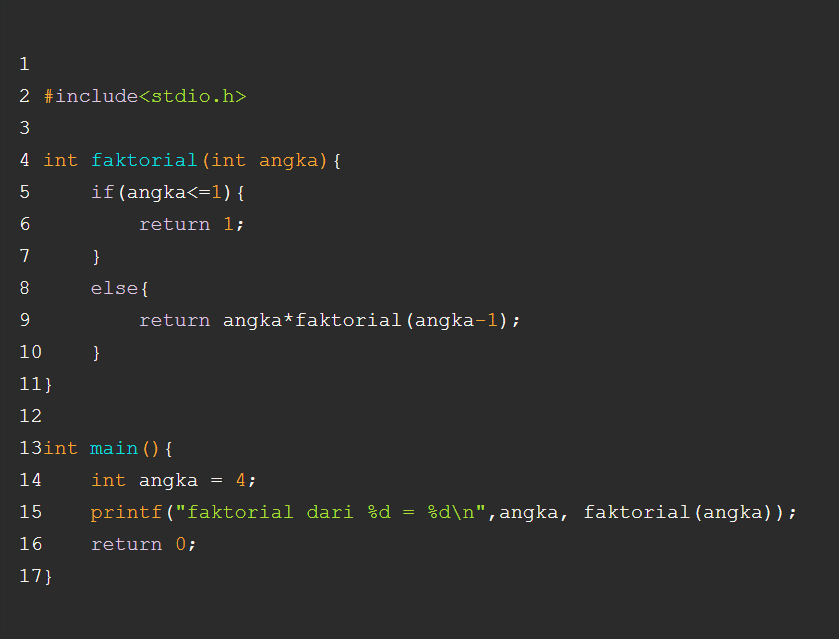




Penjelasan :

* Rekursif akan berlanjut sampai beberapa kondisi terpenuhi untuk menghentikan prosesnya.
* Untuk menghentikan rekursif tak terbatas, dapat menggunakan pernyataan if...else (atau pendekatan serupa) dapat digunakan di mana satu cabang membuat panggilan rekursif, dan yang lainnya tidak.

## Contoh Program Penggunaan Rekrusif



Salah satu contoh penggunaan rekursi adalah program mencari faktorial dari sebuah angka.

Penjelasan :

* Pada function faktorial terdapat penggunaan if else yang bergunakan untuk melanjutkan atau menghentikan rekursif karena rekursif akan melakukan sebuah proses secara berulang – ulang.
* Pada code diatas if berfungsi jika terdapat angka <= 1 maka akan mengembalikan ke 1.

## Menampilkam Deret Mengunakan Prrogram Rekrusif

1. Program 1

#include <stdio.h>

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Fungsi Rekursif untuk f(n) = f(n-1) + 1

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

unsigned int deret1(int n)

{

if (n == 1)

return 1;

else

return (deret1(n-1) + 1);

}

int main()

{

int x;

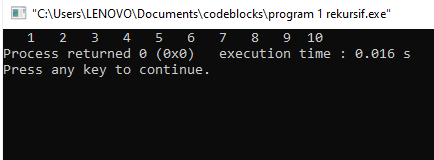
for (x=1; x<=10; x++)

printf("%4d", deret1(x));

return 0;

}

Output :



1. Program 2

#include <stdio.h>

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Fungsi Rekursif untuk f(n) = f(n-1) \* 2

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

unsigned int deret2(int n)

{

if (n == 1)

return 1;

else

return (deret2(n-1) \* 2);

}

int main()

{

int x;

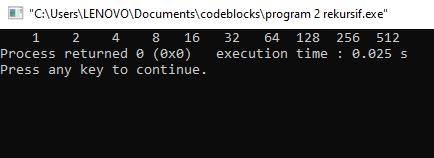
for (x=1; x<=10; x++)

printf("%5d", deret2(x));

return 0;

}

Output :



1. Program 3

#include <stdio.h>

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Fungsi Rekursif untuk f(n) = f(n-2) \* 2

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

unsigned int deret3(int n)

{

if (n == 1)

return 1;

if (n == 2)

return 4;

return (deret3(n-2) \* 2);

