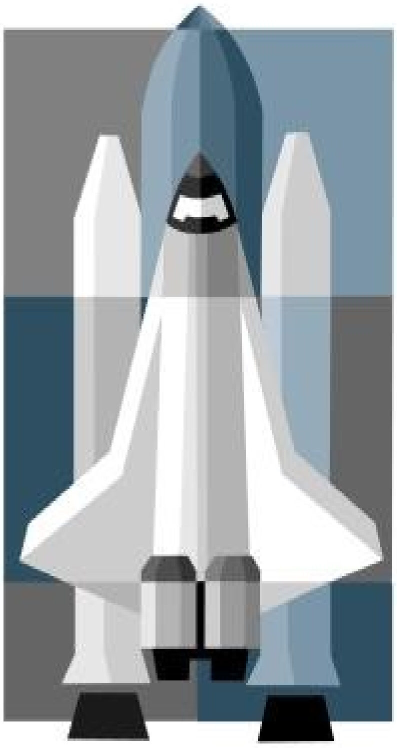
CHAPTER 9 APLICATION PROGRAM C++

Figure 9.1 may raise question. What is its relation with the above chapter title? Do you know that the space shuttle in Figure 9.1 is controlled by a sophisticated computer with most of its applications written in C++ programming language.

The standard competence in application programming using C++ consists of five (5) basic competences. In this book, each basic competence contains the objectives, material, and exercise. The summary is at end of each chapter. Basic competence of this chapter is the foundation of C++ programming, applying function, pointer, class concept, and designing an object oriented programming. Please review chapter on operating system, algorithm and concept of object oriented programming in previous chapters.

In the final section, exercise is compiled from the easiest to the more difficult exercise. These exercises are used to measure the capacity on basic competence in this chapter. It is suggested to do these exercises at school under your teacher's guidance or at home.



(Sumber: Clip Art Microsoft Office 2007)

Figure 9.1. Space Shuttle.

THE OBJECTIVES

After studying this chapter, the reader is hoped to be able to,

- Explain the basic of C++ programming.
- Applying function.
- Applying pointer.
- Applying class concept.
- Designing an object oriented application.

9.1. BASIC C++ PROGRAMMING

C++ is a popular language in the software developing world. Like its predecessor the C Language, C++ is a middle level language. The main objective of C++ is to increase the programmer productivity in application creation. C++ is of interest as it supports object oriented programming. Moreover, there are many books / library that could be used to speed up the application creation. Some of these libraries are freely available on the Internet.

Source code in C++ may not be directly used. We have to compile it using approriate C++ compiler. Compilation and execution concept of C++ program is shown in Figure 9.2. The source code is a text file with extension .cpp. The source code is put into PreProcessor. The PreProcessor output file is for the Compiler. Compiler will translate the PreProcessor output file into assembly language. This code is then processed by the Assembler into object code. If no library is used, the object code may be directly executed into a file. If library is used, Link Editor will combine the object code and the library to create the EXE file.

We may type the C++ code in any text editor, such as, Notepad, vi, etc. However, it would be much easier if we could use IDE (Integrated Development Environment) tool that integrates text editor and C++ compiler. Some of the famous C++ are Microsoft Visual the Studio, Borland C++, MingGW Developer Studio, etc.In this book, most of the code is written using MinGW Developer the Studio as shown in Figure 9.3. The licence of MinGW IDE is free. Thus, we could use it without worrying about the software license. MinGW may be downloaded from http://visual-mingw.sourceforge.net/.

