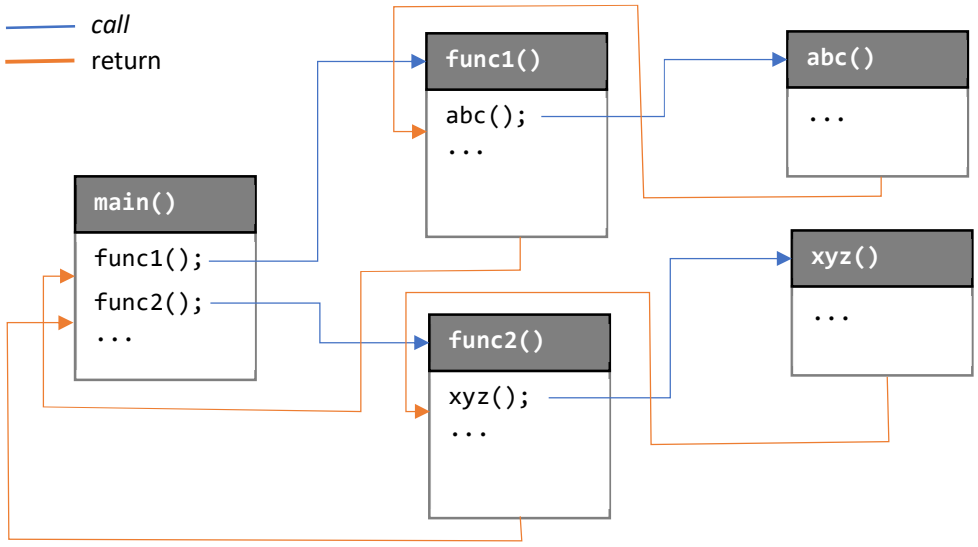
09 Fungsi

Ringkasan Materi

Pengertian

Saat membuat program; berdasar materi yang sudah pernah dipelajari sebelumnya; diketahui bahwa logika atau alur program ditulis pada sebuah blok yang disebut **main()**. Di dalam blok tersebut dituliskan berbagai macam perintah (selanjutnya disebut **fungsi**) sesuai tujuan. Sebagai contoh yang paling sering adalah penggunaan fungsi **cout** untuk menampilkan tulisan di layar, atau **getLine()** untuk membaca input dari *keyboard*.

Pada dasarnya sebuah program C++ adalah sekumpulan fungsi yang saling berinteraksi satu sama lain. Bahkan **main()** adalah sebuah fungsi. Sebagai contoh, fungsi **main()** memanggil fungsi **cout** untuk menampilkan tulisan di layar. Pada saat yang sama fungsi **cout** juga akan memanggil fungsi lainnya lagi agar bisa berjalan. Demikian seterusnya. Gambar 28 berikut ini menunjukkan interaksi antar fungsi (*function* call) ketika sebuah program berjalan.



Gambar 28 Function call

Penjelasan

Saat program dijalankan maka fungsi **main()** akan otomatis dijalankan. Dalam fungsi **main()** terdapat pemanggilan fungsi **func1()** dan **func2()**. Ketika **func1()** dipanggil alur program akan beralih ke fungsi tersebut yang di dalamnya terdapat pemanggilan fungsi **abc()** yang akan mengakibatkan alur program beralih ke fungsi **abc().** Setelah fungsi **abc()** selesai dijalankan, alur akan kembali ke fungsi **func1()** dan ketika fungsi **func1()** selesai dijalankan maka alur akan kembali lagi ke fungsi **main().** Demikian seterusnya, setiap satu fungsi selesai dipanggil (dijalankan) maka alur akan kembali ke fungsi pemanggilnya.

Catatan

Fungsi **main()** adalah fungsi utama yang wajib ada dari sebuah program C++. Fungsi ini hanya boleh ada 1 dalam sebuah program.

Struktur

Sebuah fungsi bisa diibaratkan sebagai sebuah mesin yang menjalankan pekerjaan secara spesifik. Terkadang memiliki input (parameter atau argumen) dan atau output (nilai kembalian atau *return value*). Berikut ini adalah struktur fungsi secara umum dalam bahasa C++:

```
tipe nama_fungsi([parameter])
{
        definisi
}
```

Penjelasan

- **tipe**, adalah tipe data kembalian (bila ada)
- **nama_fungsi**, nama fungsi dengan aturan seperti penamaan variabel
- **parameter** atau argumen, nilai input ke dalam fungsi. Parameter bisa lebih dari satu atau tidak ada sama sekali.
- **definisi**, berisi alur atau cara kerja fungsi sebagai bagian utama dari sebuah fungsi.