}

int main()

{

int x;

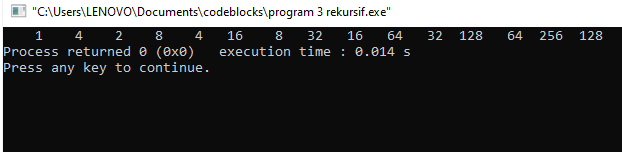
for (x=1; x<=15; x++)

printf("%5d", deret3(x));

return 0;

}

Output :



## Aplikasi Sederhana Rekursi

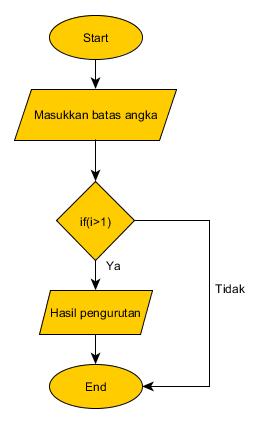
Prosedur dan fungsi merupakan sub program yang sangat bermanfaat dalam pemrograman, terutama untuk program atau proyek yang besar. Manfaat penggunaan sub program antara lain adalah :

1. meningkatkan *readibility*, yaitu mempermudah pembacaan program
2. meningkatkan *modularity*, yaitu memecah sesuatu yang besar menjadi modul-modul atau bagian-bagian yang lebih kecil sesuai dengan fungsinya, sehingga mempermudah pengecekan, testing dan lokalisasi kesalahan.
3. meningkatkan *reusability*, yaitu suatu sub program dapat dipakai berulang kali dengan hanya memanggil sub program tersebut tanpa menuliskan perintah-perintah yang semestinya diulang-ulang.

Sub Program Rekursif adalah sub program yang memanggil dirinya sendiri selama kondisipemanggilan dipenuhi adalah Dengan melihat sifat sub program rekursif di atas maka sub program rekursif harus memiliki :

1. Kondisi yang menyebabkan pemanggilan dirinya berhenti (disebut kondisi khusus atau special condition)
2. Pemanggilan diri sub program (yaitu bila kondisi khusus tidak dipenuhi)

Pada aplikasi yang saya buat merupakan sorting angka dimana yang dimasukkan akan diurutkan dari angka terbesar hingga angka terkecil. Berikut adalah rancangan aplikasi flowchart yang dibuat :



Pada penjelasan program diatas, dimana awal nya kita akan menginputkan jumlah atau batas angka yang ingin kita sorting (urutkan). setelah itu akan masuk pada perhitungan. dimana jika inputan tersebut (i) > 1 maka akan menampilkan hasil sorting , apabila kurang dari 1 maka akan keluar.

Berikut adalah source code :

Source Code :

#include

#include  //pemanggilan library

void angkaterbalik(int i);

int main(void)

{

 int nmr[1];

 printf("==========================================================\n");

 printf("= Pengrurutan Bilangan dari besar - kecil =\n");

 printf("==========================================================\n");

printf("\n");

 printf("Masukkan Batas angka yang di inginkan : \n");

 scanf("%d",&nmr[0]);

 angkaterbalik(nmr[0]);

system("pause");

 return(0);

}

void angkaterbalik(int i)

{

 if(i>=1)

 {

 printf(" %d",i);

 angkaterbalik(i-1);

 }

}

## Program Sederhana Rekrusif

1. Program 1

#include <stdio.h>

/\* prototype \*/ int fact (int N); int fact1 (int N); int fact2 (int N);

/\* program utama \*/ int main() {

/\* contoh program faktorial \*/ printf ("Fact(5)= %d", fact (5));

printf ("Fact1(5)= %d", fact1 (5));

printf ("Fact2(5)= %d", fact2 (5)); return 0;

}

/\*\*\*\*\*\* Body semua fungsi pada prototype \*\*\*\*\*/

/\* versi rekursif, definisi rekursif factorial \*/ int fact (int N) {

if (N == 0) { /\* basis \*/ return 1;

} else {

return (N \* fact(N-1));