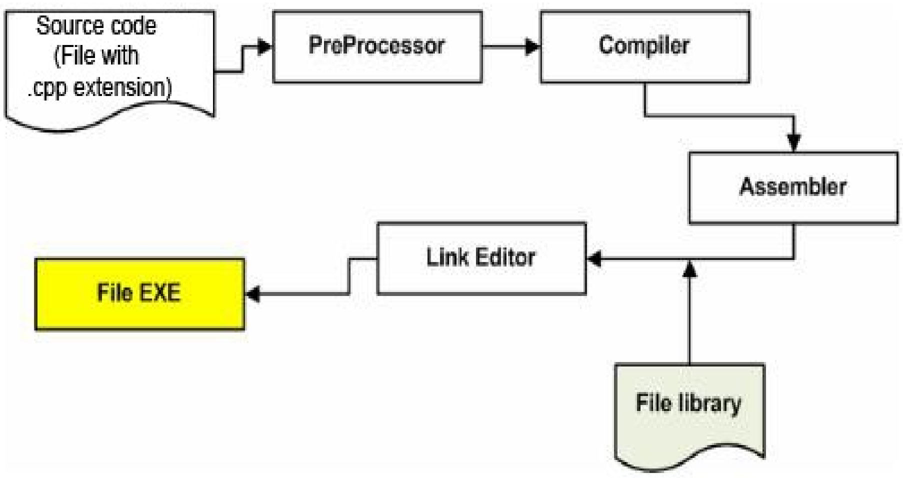
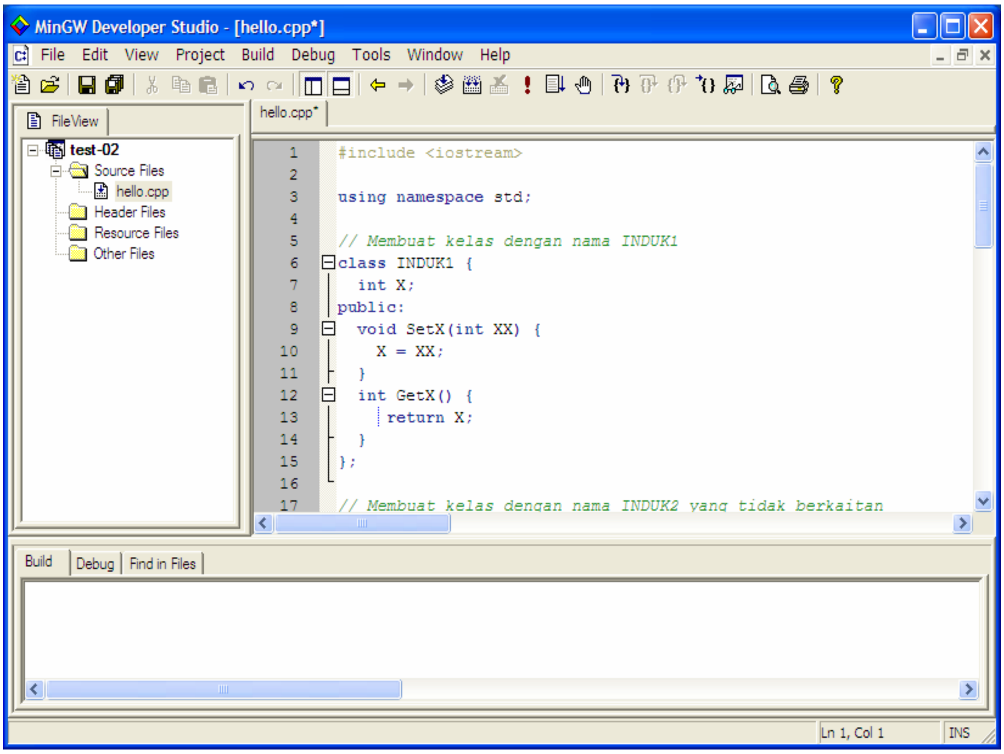


Figure 9.2. Compilation processes in C++



```
#include <iostream>

using namespace std;

// Membuat kelas dengan nama INDUK1
class INDUK1 {
  int X;
public:
  void SetX(int XX) {
    X = XX;
  }
  int GetX() {
    return X;
  }
};

// Membuat kelas dengan nama INDUK2 yang tidak berkaitan
```

