TIF1201 – Pemrograman Berorientasi Objek

HO 03 - Constructor dan Destructor

Opim Salim Sitompul

Department of Information Technology
Universitas Sumatera Utara

Outline

Tujuan

- Dalam kuliah ini mahasiswa diharapkan:
  - Mengenal konsep constructor sebagai mekanisme pemberian harga awal pada objek
  - Mengenal konsep default constructor
  - Mampu menerapkan constructor dalam pendefinisian kelas
  - Mengenal konsep destructor sebagai mekanisme pengembalian alokasi memori yang diberikan pada objek
  - Mengenal konsep multipel constructor dalam memberikan harga awal kepada objek melalui argumen

Pendahuluan

- *Constructor* dan *Destructor* dideklarasikan di dalam kelas berupa anggota fungsi.
- Fungsi *constructor* membantu program melaksanakan alokasi memory secara dinamis pada objek yang diturunkan dari sebuah kelas dimana fungsi itu berada.
- Fungsi *destructor*, sebaliknya, mengembalikan memori yang dialokasikan ke sebuah objek pada saat objek tersebut keluar dari ruang lingkup (*scope*), atau apabila pointer ke sebuah objek di *delete*.

Pendahuluan

- Fungsi *constructor* secara otomatis akan dipanggil oleh kompiler bilamana satu objek diturunkan dari kelas dimana constructor itu didefinisikan.
- Fungsi *constructor* memberikan harga awal kepada anggota variabel setelah memori dialokasikan kepada objek.

Mendefinisikan Constructor

- Berbeda dengan anggota-anggota fungsi yang telah dikenal sebelumnya, fungsi *constructor* memiliki beberapa ciri khusus.
  - Fungsi constructor tidak mengembalikan nilai dan tidak berjenis void.
  - Menggunakan nama yang sama seperti nama kelas.
- Contoh:

```
class MyClass
{
    private:
      int counter;
    public:
      MyClass() { counter = 0; }
      ...
};
```

Mendefinisikan Destructor

- Fungsi *destructor* dijalankan secara otomatis ketika objek dibuang dari memori.
- Fungsi *destructor* juga memiliki nama yang sama seperti nama kelas, tidak mengembalikan nilai, dan tidak berjenis void.
- Fungsi *destructor* tidak menerima argumen.
- Fungsi *constructor* dan *destructor* dibedakan oleh karakter tilde (~).

Mendefinisikan Destructor

- Di dalam kelas hanya terdapat satu buah fungsi *destructor*.
- Definisi fungsi *destructor* cukup dibiarkan kosong, kompiler secara otomatis akan menjalankan fungsi ini tanpa ada pemanggilan fungsi secara eksplisit.
- Contoh:

```
class MyClass
{
    private:
      int counter = 0;
    public:
      ~MyClass() { }
      ...
};
```

Default Constructor

- Default constructor adalah sebuah fungsi constructor yang tidak menerima argumen.
- Default constructor dipanggil pada saat sebuah objek diturunkan dari sebuah kelas.
- Default constructor biasanya disertakan di dalam sebuah kelas untuk memberikan harga awal pada anggota-anggota variabel dari objek yang diturunkan dari sebuah kelas tersebut.

Default Constructor

- Contoh:
  MyClass oneObject;
- Objek oneObject pada MyClass seperti pada contoh definisi kelas yang telah diberikan memiliki anggota variabel bernama *counter*.
- Anggota variabel *counter* akan diberi harga awal 0 oleh kompiler.

Default Constructor

- Contoh Program:

```
1  // Contoh3_1.cpp
2  #include <iostream>
3  using namespace std;
4  class MyClass
5  {
6    private:
7      int counter;
8    public:
9      MyClass() {counter = 0;}
10     void incrCounter(){counter++;}
11     void showCounter();
12     ~MyClass() {cout << "This␣statement␣is␣"
13             << "automatically␣called..."
14   << endl;}
15 };
```

Default Constructor

```
1  void MyClass::showCounter()
2  {
3    cout << "Counter is " << counter << endl;
4  }
5
6  int main()
7  {
8    MyClass myCounter;
9
10   myCounter.showCounter();
11   myCounter.incrCounter();
12   myCounter.showCounter();
13
14   return 0;
15 }
```

Constructor Berparameter

- Constructor berparameter adalah fungsi constructor yang memiliki argumen.
- Melalui argumen inilah fungsi constructor akan memberikan harga awal kepada anggota-anggota variabel milik sebuah objek yang diturunkan dari sebuah kelas.
- Contoh Progam:

Constructor Berparameter

```
1  // Contoh3_2.cpp
2  #include <iostream>
3  using namespace std;
4
5  class Number
6  {
7    private:
8      int value;
9    public:
10     Number(int aValue);
11     void getNumber();
12     void showNumber();
13     ~Number() {}
14 };
```

Constructor Berparameter

```
1  Number::Number(int aValue)
2  {
3    value = aValue;
4    cout << "Got " << value << endl;
5  }
6
7  void Number::getNumber()
8  {
9    cout << "Enter an integer: ";
10   cin >> value;
11 }
```

Constructor Berparameter

```
1  void Number::showNumber()
2  {
3    cout << "You␣entered␣" ;
4    cout << value << endl;
5  }
6
7  int main()
8  {
9    Number yourNumber(10);
10
11   yourNumber.getNumber();
12   yourNumber.showNumber();
13
14   return 0;
15 }
```

Copy Constructor

- Copy constructor adalah satu jenis constructor yang digunakan untuk menghasilkan sebuah salinan dari objek yang sudah ada dari suatu jenis kelas.
- Karena digunakan untuk membuat sebuah objek, maka disebut constructor. Sementara itu, karena objek baru yang dibuat itu adalah salinan langsung dari objek yang sudah ada, maka disebut copy constructor.
- Bentuk umumnya adalah X (X&), dimana X adalah nama kelas.