}

}

/\*\* Versi iteratif, dengan loop\*\*\*\*\*/ int fact1 (int N) {

int I;

int F; /\* hasil sementara \*/

I = 1; F = 1;

while (I<=N) { F =F\* I; I = I + 1;

} /\* I > N \*/ return F;

}

/\*\*\*versi iteratif, hasil sementara disimpan dalam variabel lokal\*\*\*/

int fact2 (int N) {

int I;

int F; /\* hasil sementara \*/

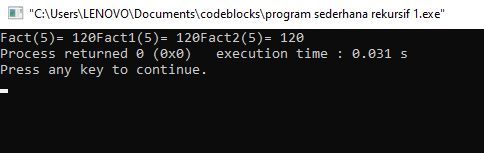
F = 1; /\* penyimpan nilai Faktorial; 0!=1 \*/ I =0;

while (I<N) { I = I+ 1; F = F\* I;

} /\* I= N \*/ return F;

}

Output :



1. Program 2

/\* file : faktorial2.c \*/ #include <stdio.h>

/\* prototype \*/

void fact (int N, int\*Hasil);

void factSalah (int N, int\*Hasil);

int main () {

/\*\*\* variabel milik main \*/ int Hsl;

int HslSalah;

act (5, & Hsl); printf ("Hasil fact (5,&Hsl: %d", Hsl);

factSalah (5, & HslSalah); printf ("Hasil factSalah (5,&HslSalah: %d", HslSalah);

return 0;

}

void fact (int N, int\*Hasil) {

/\* prosedur rekursif, hasil disimpan dalam parameter output\*/ if (N == 0) { /\* basis \*/

\*Hasil = 1;

} else {

fact (N-1, Hasil); /\* Komputasi Hasil=(N-1)! \*/

\*Hasil = N \* (\* Hasil); /\* fact (N-1, N\*Hasil \*/

}

}

void factSalah (int N, int\*Hasil) {

int HslTemp; /\* apa akibatnya kalau ditaruh sbg VAR dalam main\*/

if (N == 0) { /\* basis - 0\*/

\*Hasil = 1;

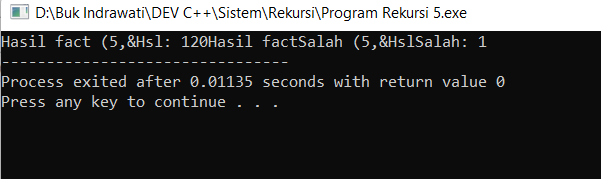
} else {

HslTemp = N \* (\*Hasil); factSalah (N-1, &HslTemp);

}

}

Output :



1. Program 3

/\* file : Add.c \*/ #include <stdio.h>

/\* prototype \*/

int Add (int a, int b);

int main () {

printf ("Add(8,7)= %d", Add(8,7)); return 0;

}

int Add (int a, int b) {

/\* a dan b adalah bilangan integer positif \*/

/\* a+b = (1+1+1+ .....1) + b \*/

/\* 1+1+1... + 1 sebanyak a \*/

if (a == 0) {

return b;

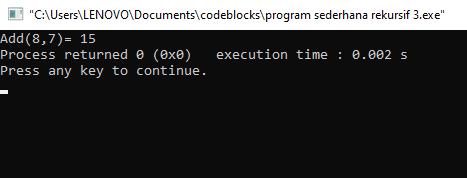
} else {

return ( 1 + Add (a-1, b));

}

}

Output :



1. Program 4

/\* file : Summasi.c \*/ #include <stdio.h>

/\* prototype \*/

int Sigma(int a, int b);

int main () {

printf ("sigma(1,7)= %d", Sigma(1,7)); return 0;

}

int Sigma(int a, int b) {

/\* [a,b] adalah interval yang akan dijadikan type rekursif \*/

/\* Sigma (a,b) menjumlahkan nilai i=a, a+1, .untuk i=a s/d b \*/

/\* pada contoh ini, .interval. [a,b] dibuat mengecil \*/ if (a > b) { /\* Basis-0 : interval kosong \*/

return 0;