Figure 9.3. MingGW Developer Studio

9.1.1. Structure of C++ Program.

```
#include <iostream>

using namespace std;

void nama_fungsi() {
   //kode untuk nama_fungsi
     .....;
   }

// Fungsi utama
int main() {

   // kode bagian main/utama
   .....;

   return 0;
}
```

Please examine the above source code. The code consists of several sections, namely,

- Section to include libraries or setting a certain condition. It uses a statement starts with #.
- Section to define a function. A function starts with void and follows by function name. This section may or may not be used to create a function.
- Section main(). The main program code starts with int main(). This section must be available in all program as it is the main program.
- Section starts with { and end with }, in

void as well as in main, is called block code.

9.1.2. Header File (.h)

Header file is a file with extension. h that contains certain functions that has been previously compiled. This file is usually contained files / library to be included in our source code / program. For example file iostream.h has some functions to show output and capture input such as cout and cin.

There are two writing file header models, namely, write complete with extension and not. First model is for older C++ compiler as Example 9.1. The Second model is a standard C++ compiler as in Example 9.2.

Example 9.1. Writing file header for older C++ Compiler.

```
#include <iostream.h>
Int main() {
        ...
        return 0;
}
```

Example 9.2. Writing file header for standard C++ Compiler.

```
#include <iostream>
using namespace std;
Int main() {
        ...
       return 0;
}
```

9.1.3. Comment, Identifier and Data Type

- **Comment**

Comment in C++ may be written in two format, namely, starts with // sign or put the comments start with /* and close with */. First method is used if comment only one line, whereas second method if comment more than one line.

Example 9.3. Comment with sign //.

```
// first program
#include <iostream>
int main( ){
```

```
cout << "Hello World"; // print “Hello World”  on screen
}
```

Example 9.4. Comment with sign /* .. */..

```
/* first program
Written by ARM
Date 17-11-2007 */

#include <iostream>
int main( ){
std::cout << "Hello World"; // print “Hello World” on screen
}
```

- **Identifier**

Identifier or variable name or constant in C++ is generally followed the naming rules as described in Chapter 5. There is one important note that C++ identifier is case sensitive. Thus, varible techername and TeacherName is different.

There are two ways to declare a constant. Firstly, we use preprocessor directive #define. Secondly, we use the key word const. Type the following program and check execution result.

Example 9.5. Constant Declaration

```
#include <iostream>
#define cut 0.1;
using namespace std;
int main() {
    const float pricePerUnit = 2500;
    int totalUnit;
    float priceTotal, priceDiscount, discount;
    cout << "Input total purchasing unit : ";
    cin >> totalUnit;
    priceTotal = totalUnit * pricePerUnit;
    discount = priceTotal * cut;
    priceDiscount = priceTotal - discount;
    cout << "Total price purchasing = " << priceTotal <<  endl;
    cout << "Discount = " << discount << endl;
    cout << "Price Discount = " << priceDiscount;
    return 0;
}
```

In the above example, there are two constants, namely, cut and pricePerUnit which are differently declared.

Variable declaration is firstly done by setting the data type followed by variable name as in Figure 9.5. There are some defined variables, namely, totalUnit, priceTotal, priceDiscount and discount. For variable with the same data type, it may be declare in one line separated by comma. Examine the above example, priceTotal, priceDiscount and discount have same float data type.

You should completely declare all variables or constants prior to use. If not then the program will not be executed.

- **Data type**

As explain in Chapter 5, data type depends on the programming language. The basic data type in C++ is in the following table. The data type is fairly similar to Java. This is due to Java takes some elements programming language from C++.

Table 9.1. Data type in C++

Data Type	Describes
Int	Integer data type with 4 bytes width
long	Integer type but larger than int
float	Data type in fraction
double	Data type in fraction but larger than float
char	Data type in character with letter, numeric, symbols (alphanumeric) as long as within sign " " or ' '.
bool	Data type boolean
short	Data type in integer with 2 bytes width

Data structure in C++ are array, struct and enum. Array will explain an other section in this chapter. While struct and enum has been described in Chapter 5. In this chapter, we will describe the syntax to write these structure. Please type the following source code and see the execution result.