Copy Constructor

- Contoh:
  Namakelas(const Namakelas & namaobjek)
  {
  . . .
  }
- Kompiler memberikan sebuah default Copy Constructor ke semua kelas.
- Contoh Program:

Copy Constructor

```
1  //Contoh3_3.cpp
2  #include<iostream>
3  using namespace std;
4
5  class Myconstructor
6  {
7    private:
8      int x, y;    //data members
9    public:
10     Myconstructor(int, int);
11     Myconstructor(const Myconstructor &);
12     void display();
13 };
```

Copy Constructor

```
1  /* Copy constructor */
2  Myconstructor::Myconstructor(int x1, int y1)
3  {
4    x = x1;
5    y = y1;
6  }
7
8  Myconstructor::Myconstructor(const
       Myconstructor &sam)
9  {
10   x = sam.x;
11   y = sam.y;
12 }
```

Copy Constructor

```
1  void Myconstructor::display()
2  {
3    cout<<x<<"␣"<<y<<endl;
4  }
5
6  int main()
7  {
8    Myconstructor obj1(10, 15);//Normal constr
9    Myconstructor obj2 = obj1;//Copy constr
10   cout<<"Normal␣constructor␣:␣";
11   obj1.display();
12   cout<<"Copy␣constructor␣:␣";
13   obj2.display();
14
15   return 0;
16 }
```

Copy Constructor

- Contoh Program:

```
1  // Contoh3_4.cpp
2  #include <iostream>
3  using namespace std;
4
5  class Number
6  {
7    private:
8      int value;
9    public:
10     Number(int);
11     Number(const Number &);
12     void getNumber();
13     void showNumber(string);
14     ~Number() {}
15 };
```

Copy Constructor

```
1  Number::Number(int aValue)
2  {
3    value = aValue;
4  }
5
6  Number::Number(const Number &aCopy)
7  {
8    value = aCopy.value;
9    cout << "Got " << value << endl;
10 }
```

Copy Constructor

```
1  void Number::getNumber()
2  {
3    cout << "Enter an integer: ";
4    cin >> value;
5  }
6
7  void Number::showNumber(string txt)
8  {
9    cout << txt ;
10   cout << value << endl;
11 }
```

Copy Constructor

```
1  int main()
2  {
3    Number yourNumber(10);
4    Number myCopy = yourNumber;
5
6    yourNumber.getNumber();
7    yourNumber.showNumber("Your new number: ");
8
9    myCopy.showNumber("Copy of your number: ");
10
11   return 0;
12 }
```

Multiple Constructor

- Di dalam kelas terdapat beberapa constructor yang memiliki argumen-argumen yang berbeda.
- Pada waktu menurunkan objek, kompiler menentukan constructor mana yang dipanggil, berdasarkan jumlah parameter yang diberikan.
- Contoh Program:

Multiple Constructor

```
1 //Contoh3_5.cpp
2 #include <iostream>
3 #include <iomanip>
4 #include <cstdlib>
5 #include <ctime>
6 #define N 100
7
8 using namespace std;
```

Multiple Constructor

```
1  class Array
2  {
3    private:
4      int array_integer[N];
5      float array_float[N];
6    public:
7      Array(int);
8      Array(double);
9      long jlh_Array(int);
10     float jlh_Array(double);
11     void print_array(int);
12     void print_array(double);
13     ~Array() { }
14 };
```

Multiple Constructor

```
1  float Array::jlh_Array(double type)
2  {
3    float jlh = 0.0;
4
5    for(int i=0; i < N; i++)
6      jlh += array_float[i];
7    return jlh;
8  }
```

Multiple Constructor

```
1  long Array::jlh_Array(int type)
2  {
3    long jlh = 0L;
4
5    for(int i=0; i < N; i++)
6      jlh += array_integer[i];
7    return jlh;
8  }
```

Multiple Constructor

```
1  void Array::print_array(double type)
2  {
3    cout << "Array Float: " << endl;
4    for(int i=0; i < N; i++)
5      cout << setprecision(2)
6        << setiosflags(ios::fixed)<< array_float
             [i] << " ";
7    cout << endl;
8  }
```

Multiple Constructor

```
1  int main()
2  {
3    Array myArr1(0), myArr2(0.0);
4
5    myArr1.print_array(0);
6    cout << "Jumlah array integer = "
7      << myArr1.jlh_Array(0) << endl;
8    myArr2.print_array(0.0);
9    cout << "Jumlah array float = "
10     << setprecision(2)
11     << setiosflags(ios::fixed)
12     << myArr2.jlh_Array(0.0)
13     << endl;
14   return 0;
15 }
```