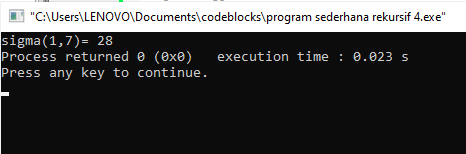
} else { /\* recc, a<=b \*/

return ( a + Sigma (a+1, b));

}

}

Output :



1. Program 5

/\* file : mat.c \*/ #include <stdio.h>

int main () {

/\* kamus \*/

typedef int bulat;

typedef int matriks[10][10]; matriks M;

bulat i,j;

/\* algo\*/

for (i=0;i<10;i++) for (j=0;j<10;j++)

{ M[i][j]= i+j;}

for (i=0;i<10;i++) { for (j=0;j<10;j++)

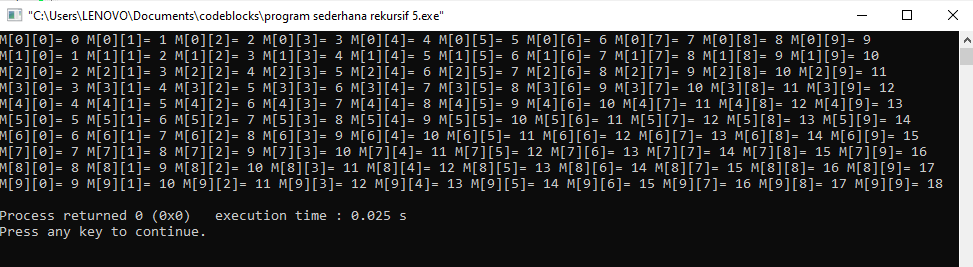
{ printf ("M[%d][%d]= %d ",i,j, M[i][j]); } printf("\n");

}

return 0;

}

Output :



1. Program 6

/\* file : TabelRecc1.c \*/ #include <stdio.h>

/\* type global \*/

typedef struct { int T [11]; /\*T adalah array dengan 10 elemen \*/

} tabel;

/\* variabel global\*/

/\* prototype \*/

void Tulis(tabel T, int a, int b);

/\* menulis isi tabel T dengan indeks a s/d b secara rekursif\*/

void Isi(tabel \* T, int a, int b, int val);

/\* mengisi tabel T dengan indeks a s/d b dengan val secara rekursif\*/ int Sum (tabel T, int a, int b);

/\* menjumlahkan T[a] s/d T[b] secara rekursif \*/

int main () {

tabel T ; /\* T.T[a] \*/

Isi (&T, 1, 10, 5); Tulis (T,1,10);

printf ("Sum(T,1,10) = %d",Sum (T,1,10));

return 0;

}

void Tulis(tabel T, int a, int b) {

/\* Menulis isi Tabel T[a..b] secara rekursif \*/

if (a>b) { /\* Basis-0 : interval kosong \*/

/\* tidak menulis apa-apa \*/

} else { /\* a<= b recc \*/

printf ("T[ %d ] = %d \n", a, T.T[a]);

Tulis (T, a+1, b);

/\* apa akibatnya kalau dua instruksi di atas dibalik ? \*/

/\* Tulis (T, a+1, b); \*/

/\* writeln ('T[',a,'] =', T[a]); \*/

}

}

void Isi(tabel\* T, int a, int b, int val) {

/\* Mengisi isi Tabel T[a..b] supaya bernilai val secara rekursif\*/ if (a>b) { /\* Basis-0 : interval kosong \*/

/\* tidak ada yang diisi \*/

} else { /\* a<= b recc \*/

(\*T).T[a] = val;

Isi (T, a+1, b, val);

}

}

int Sum (tabel T, int a, int b) {

/\* menjumlahkan T[a] s/d T[b] secara rekursif \*/ if (a > b) { /\* Basis-0 : interval kosong \*/

return 0;

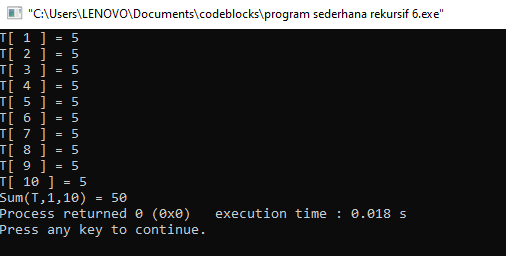
} else { /\* a<= b recc \*/

return (T.T[a]+ Sum (T, a+1, b));