Example 9.6. Using struct data type

```
#include <iostream>
using namespace std;
int main() {

  struct Teacher {
    char* NIP;
    char* Name;
    char* Address;
  };

 Teacher A;
  A.NIP = "132 232 477";
  A.Name = "Syafiq";
  A.Address = "Perum. Dirgantara Permai";

  // Showing score – fill in to screen
  cout<<A.NIP<<endl;
  cout<<A.Name<<endl;
  cout<<A.Alamat<<endl;
  return 0;
}
```

Example 9.6. Using enum data type

```
#include <iostream>
using namespace std;
enum SEX { Man, Woman };
int main() {

  struct Teacher {
    char* NIP;
    char* Name;
    SEX JK;
  } A;

  A.NIP = "132 232 477";
  A.Name = "Syafiq";
  A.JK = Man;

  cout<<"NIP      : "<<A.NIS<<endl;
  cout<<"Name     : "<<A.Name<<endl;
  cout<<"Sex      : "<<A.JK<<endl;

   return 0;
}
```

9.1.4. Operator

Similar to VB and Java, C++ also provides operators to help solving problems. There are fairly similar operators between Java and C++. The available operator in C++ is shown in Table 9.2.

Table 9.2. Operator in C++

Operator Type	**Function**	**Example**
Operator Assignment		
'=	Assign value into a variable	C = 5
Unary Operator		
+	Make positive value	X = + 10
-	Make negative value	Y = - 12
++	Increase variable one value	++C (pre-increment) C++ (post-increment)
- -	Decrease variable one value	--C (pre-increment) C-- (post-increment)
Binary Operator		
+	Adding	3 + 5 = 8
-	Substitute	7 – 2 = 5
*	Multiplying	5 * 2 = 10
/	Dividing	6 / 3 = 2
%	Modulus	5 % 2 = 1
Logical Operator		
&&	AND	1 && 1 = 1
II	OR	1 II 0 = 1
!	NOT	10 = 1
Relational Operator		
>	Bigger	(5 > 4) = 1
<	Smaller	(5 < 4) = 0
>=	Bigger or equals	(4 > = 4) = 1
<=	Smaller or equals	(5 < = 4) = 0
'==	Equals	(5 == 4) = 0

!=	Unequals	(5 ! = 4) = 1
Bitwise Operator		
&	AND	1 & 0 = 0
I	OR	1 I 0 = 1
^	XOR	1 ^ 1 = 0
~	NOT	~0 = 1
Ternary Operator		
?:	Use if it involves 3 operators	

9.1.5. Control Program Structure

C++ control program structure is fairly the same as VB and Java. The difference is only in its syntax. For branching, C++ provides if (without then) and switch ... case. For repetition, C++ provides for, while, do-while. In addition, C++ provides break and continue facilities for loop. Below is some example in applying control program structure. Type in the code, and observed the execution result.

Example 9.7. The use of simple if branching

```
#include <iostream>
using namespace std;
int main() {
   int bil;

   cout<<"input an integer : ";
   cin>>bil;

   // To check variable by operator modulus
   if (bil % 2 == 0) {
     cout<<bil<<" is  even number" << endl;
   } else {
     cout<<bil<<" is odd number" << endl;
   }
   return 0;
}
```

In the above example, we use if to check whether a number is even or odd. Modulus (%) operator is used. Examine how to write the structure if and else.

Example 9.8. Using if branch structure for three (3) conditions

```
#include <iostream>
using namespace std;
int main() {
   int bil;

   cout<<"Input an integer : ";
   cin>>bil;

   if (bil > 0) {
     cout<<bil<<" is POSITIF number";
   } else if (bil < 0) {
     cout<<bil<<" is NEGATIF number";
   } else {
     cout<<"is ZERO number";
   }
   return 0;
}
```

Example 9,8. is a development from Example 9.7. Branching structure is developed for three (3) conditions by adding else. For more than three (3) or many conditions, C++ provides switch ... case command to facilitate the selection process. Examine the following example.