Berikut ini adalah contoh sebuah fungsi dengan nama **add()** untuk menjumlahkan 2 buah bilangan a dan b.

```
int add(int a, int b)
{
        int result = a + b;
        return a + b;
}
```

Coba identifikasilah struktur fungsi di atas!

Jenis

Berdasar nilai yang dikembalikan, fungsi dibagi menjadi 2, yaitu:

1. ***Non-Void***
   Fungsi yang mengembalikan nilai seperti **int**, **float**, atau tipe – tipe data yang lain. Di dalam fungsi ini ditandai dengan sebuah *statement* **return *nilai.*** Contoh yang sering dijumpai adalah pada fungsi **main()** di mana setiap akhir barisnya selalu ada **return 0.** Perhatikan kembali contoh fungsi **add()** berikut ini:

   ```
   int add(int a, int b)
   {
     int result = a + b;
     return a + b;
   }
   ```

   Tipe data kembalian pada fungsi harus sesuai dengan nilai yang dikembalikan (*return value*). Bisa diibaratkan *return value* adalah nilai (output) yang dikeluarkan dari sebuah fungsi.

2. ***Void***
   Fungsi ini tidak mengembalikan nilai sehingga tidak diperlukan *statement* **return** di akhir baris fungsi. Namun, untuk tipe kembalian tetap harus ada, yaitu **void** (kosong). Berikut ini adalah contoh fungsi **void()** untuk menampilkan ucapan salam dari nama orang sebagai parameter.

```
void greeting(string name)
{
   cout << "Hello, " << name << ". How are you?";
}
```

Walaupun fungsi void tidak mengembalikan nilai, dalam kasus tertentu bisa diberi *statement* **return** untuk keluar dari fungsi secara 'paksa' (*force quit*).

Rekursi

Rekursi adalah teknik pemrograman di mana **suatu fungsi memanggil dirinya sendiri** untuk menyelesaikan masalah yang lebih kecil dari masalah aslinya. Rekursi bergantung pada dua komponen utama:

- **kasus dasar** (*base case*) yang **menghentikan pemanggilan berulang**, dan
- **kasus rekursif** (recursive case) yang memecah masalah menjadi sub-masalah yang lebih sederhana.

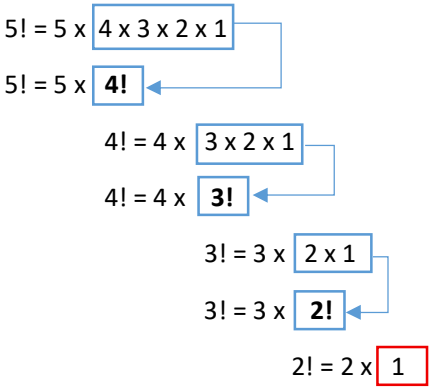
Konsep ini sering digunakan dalam algoritma seperti perhitungan faktorial, deret Fibonacci, pencarian biner, dan lain – lain.

Dalam kasus perhitungan faktorial di mana **n faktorial** yang disimbolkan dengan **n!** dapat diselesaikan dengan rumus:

n! = n x (n – 1) x (n – 2) x (n – 3) x .. x 1

n! = n x (n – 1)!

Contoh



Dalam contoh kasus faktorial, maka:

- *Base case*: **1,** di mana rekursi akan berhenti ketika **n = 1**
- *Recursive case*: **n! = n x (n – 1)!**

Bila diimplementasikan dalam fungsi, maka bentuknya adalah sebagai berikut:

```
int factorial(int n)
{
        if(n == 1)
                return 1;          // base case
        return n * factorial(n - 1); // recursive case
}
```

Percobaan

Factorial

Project ini mendemokan fungsi *non-void* untuk menghitung faktorial. Seperti telah dibahas pada contoh kasus rekursi di atas, faktorial dari **n** dapat dihitung menggunakan rumus **n! = n x (n-1) x (n-2) x ... x 1,** di mana **n** adalah bilangan bulat positif. Tipe data untuk hasil perhitungan faktorial adalah **unsigned long long int**. *Keyword* **unsigned** menandakan bahwa nilai data tidak memiliki tanda (-negatif); atau bisa dikatakan positif. Sedangkan *keyword* **long** di depan **int** (bilangan bulat) menandakan ukuran data yang bisa disimpan.

