Fungsi

- Pemrograman modular
- *Library functions* vs *user-defined functions*
- Konstruksi fungsi & prototipe fungsi
- Lingkup *identifier*
- Pengiriman parameter
- Array sebagai parameter
- Fungsi rekursif

Pemgrograman Modular

- Program dibagi menjadi modul-modul
- Modul dalam bahasa C diimplementasikan dengan Fungsi
- Fungsi dibentuk dengan mengelompokkan sejumlah perintah untuk menyelesaikan tugas tertentu.
- Modul diperlukan jika kelompok perintah tersebut kerap kali digunakan di tempat lain dalam program
- Modul sering disebut juga dengan Sub-Program

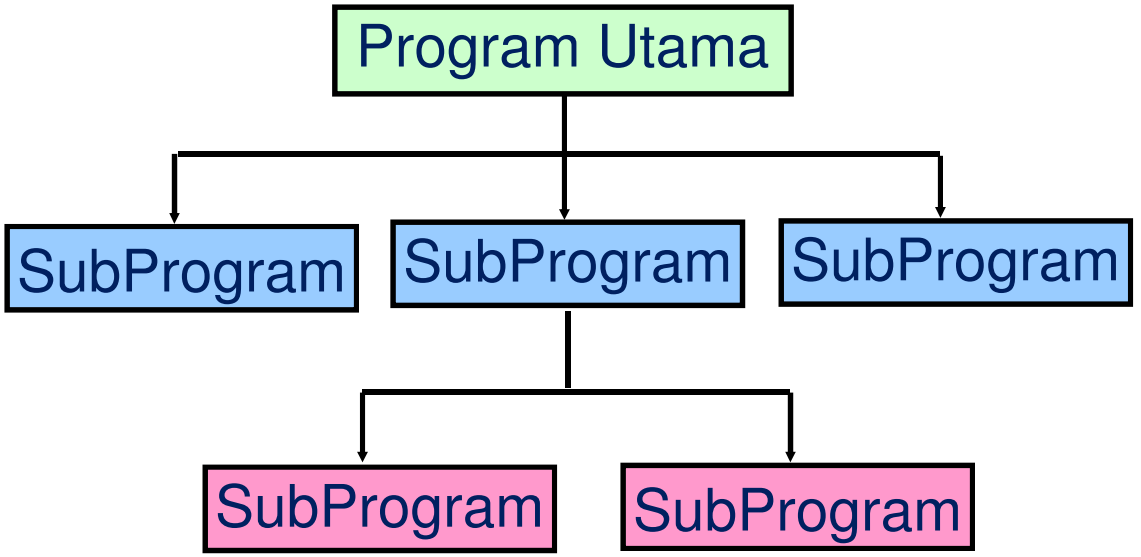
Pemgrograman Modular

Keuntungan:

- Rancangan *top-down* dengan teknik *sub-goal*:
  - Masalah besar/kompleks dapat dijadikan masalah-masalah lebih kecil/sederhana
  - Program besar/kompleks dapat dibagi menjadi modul-modul yang lebih kecil/sederhana.
- Dapat dikerjakan oleh lebih dari satu orang dengan koordinasi yang relatif mudah
- Mencari kesalahan relatif lebih mudah karena alur logika lebih jelas; kesalahan juga dapat dilokalisasi dalam satu modul.
- Modifikasi dapat dilakukan tanpa menggangu program secara keseluruhan
- Mempermudah dokumentasi

Pemrograman Modular

- Bahasa C melengkapi fasilitas modular dengan menggunakan ***fungsi*** pada setiap subprogram.
- Contoh pembagian program menjadi beberapa subprogram.



Pemrograman Modular

- **Sifat-sifat modul yang baik adalah :**
  - **Kohesi (cohesion)** yang tinggi
    - Indikasi kualitatif yang menggambarkan sejauh mana sebuah modul memfokuskan pada satu macam pekerjaan
    - Semakin tinggi kohesi, semakin spesifik tugas yang dikerjakan oleh modul tersebut.
  - **Kopling (coupling)** yang rendah
    - Indikasi kualitatif yang menggambarkan tingkat keterkaitan sebuah modul dengan modul-modul lain dan dengan dunia luar
    - Konektivitas sederhana -> memudahkan pemahaman dan menghindari “ripple effect”
  - **Self-contained**, memenuhi kebutuhannya sendiri.