Example 9.9. Using branch structure switch ... case.

```
#include <iostream>
using namespace std;
int main() {
   int bil;

   cout<<"Input a number (1 - 5) : ";
   cin>>bil;

   switch (bil) {
       case 1 : cout<<"Your number is : ONE";
                break;
       case 2 : cout<<"Your number is : TWO";
                break;
       case 3 : cout<<"Your number is : TIGA";
                break;
       case 4 : cout<<"Your number is : FOUR";
```

```
            break;
        case 5 : cout<<"Your number is : FIVE";
            break;
        default : cout<<"Your input over capacity";
    }
    return 0;
}
```

Example 9,9. uses switch for branching and convert number into text. Examine how to write switch and case.

As in Java, looping with for may be used if we know exactly how many repetition will be done. The following is a loop example with for.

Example 9.10. Loop structure with for.

```
#include <iostream>
using namespace std;
int main() {
  int C, J;

  cout<<"Print number from small to big :"<<endl;
  for (C=0; C<10; C+3) {
    cout<<C+1<<endl;
  }

  cout<<endl;
  cout<<"Print number from big to small"<<endl;
  for (J=10; J>0; J--) {
    cout<<J<<endl;
  }
  return 0;
}
```

Example 9.10. is a simple case on using for. There are two (2) for loop, one is to print 1 to 10 and from 10 to 1. Note the use of increment ++ and decrement --. The syntax / written method is fairly similar to Java. In Example 9.11 the loop using for is expanded by using nested for,.

Example 9.11. Loop structure using nested for.

```
#include <iostream>
using namespace std;
```

```
int main() {
  for (int j=1; j<=4; j++) {
    for (int k=1; k<=3; k++) {
        cout<<k+j<<' ';
    }
    cout<<endl;
  }
  return 0;
}
```

There are two loops in Example 9.11. The first loop uses j as variable counter, the second loop uses variable counter k that nested to loop j. What should be the output of the program?

Use while in a loop, is not that different from Java and VB. Please examine Example 9.12 and 9.13.

Example 9.12. Loop structure using while command.

```
#include <iostream>
using namespace std;

int main() {
  int C;
  C = 1;              // initialization value C

  while (C<10) {
    cout<<"I am not naughty anymore"<<endl;
    C++;              // increment
  }
  return 0;
}
```

Example 9.13. Loop structure using nested while.

```
#include <iostream>
using namespace std;

int main() {
  int x, y;
  x = 1;              //initialization variabel x

  while (x<=4){
   y = 1;             //initialization variabel y
```

```
    while (y<=3){
      cout<<y+x<<' ';
      y++;
      }
    cout<<endl;
    x++;
  }
  return 0;
}
```

Examine the above Example 9.12, what do you think output from the program? Compare it with Example 9.11. Try to change initialization for variable y. Put it after after initialization of variable x. How is the results?

Another from of loop in C++ is the do-while. A little bit different from while, condition check in do-while is at the end of the loop body. Example 9.14 gives explanation how to use a do-while in in C++.

Example 9.14. The use of do-while structure.

```
#include <iostream>
using namespace std;

int main() {
  int J = 5;
  int K;
  do {
     K = 1;
     do {
       cout<<K*J<<' ';
       K++;
     } while (K <= J);
     cout<<endl;
     J--;
  } while (J >= 1);
  return 0;
}
```

Example 9.14 shows a quite complicated source code. As we examine more closely, it is basically a loop with nested do while. Examine how to write and logical flow of the program. The outer do-while uses variable counter J and will loop from the large number to small as we can see J is initialized with 10 while J > =1. While the inner do-while uses variable counter K and loop from the small to big. How is the execution results of the source code? Examine the following output. Please examine the source code to really understand why the output program becomes like this.

```
5  10 15 20 25
4  8 12 16
3  6 9
2  4
1
```