Berikut ini adalah kode selengkapnya:

```
#include <iostream>

using namespace std;

unsigned long long factorial(int n);

int main()
{
	int number;
	cout << "Input: ";
	cin >> number;

	// jika input tidak valid (bilangan negative)
	// tampilkan pesan dan keluar dari program
	if(number < 0)
	{
		cout << "Invalid input\n";
		return 1;
	}

	unsigned long long result = factorial(number);
	cout << "Factorial of " << number << ": " << result << endl;

	system("pause");

	return 0;
}

// fungsi factorial()
// menghitung faktorial dari sebuah bilangan bulat non-negatif
// menggunakan pendekatan iteratif
// parameter: n - bilangan bulat positive
// return: faktorial dari n

unsigned long long factorial(int n)
{
	// faktorial dari n (n > 1) adalah n * (n-1) * ... * 2 * 1
	unsigned long long result = 1;
	for (int i = n; i > 1; --i) {
		result *= i;
	}

	return result;
}
```

Cobalah ganti kode pada fungsi **factorial()** menggunakan teknik rekursi seperti yang telah dibahas sebelumnya.

Pyramid

Demo fungsi *non-void* untuk menggambar piramida. Fungsi memiliki sebuah parameter untuk menentukan tinggi piramida. Berikut ini adalah kode selengkapnya:

```
#include <iostream>

using namespace std;

void pyramid(int height);

int main()
{
	int height = 5;
	pyramid2(height);

	system("pause");

	return 0;
}

// fungsi pyramid()
// menggambar piramida menggunakan karakter ASCII 178
// parameter: height – tinggi piramida
// return: void

void pyramid(int height)
{
	for(int i = 1; i <= height; i++) {
		// Cetak spasi sebelum bintang
		for (int j = 0; j < height - i; j++) {
			cout << " ";
		}
		// Cetak karakter ASCII 178
		for (int k = 0; k < (2 * i - 1); k++) {
			cout << char(178);
		}
		cout << endl;
	}
}
```

Cobalah modifikasi juga fungsi **pyramid()** di atas menggunakan teknik rekursi!

Distance

Project ini mendemokan sebuah fungsi bernama **dist()** untuk menghitung jarak 2 benda. Perhitungan berdasar posisi koordinat objek 1 (x1, y1) dan objek 2 (x2, y2) menggunakan rumus sisi miring segitiga. Rumus sisi miring segitiga dapat dihitung menggunakan fungsi **hypot()** yang terdapat pada header **<cmath>**. Berikut ini contoh kode lengakpnya:

```
#include <iostream>
#include <cmath>

using namespace std;

double dist(double x1, double y1, double x2, double y2);

int main()
{
	double x1, y1, x2, y2;

	cout << "Objek 1 (x1 y1): ";
	cin >> x1 >> y1;
```

```
        cout << "Objek 2 (x2 y2): ";
        cin >> x2 >> y2;

        cout << "Jarak kedua objek: " << dist(x1, x2, y1, y2) << endl;

        system("pause");

        return 0;
}

// fungsi dist()
// menghitung jarak antara dua titik (x1, y1) dan (x2, y2)
// menggunakan fungsi hypot() dari pustaka cmath
// parameter: x1, y1, x2, y2 - koordinat kedua titik
// return: jarak antara kedua titik

double dist(double x1, double y1, double x2, double y2)
{
        double deltaX = x2 - x1;
        double deltaY = y2 - y1;

        return hypot(deltaX, deltaY);
}
```

Fungsi di atas dapat diimplementasikan untuk pendeteksi tumbukan (*colision detection*) sederhana pada game 2D.

Latihan

1. Buatlah fungsi dengan nama **encrypt()** untuk mengenkripsi (menyandikan) sebuah kalimat dengan cara mengubah semua karakter vokal menjadi ***'*'*** (asterik). Setelah itu cobalah fungsi tersebut dalam fungsi **main()** seperti contoh berikut ini:

```
string result = encrypt("Yogyakarta");
cout << "Result: " << result << endl;
```

akan menghasilkan "**Y*gy*k*rt***"

2. Ubahlah fungsi **pyramid()** pada *project* **Pyramid** di atas menjadi 2 parameter. Parameter pertama (seperti sebelumnya) adalah untuk tinggi, dan parameter kedua adalah untuk variasi bentuk. Kententuan untuk parameter kedua; bila:
   a. Bernilai **0**, bentuk piramida **rata kiri**
   b. Bernilai **1**, bentuk piramida **rata kanan**
   c. Bernilai **2**, bentuk piramida lancip di tengah (***center***)
   d. Selain 3 nilai tadi, bentuk piramida rata kiri.

   Berikut ini adalah contoh penggunaan dan hasilnya:

pyramid(5, 0)

```
*
**
***
****
*****
```

Rata kiri

pyramid(5, 1)

```
    *
   **
  ***
 ****
*****
```

Rata kanan

pyramid(5, 2)

```
    *
   ***
  *****
 *******
*********
```

Center