Library vs User-defined Functions

Fungsi dalam bahasa C:

- Library functions
  - fungsi-fungsi standar yang sudah disediakan oleh *library* bahasa C.
  - dideklarasikan dan didefinisikan dalam *header file* (.h):
    - `printf()` dan `scanf()` dalam `stdio.h`
    - `clrscr()` dalam `conio.h`
    - `sqrt()` dalam `math.h`
- User-defined functions
  - fungsi-fungsi yang didefinisikan sendiri oleh pemrogram

Library vs User-defined Functions

```
#include <stdio.h>
#include <math.h>

int main() {
   int i;
   for(i=0; i<6; i++)
       printf("%d %f",i,sqrt(i));
   return 0;
}
```

Contoh program yang menggunakan *Standard Library Functions*: printf dan sqrt

Konstruksi Fungsi

- Format

```
return-value-type  function-name( parameter-list )
{
      statements;
}
```

- ***return-value-type***: tipe data yang dikembalikan oleh fungsi
  - Jika ***return-value-type*** diganti **void** maka fungsi tidak mengembalikan nilai
- ***parameter-list***: daftar nilai yang dikirimkan dari fungsi pemanggil sebagai parameter fungsi yang dipanggil ini

Konstruksi Fungsi

- Contoh :

formal parameter

Fungsi

```
int maksimum (int x, int y){
        int maks = x;
        if (y > maks) maks = y;
        return maks;
}
```

Pemanggil

```
void main () {
      int a,b;
      printf("Input 2 bilangan bulat : ");
      scanf("%d %d", &a, &b);
      printf("Bilangan yg lebih besar : %d\n",maksimum(a,b));
}
```

Actual parameter

Prototipe Fungsi

- Fungsi pada bahasa C pada dasarnya didefinisikan diatas blok atau fungsi pemanggilnya (`main()` atau fungsi lainnya*)*.

  Namun adakalanya definisi fungsi diletakkan setelah blok pemanggil. Pada kondisi tersebut perlu digunakan prototipe fungsi yang dideklarasikan sebelum fungsi pemanggilnya.

- Tujuan dari prototipe fungsi :
  - Membuat sebuah fungsi dikenal oleh pemanggilnya sebelum fungsi itu didefinisikan

  Compiler akan memvalidasi parameter fungsi tsb

- Sintaksis

```
return-value-type  function-name( parameter-list);
```

Prototipe Fungsi

- Contoh :

```
#include<stdio.h>

int maksimum (int x, int y) {
    int maks = x;
    if (y > maks)
       maks = y;
    return maks
}

void main () {
     int a,b;
     printf("Input 2 bilangan bulat: ");
     scanf("%d %d", &a, &b);
     printf("Bilangan yg lebih besar: %d\n",maksimum(a,b));
}
```

Karena fungsi maksimum diletakkan di atas pemanggilnya (main program), maka tidak perlu prototipe fungsi

Prototipe Fungsi

- Contoh :

```
#include<stdio.h>

int maksimum(int, int);

void main () {
     int a,b;
     printf("Input 2 bilangan bulat: ");
     scanf("%d %d", &a, &b);
     printf("Bilangan yg lebih besar: %d\n",
              maksimum(a,b));
}

int maksimum (int x, int y){
      int maks = x;
      if ( y > maks) maks = y;
      return maks;
}
```

Prototipe Fungsi

Karena fungsi maksimum diletakkan di bawah pemanggilnya (main), maka perlu diletakkan prototipe fungsi diatas,supaya dikenal oleh pemanggilnya

Prototipe Fungsi

- Penulisan prototipe fungsi di atas bisa ditambah nama parameternya:

```
int maksimum(int a, int b);
```

- Yang dipentingkan dalam prototipe fungsi:
  - tipe parameter
  - jumlah parameter
  - urutan parameter