}

}

Output :



1. Program 7

/\* file : TabelRecc1a.c \*/ #include <stdio.h>

/\* type global \*/

typedef int tabel [11]; /\*T adalah array dengan 10 elemen \*/

/\* variabel global\*/

/\* prototype \*/

void Tulis(tabel T, int a, int b);

/\* menulis isi tabel T dengan indeks a s/d b secara rekursif\*/

void Isi(tabel T, int a, int b, int val);

/\* mengisi tabel T dengan indeks a s/d b dengan val secara rekursif\*/ int Sum (tabel T, int a, int b);

/\* menjumlahkan T[a] s/d T[b] secara rekursif \*/

int main () { tabel T ;

Isi (T, 1, 10, 5); Tulis (T,1,10);

printf ("Sum(T,1,10) = %d",Sum (T,1,10));

return 0;

void Tulis(tabel T, int a, int b) {

/\* Menulis isi Tabel T[a..b] secara rekursif \*/

if (a>b) { /\* Basis-0 : interval kosong \*/

/\* tidak menulis apa-apa \*/

} else { /\* a<= b recc \*/

printf ("T[ %d ] = %d \n", a, T[a]); Tulis (T, a+1, b);

/\* apa akibatnya kalau dua instruksi di atas dibalik ? \*/

/\* Tulis (T, a+1, b); \*/

/\* writeln ('T[',a,'] =', T[a]); \*/

}

}

void Isi(tabel T, int a, int b, int val) {

/\* Mengisi isi Tabel T[a..b] supaya bernilai val secara rekursif\*/ if (a>b) { /\* Basis-0 : interval kosong \*/

/\* tidak ada yang diisi \*/

} else { /\* a<= b recc \*/ T[a] = val;

Isi (T, a+1, b, val);

}

}

int Sum (tabel T, int a, int b) {

/\* menjumlahkan T[a] s/d T[b] secara rekursif \*/ if (a > b) { /\* Basis-0 : interval kosong \*/

return 0;

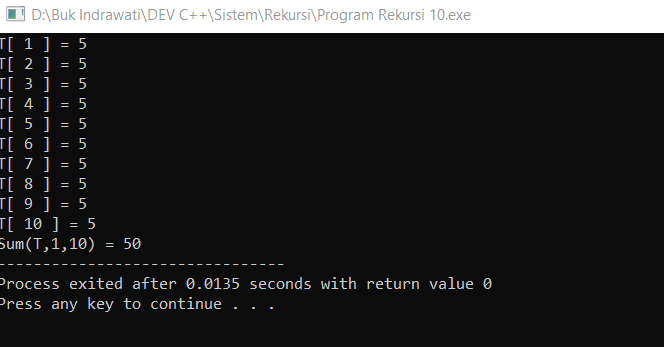
} else { /\* a<= b recc \*/

return (T[a]+ Sum (T, a+1, b));

}

}

Output :



1. Program 8

/\* file : TabelRecc2.c \*/ #include <stdio.h>

/\* pada contoh ini tabel T dinyatakan global, sehingga passing parameter \*/

/\* hanya mempermainkan indeks saja\*/

/\* pada kebanyakan kasus, hal ini dipakai \*/

/\* type global \*/

typedef struct { int T [11]; /\*T adalah array dengan 10 elemen \*/

} tabel;

/\* variabel global\*/ tabel T ;

/\* prototype \*/

void Tulis( int a, int b);

/\* menulis isi tabel T dengan indeks a s/d b secara rekursif\*/ void Isi( int a, int b, int val);

/\* mengisi tabel T dengan indeks a s/d b dengan val secara rekursif\*/ int Sum ( int a, int b);

/\* menjumlahkan T[a] s/d T[b] secara rekursif \*/

int main () {

Isi ( 1, 10, 5); Tulis (1,10);

printf ("Sum(1,10) = %d",Sum (1,10)); return 0;