9.1.6. Input / Output

So far, we have done a significant amount of programming exercises using C++. However, we have not yet learn the input / output statement in C++. Although, if you noticed the input / output statement has been indirectly studied. cout and cin command are the mostly used input/output statement.

cout and cin commands are classified as stream and included in iostream class. Thus, we need to include header file iostream in the beginning of a program to use these two (2) commands. Stream is a logical device to get or to put data. Stream connects with hardware such as keyboard, screen monitor, printer via I/O system.

Command cin is stream for input standard. This command will get anything we type on keyboard. Examine how to write it in Example 9.15.

Example 9.15. The use of cin and cout.

```
#include <iostream>
using namespace std;

int main() {
   int bil1, bil2;
   //cin part one
   cout<<"In put first number : ";
   cin>>bil1;
   cout<<"In put second number : ";
   cin>>bil2;
   cout<<"Result multiply both number = "<<bil1*bil2<<endl;
   //cin part two
   cout<<"In put two numbers : ";
   cin>>bil1>>bil2;
   cout<<"Result multiply both number = "<<bil1*bil2<<endl;
   return 0;
}
```

cin command may be used to get character one by one as in Example 9.15, please examine section below //cin part one, or get directly the data in sequence as shown in section below //cin part two. Command cin must be followed by operator >>.

cout command will do a standard output to monitor screen. cout command must be followed by operator <<. Examine the above Example 9.15. cout command may be used to directly print character in between two (2) marked " " or content variable. As in cin, cout command may be used to display character one by one or at the same time. In the above Example, the endl statement is used to print new line.

9.2. FUNCTIONS IN C++

Functions in C++ has an important role. As C++ code is basically a collection of functions. All available functions as well as the one we make, may be called from the main() function. As all programming languages, C++ provides built-in function and may be accessed by including the header file in the beginning of the C++ Program. C++ also provides facilities to create user-defined functions. This section will explain how to create a user-defined function.

9.2.1. Function Types

There are two (2) types of function, namely, function without return value and function that returns value.

- **Function without return value.**

This function is a void type. In VB or Pascal, this function is known as procedure. Examine the Example 9.16.

Example 9.16. Function without return value.

```
#include <iostream>
using namespace std;

// Making function print-out number
void Print-outNumber() {
  for (int C=0; C<10; C++) {
    cout<<C+1<<endl;
  }
}
// Main function in program C++
int main() {
  // Calling funcion Print-outNumber
  Print-outNumber();
  return 0;
}
```

In the above example, we create a function, namely, Print-outNumber with type void so no return value. Examine how to declare the function. The function will run up to end of function code.

- **Function with return value.**

This function will return a value to be used by other program. To define this function, we don't use void, use data type of the returned value. Examine Example 9.17.

Example 9.17. Function with return value

```
#include <iostream>
using namespace std;

// Making function with return value by char type
char* CharValue() {
   return "Value to be return";
}
// Main function

int main() {
   // Calling and viewing result function
   cout<< CharValue();
   return 0;
}
```

In type of function, we need a return statement to show section that returns value. In the above Example, the return value data type is char. The mark * in data type char means that variable CharValue may contain more than one letter and will keep/print as we input data.

9.2.2. Using Parameter in Function

As VB and Java, function C++ also allows the use of parameter or argument to pass input or collect output from this function. Examine the example below.

Example 9.18. Function by parameter input.

```
#include <iostream>
using namespace std;

// Making function by parameter input
int Quadratic(int X) {
   int result;
   result = X * X;
   return value;
}

int main() {
   int Val, Result;
   cout<<"Input an integer : ";
   cin>>Val;
   RESULT = Quadratic(Val);              //calling quadrant function
   cout<<"Quadratic from number “<<Val<<”is :  "<<RESULT;
   return 0;
}
```

Function in example 9,18 are a simple quadratic. This function needs one input variable input, namely, X. In function main(), variable Val is used to keep value to pass the parameter for calling quadratic function.