Lingkup *Identifier (scope)*

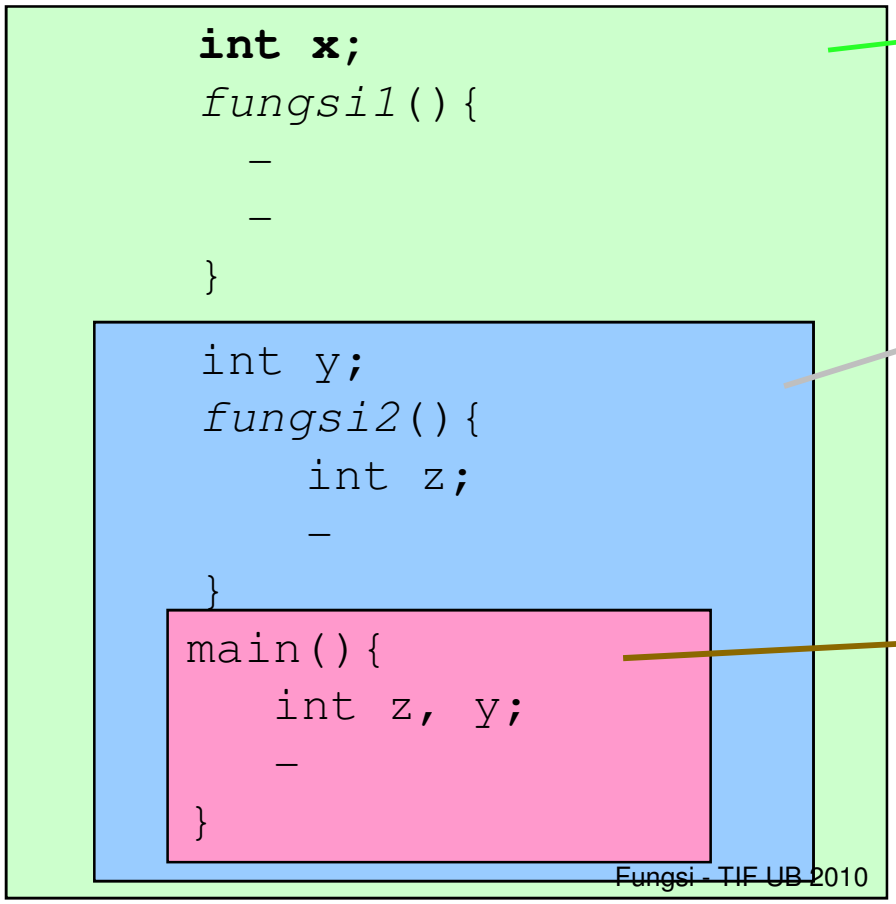
- Lingkup *identifier* meliputi bagian-bagian program dimana sebuah identifier masih bisa diakses.
- Lingkup *identifier* meliputi :
  - *Local*
  - *Global*
- *Local identifier*
  - Dideklarasikan di dalam fungsi, termasuk daftar parameter.
  - Lingkupnya terbatas pada fungsi tempat dideklarasikan.

Lingkup *Identifier*

- *Global identifier*
  - Dideklarasikan di luar fungsi
  - Ruang lingkupnya meliputi seluruh program
  - Dapat diakses dari fungsi-fungsi dalam program
  - Disarankan tidak banyak menggunakan identifier global karena:
    - Jika program semakin besar, kecenderungan error semakin besar .
    - Sulit melacak bila terjadi kesalahan.
    - Data tidak terjaga dengan baik, setiap fungsi dapat mengubah nilai variabel tanpa sepengetahuan fungsi lainnya.

Lingkup *Identifier*

- Contoh :



lingkup dari variabel x

lingkup dari variabel y

z dan y hanya dikenal oleh `main()`
z di `main()` berbeda dgn yang di *`fungsi2()`*
y di `main()` berbeda dgn yang di *`fungsi2()`*

Parameter Fungsi

- Pengiriman nilai data antar fungsi dapat dilakukan melalui penggunaan parameter fungsi.
- Pengiriman nilai data melalui parameter:
  - ***By value***

    Yang dikirim ke fungsi lain adalah nilai datanya
  - ***By reference/by location***

    Yang ditransfer ke fungsi lain adalah alamat memorinya

Pengiriman Parameter

- Contoh pengiriman parameter *by value*

```
#include <stdio.h>

void Garis (char x ) { // x sbg Parameter Formal
      int i; // i, x adalah Local Variabel
      for (i = 1; i<=10; i++)
            printf("%c",x);
}

/*Program Utama*/
void main(){
      char A = '-';
      Garis(A); // A disebut Parameter Aktual
}
```

Pengiriman Parameter

- Contoh pengiriman parameter *by reference*

```
#include <stdio.h>

void Hitung (int X, int Y, int *P, int *Q)
{       *P = X + Y;
        *Q = X * Y;
}

void main()
 {      int X, Y, P, Q; // local variables
        printf(" X="); scanf("%d",&X);
        printf(" Y="); scanf("%d",&Y);
        Hitung(X,Y,&P,&Q);
        printf("X + Y = %d\n", P);
        printf("X * Y = %d\n", Q);
}
```


Array sebagai Parameter

- Jika array digunakan sebagai parameter dalam suatu fungsi, maka passing parameter harus *by reference*.
- Contoh:

```
#include <stdio.h>
void cetak_array(int index, int *A) {
      printf("A[%d]=%d\n",index, A[index]);
}

void main() {
      int A[ ]={1,6,2,8,12};
      cetak_array(2, A);
}
```