}

void Tulis( int a, int b) {

/\* Menulis isi Tabel T[a..b] secara rekursif \*/ if (a>b) { /\* Basis-0 : interval kosong \*/

/\* tidak menulis apa-apa \*/

} else { /\* a<= b recc \*/

printf ("T[ %d ] = %d \n", a, T.T[a]); Tulis ( a+1, b);

/\* apa akibatnya kalau dua instruksi di atas dibalik ? \*/

/\* Tulis (T, a+1, b); \*/

/\* writeln ('T[',a,'] =', T[a]); \*/

}

}

void Isi( int a, int b, int val) {

/\* Mengisi isi Tabel T[a..b] supaya bernilai val secara rekursif\*/ if (a>b) { /\* Basis-0 : interval kosong \*/

/\* tidak ada yang diisi \*/

} else { /\* a<= b recc \*/ T.T[a] = val;

Isi ( a+1, b, val);

}

}

int Sum (int a, int b) {

/\* menjumlahkan T[a] s/d T[b] secara rekursif \*/ if (a > b) { /\* Basis-0 : interval kosong \*/

return 0;

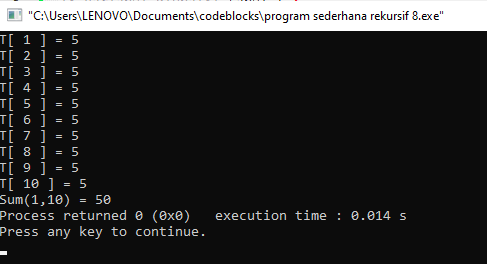
} else { /\* a<= b recc \*/

return (T.T[a]+ Sum ( a+1, b));

}

}

Output :



1. Program 9

/\* File : PrgReccAneh.c \*/ #include <stdio.h>

/\* prototype \*/

void Sum (int N, int \* Counter, int \* Akum);

int main () {

/\* VARiabel global \*/ int Counter;

int Hasil;

/\* algoritma \*/

Counter = 1; Hasil = 1;

Sum (5, &Counter, &Hasil); /\* aneh sekali, tidak konsisten . . \*/ printf ("Counter = %d, Hasil 1+2+3+4+5 = %d \n", Counter,Hasil);

return 0;

}

/\* contoh program rekursif yang agak "aneh" walau hasilnya benar\*/

/\* Mengapa ? Di mana keanehannya ? \*/

void Sum (int N, int \* Counter, int \* Akum){

/\* Menjumlahkan 1+2+3+.... N secara rekursif \*/

if ( \*Counter == N) { /\* .Basis., berhenti \*/

/\* tidak perlu berbuat apa-apa. Kenapa ? \*/

} else { /\* Counter < N \*/

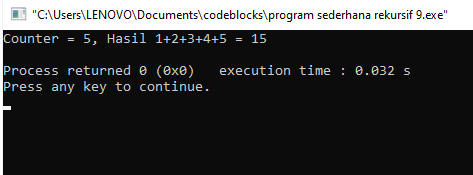
\*Counter = \*Counter + 1; \*Akum = \*Akum + \*Counter; Sum(N, Counter, Akum);

/\*Akan salah jika ditulis Sum (N,Counter+1, Akum+1); Kenapa? \*/

}

}

Output :



1. Program 10

#include <stdio.h>

#include<stdlib.h>

int main(void)

{

int n; int biner (int n);

printf("Konversi Bilangan Desimal Ke Biner\n");

printf("Menggunakan Fungsi Rekursif\n");

printf("==== By: Ronan Rizky ====\n");

printf("Masukan bilangan desimal :"); scanf("%d",&n);

printf("\nbilangan biner = ");

biner(n);

printf("\n");

system("pause");

return(0);

}

int biner( int n) { if(n>1)

{

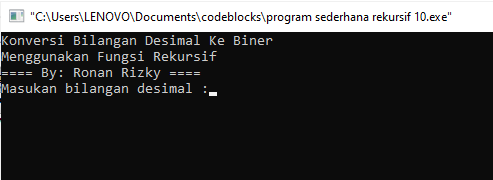
biner(n/2);