Example 9.19. Function with input and output parameter.

```
#include <iostream>
using namespace std;

// Making function with parameter input
int Quadrant(int X, int *result) {
   *result = X * X;
   return *result;
}
int main() {
   int NUMBER, RESULT;
   cout<<"Input an integer : ";
   cin>>Bil;
   // Showing value after process in function
   cout<<"Quadrant from number"<<Number<<" is : "<<Quadrant(Number,
&RESULT);
   return 0;
}
```

Example 9.19 is a development from Example 9.18. In the Quadratic function, we add parameter output, namely, result. Output parameter must be passed base on memory address (namely result), pointer (mark *). In calling the function, input and output parameter must be called. The output parameter to store the calculation results must be given prefix &.

9.3. POINTER AND ARRAY

C++ language allows us to manipulate memory by using pointer. This feature is not provided in other programming language. It is very beneficial if it is correctly used. However, it may crashing the operating system if wrongly used.

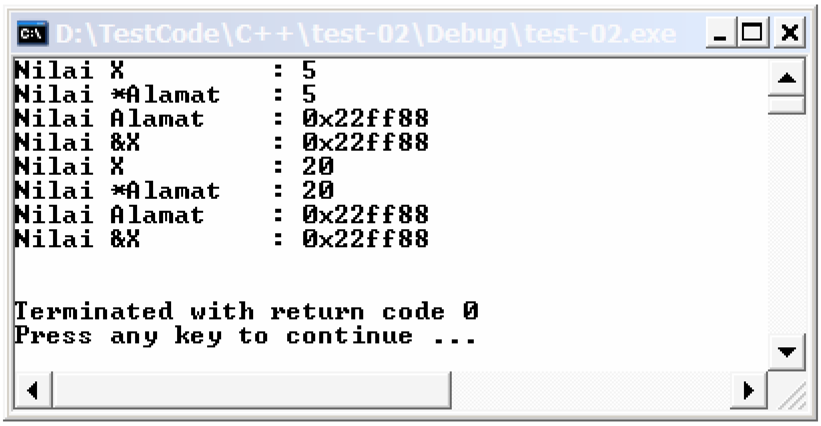
9.3.1. Concept and Understand Pointer

Pointer is variable. However, it is different with normal variable, pointer keeps the address in memory, but not the input value. Examine the following example.

Example 9.20. Pointer declaration.

```
#include <iostream>
using namespace std;

int main() {
  long *Address;
  long X;

  Address = &X;
  X = 5;     // Input value 5 into variabel X

  cout<<"Value  X : "<<X<<endl;
  cout<<"Value  *Address : "<<*Address<<endl;
  cout<<"Value  Address       : "<<Address<<endl;
  cout<<"Value  &X          : "<<&X<<endl;

  *Address = 20;      // Input value 20 into *Address

  cout<<"Value  X : "<<X<<endl;
  cout<<"Value  *Address : "<<*Address<<endl;
  cout<<"Value  Address       : "<<Address<<endl;
  cout<<"Value  &X          : "<<&X<<endl;
  return 0;
}
```

In the above example, we declare variable Address as pointer by adding mark * in front of variable name. If we do not use * , then variable will function as normal variable. Variably X is declared as normal variable with data type long. Please note the statement Address = &X. This statement means that variable Address (not pointer) will fill in with value address from X. Mark & in front of variable name means we want use the value of memory address but not the actual value. The execution result is shown in Figure 9.4.