Array sebagai Parameter

Array 2 Dimensi

- Deklarasi fungsinya dapat berupa:

```
void isimatriks(int a[10][10], int b, int k)
```

atau

```
void isimatriks(int a[][10], int b, int k)
```

- tetapi ***TIDAK*** bisa berupa:

```
void isimatriks(int a[10][], int b, int k)
```

atau

```
void isimatriks(int a[][], int b, int k)
```

Array sebagai Parameter

Array 2 Dimensi

Contoh:

```
#include <stdio.h>

void cetak(int A[3][4]){
   int row,col;
   for(row=0; row<3; row++){
       for(col=0; col<4; col++){
              printf("X[%d][%d]=%d",
                     row,col,A[row][col]);
              printf("\n");
       }
    }
}

int main(){
   int x[3][4] = {{1,2,3,4},{8,7,6,5},{9,10,11,12}};
   cetak(x);
   return(0);
}
```

Pengiriman Parameter

```
int main()
{   char ss[20]="KASUR";
    balik(ss);
    printf("%s\n",ss);
    getch();
    return(0);
}
```

Untuk string pada formal parameter bisa :

char[ ] atau char *

```
void balik( char ss[ ] )
{   int c,i,j;
    for(i=0, j=strlen(ss)-1;
        i<j; i++, j--){
        c=ss[i];
        ss[i]=ss[j];
        ss[j]=c;
    }
}
```

```
void balik( char *ss )
{   int c,i,j;
    for(i=0, j=strlen(ss)-1;
        i<j; i++, j--){
        c=ss[i];
        ss[i]=ss[j];
        ss[j]=c;
    }
}
```

Fungsi Rekursif

- Fungsi rekursif:

  Di dalamnya terdapat pernyataan yang memanggil dirinya sendiri.

- Berguna untuk memecahkan masalah yang dapat didefinisikan secara rekursif pula.

- Contoh :

  Faktorial (n) atau n! didefinisikan sebagai berikut :

  jika n = 0, n! = 1

  *jika n > 0, n! = n * (n-1)!*

  Contoh:

  4! = 4 * 3!
  3! = 3 * 2!
  2! = 2 * 1!
  1! = 1* 0!
  0! = 1

Jadi: 4! = 4*3*2*1= 24

Fungsi Iteratif

4! = 4*3! = 4*3*2! = 4*3*2*1! = 4*3*2*1 = 24

```
// iteratif dekremental
long faktorialIteratifDec(long n){
 long i, faktorial = 1;
   for(i=n; i>=1; i--)
        faktorial *= i;
   return faktorial;
}
```

4! = 1*2*3*4 = 24

```
// iteratif inkremental
long faktorialIteratifInc(long n){
  long i, faktorial = 1;
   for(i=1; i<=n; i++)
       faktorial *= i;
   return faktorial;
}
```

Fungsi Rekursif

- Contoh perhitungan 5 faktorial

5!
(5 * 4!)
(5 * (4 *3!))
(5 * (4 * (3 * 2!)))
(5 * (4 * (3 * (2 * 1!))))
(5 * (4 * (3 * (2 * (1 * 0!)))))
(5 * (4 * (3 * (2 * (1 * 1)))))
(5 * (4 * (3 * (2 * 1))))
(5 * (4 * (3 * 2)))
(5 * (4 * 6))
(5 * 24)
120

Fungsi Rekursif

- Fungsi rekursif mempunyai dua komponen yaitu:
  - ***Base case***:

    Mengembalikan nilai tanpa melakukan pemanggilan rekursi berikutnya.

    Rekursi berakhir jika *base case* dijumpai/dipenuhi
  - ***Recursion call / Reduction step***:

    Memanggil fungsi rekursif di dalam fungsi rekursif di atas

    Menghubungkan sebuah fungsi rekursif dengan fungsi rekursif di dalamnya

    Biasanya memiliki *keyword* **`return`** untuk mengembalikan nilai ke fungsi yang memanggilnya

Fungsi Rekursif

- Fungsi faktorial
  - Base case: n = 0
  - Reduction step: f(n) = n * f(n-1)

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

Rekursif vs Iteratif

- Contoh:

- ***Faktorial - Rekursif***

```
long faktorial(long n){
  if(n==0)
   return (1);
  else
   return(n*faktorial(n-1));
}
```

- ***Faktorial - Iteratif***

```
// dekremental
long faktorial(long n){
   long i, faktorial = 1;
   for(i=n; i>=1; i--)
       faktorial *= i;
   return faktorial;
}
```