}

printf("%d",n%2);

}

Output :



1. Program 11

#include <stdio.h>

#include <stdlib.h>

void DesimalToBiner(int N){

if(N>1){

DesimalToBiner(N/2);

}

printf("%d",N%2);

}

int main()

{

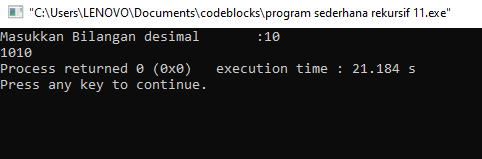
int desimal;

printf("Masukkan Bilangan desimal \t:");scanf("%d",&desimal);

DesimalToBiner(desimal);

}

Output :



1. Program 12

#include <stdio.h>

int faktorial(int angka){

if(angka<=1){

return 1;

}else {

return angka\*faktorial(angka-1);

}

}

int main(){

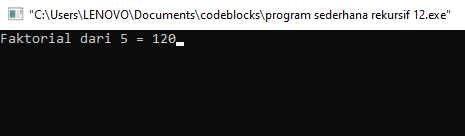
printf("Faktorial dari 5 = %d", faktorial(5));

getchar();

return 0;

}

Output :



## Analisis Algoritma Rekrusif

Melakukan analisis untuk algoritma rekursif pada dasarnya sama dengan melakukan analisis terhadap algoritma imparatif lain. Perbedaan utama pada algoritma rekursif ialah kita tidak dapat secara langsung melihat berapa kali bagian rekursif dari algoritma akan dijalankan. Pada algoritma yang menggunakan perulangan for misalnya, kita dapat langsung menghitung jumlah perulangan untuk menghitung total langkah yang dibutuhkan. Dalam algoritma rekursif, jumlah perulangan tidak secara eksplisit bisa didapatkan karena informasi yang kita miliki adalah kapan algoritma berhenti, bukan berapa kali kode dieksekusi.

## Tail Call

Sesuai definisinya, dalam membuat fungsi rekursif pada satu titik kita akan harus memanggil fungsi itu sendiri. Pemanggilan diri sendiri di dalam fungsi tentunya memiliki konsekuensi tersendiri, yaitu pengunaan memori. Dengan memanggil dirinya sendiri, secara otomatis sebuah fungsi akan memerlukan memori tambahan, untuk menampung seluruh variabel baru serta proses yang harus dilakukan terhadap nilai-nilai baru tersebut. Penambahan memori ini seringkali menyebabkan stack overflow ketika terjadi pemanggilan rekursif yang terlalu banyak.

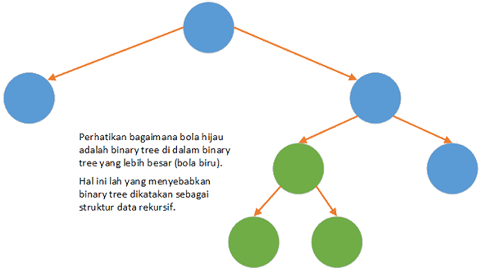
Untuk menanggulangi kesalahan stack overflow ini, para pengembang bahasa pemrograman mengimplementasikan apa yang dikenal dengan **tail call optimization**. Dalam merancang dan menganalisa algoritma rekursif, pengertian akan tail call optimization merupakan hal yang sangat penting. Jadi, apa itu tail call?

Tail call merupakan pemanggilan fungsi sebagai perintah terakhir di dalam fungsi lain.

## Menyelesaikan Masalah Rekrusif

Pendekatan rekursif memecah-mecah masalah untuk kemudian menyelesaikan masalah sedikit demi sedikit, pendekatan iteratif justru langsung mencoba menyelesaikan masalah, tanpa memecah-mecahkan masalah tersebut menjadi lebih kecil terlebih dahulu. Untungnya, baik teknik iterasi maupun rekursi sama-sama memiliki tingkat ekspresi yang sama: segala hal yang dapat diselesaikan dengan itearsi akan dapat diselesaikan dengan rekursif. Lalu, kapan dan kenapa kita harus menggunakan rekursif?

Meskipun dapat menyelesaikan masalah yang sama, terdapat beberapa permasalahan atau solusi yang lebih tepat diselesaikan dengan menggunakan fungsi rekursif. Salah satu contoh dari masalah ini adalah penelurusan data di dalam sebuah binary tree. Sebuah binary tree, yang dapat didefinisikan sebagai sebuah pohon dengan jumlah cabang yang selalu dua, secara alami adalah struktur data rekursif.



Binary Tree

Sebagai struktur rekursif, tentunya penelusuran binary tree akan lebih mudah dilakukan secara rekursif dibandingkan iterasi. Hal ini sangat kontras dengan, misalnya, pencarian karakter di dalam string. Sebagai data yang disimpan secara linear, pencarian karakter dalam string akan lebih mudah untuk dilakukan secara iteratif.

## Prosedur Rekrusif

Prosedur dan fungsi merupakan sub program yang sangat bermanfaat dalam pemrograman, terutama untuk program atau proyek yang besar. Manfaat penggunaan sub program antara lain adalah :

1. meningkatkan readibility, yaitu mempermudah pembacaan program
2. meningkatkan modularity, yaitu memecah sesuatu yang besar menjadi modul-modul atau bagian-bagian yang lebih kecil sesuai dengan fungsinya, sehingga mempermudah pengecekan, testing dan lokalisasi kesalahan.
3. meningkatkan reusability, yaitu suatu sub program dapat dipakai berulang kali dengan hanya memanggil sub program tersebut tanpa menuliskan perintah-perintah yang semestinya diulang-ulang.

Sub Program Rekursif adalah sub program yang memanggil dirinya sendiri selama kondisi pemanggilan dipenuhi. Dengan melihat sifat sub program rekursif di atas maka sub program rekursif harus memiliki :

1. Kondisi yang menyebabkan pemanggilan dirinya berhenti (disebut kondisi khusus atau special condition)
2. Pemanggilan diri sub program (yaitu bila kondisi khusus tidak dipenuhi)

Secara umum bentuk dari sub program rekursif memiliki statemen kondisional:

if kondisi khusus tak dipenuhi  
then panggil diri-sendiri dengan parameter yang sesuai  
else lakukan instruksi yang akan dieksekusi bila kondisi khusus dipenuhi.

Sub program rekursif umumnya dipakai untuk permasalahan yang memiliki langkah penyelesaian yang terpola atau langkah-langkah yang teratur. Bila kita memiliki suatu permasalahan dan kita mengetahui algoritma penyelesaiannya, kadang-kadang sub program rekursif menjadi pilihan kita bila memang memungkinkan untuk dipergunakan. Secara algoritmis (dari segi algoritma, yaitu bila kita mempertimbangkan penggunaan memori, waktu eksekusi sub program) sub program rekursif sering bersifat tidak efisien .

Dengan demikian sub program rekursif umumnya memiliki efisiensi dalam penulisan perintah, tetapi kadang tidak efisien secara algoritmis. Meskipun demikian banyak pula permasalahan-permasalahan yang lebih sesuai diselesaikan dengan cara rekursif (misalnya dalam pencarian / searching, yang akan dibahas pada pertemuan-pertemuan yang akan datang).

**BAB III  
PENUTUP**

# KESIMPULAN

Fungsi rekursif merupakan fungsi yang memanggil dirinya sendiri. Terdapat dua komponen penting dalam fungsi rekursif, yaitu kondisi kapan berhentinya fungsi dan pengurangan atau pembagian data ketika fungsi memanggil dirinya sendiri. Optimasi fungsi rekursif dapat dilakukan dengan menggunakan teknik tail call, meskipun teknik ini tidak selalu diimplementasikan oleh semua bahasa pemrograman.

Selain sebagai fungsi, konsep rekursif juga terkadang digunakan untuk struktur data seperti binary tree atau list.

**DAFTAR PUSTAKA**

*khabib.staff.ugm.ac.id/index.php?option=com\_content&task=view&id=82&Itemid=33*