Rekursif vs Iteratif

Rekursif

Pengulangan dengan struktur seleksi (if-else) dan pemanggilan fungsi (dirinya sendiri) -> rekursi

Pengulangan berhenti saat *base case* dijumpai/dipenuhi (konvergen terhadap base case)

Pengulangan **tanpa henti** jika *base case* **tidak pernah** dijumpai/dipenuhi (tidak konvergen terhadap base case)

Biaya proses lebih tinggi dengan pemanggilan banyak fungsi (butuh memori lebih besar & kerja prosesor lebih tinggi)

Terbaca lebih jelas, model lebih dekat dengan masalah (contoh: faktorial, fibonacci)

Iteratif

Pengulangan dengan struktur repetisi (for/while)

Pengulangan berhenti saat kondisi pengulangan bernilai salah (*false*)

Pengulangan **tanpa henti** jika kondisi pengulangan selalu benar

Biaya proses lebih rendah (kebutuhan memori lebih kecil & kerja prosesor lebih rendah) karena proses pengulangan berada dalam satu fungsi

Terbaca kurang jelas, model kurang dekat dengan masalah (contoh: faktorial, fibonacci)

Kekurangan Rekursi

- Meskipun penulisan program dengan cara rekursif bisa lebih jelas dan pendek, namun fungsi rekursif memerlukan :
  - Memori yang lebih banyak untuk mengaktifkan *stack* (memori yang digunakan untuk pemanggilan fungsi).
  - Waktu lebih lama untuk menjejaki setiap rekursi melalui *stack*.

Kapan Rekursi?

- Secara umum, hanya jika :
  - Penyelesaian sulit dilaksanakan secara iteratif
  - Efisiensi dengan cara rekursif masih memadai
  - Efisiensi bukan masalah dibandingkan dengan kejelasan logika program
  - Tidak mempertimbangkan faktor penghematan memori dan kecepatan eksekusi program

Kecepatan kerja dan penghematan memori (iteratif)

vs

Perancangan logika yang baik (rekursif)

Bilangan Fibonacci

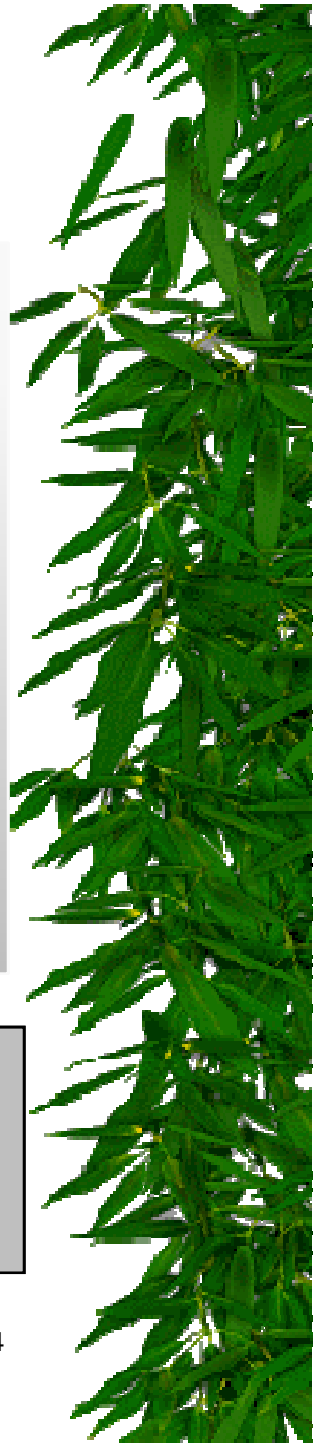
- Urutan bilangan 0, 1, 1, 2, 3, 5, 8, 13 … disebut bilangan Fibonacci. Hubungan antara satu angka dengan angka berikutnya didefinisikan secara rekursif sebagai berikut :
  - Fib(N) = N, jika N = 0 atau 1
  - Fib(N) = Fib(N-2) + Fib(N-1) , jika N >= 2

Bilangan Fibonacci

```
int Fib(int n) {
    int f;
    if(n==0)
       f = 0;
    else if(n==1)
       f = 1;
    else
       f = Fib(n-2) + Fib(n-1);
    return f;
}
```

Fungsi fib() di atas ditulis secara rekursif dan disebut sebagai slow_Fib()

Tulislah fast_Fib() jika menggunakan iterasi.



Bilangan Fibonacci

- Contoh : Skema fibonacci jika N=4