D:\TestCode\C++\test-02\Debug\test-02.exe

```
Nilai X         : 5
Nilai *Alamat   : 5
Nilai Alamat    : 0x22ff88
Nilai &X        : 0x22ff88
Nilai X         : 20
Nilai *Alamat   : 20
Nilai Alamat    : 0x22ff88
Nilai &X        : 0x22ff88


Terminated with return code 0
Press any key to continue ...
```

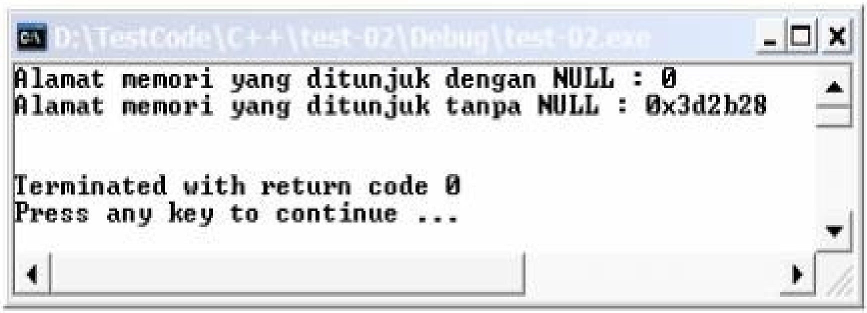
Figure 9.4. Execution result from pointer declaration.

Examine the output values in Figure 9,4. Value 0x22ff88 is a hexadecimal number from variable address. If you observe, when we put the value 5 into variable X then variable *Address will contain the value 5. And if we put the value 20 into variable *Address, value X is changed to 20. This because variable *Address and X occupy same address memory.

Every time we declare a pointer, it will point to random address in memory. To avoid this, we must set the variable pointer with adding value NULL. Please see the execution result in Figure 9.5.

Example 9.21. Pointer declaration with NULL.

```
#include <iostream>
using namespace std;

int main() {
  long *Address;
  long *Address1;
  Address = NULL;
  cout<<"Address memory pointer with NULL : "<<Address<<endl;
  cout<<"Address memory pointer without NULL : "<<Address1<<endl;
  return 0;
}
```



```
Alamat memori yang ditunjuk dengan NULL : 0
Alamat memori yang ditunjuk tanpa NULL : 0x3d2b28


Terminated with return code 0
Press any key to continue ...
```

Figure 9.5. Execution result with NULL pointer.

9.3.2. Using New and Delete in Pointer

Keyword New may be used to allocate memory in empty space. This keyword should be followed by the data type used to let the compiler knows the memory to be allocated. While keyword Delete is the opposite of New. Delete will release memory from our variable. Delete is very important, as undelete memory will consume / waste the memory and make the system uncontrollable. Examine the following example in using New and Delete.

Example 9.22. Using new and delete.

```
#include <iostream>
using namespace std;

int main() {
   int *Address;
   // To allocation memory
   Address = new int;

   // Using memory which allocated
   *Address = 100;

   cout<<"Value *Address : "<<*Address<<endl;

   // free  the memory
   delete Address;
   return 0;
```

}

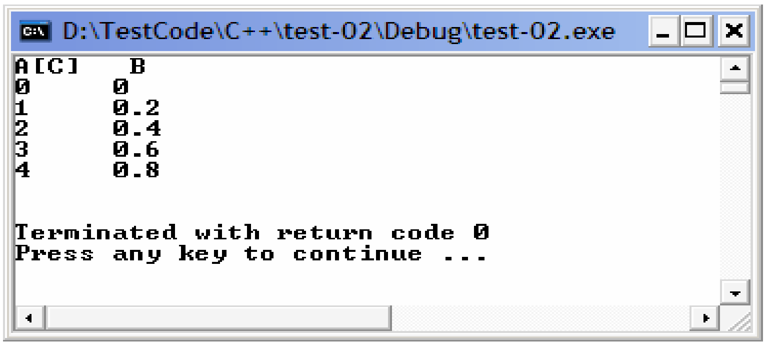
9.3.3. Array

As mention earlier, array is used to keep many data with same type in one variable name. Array definition in C++ almost the same as VB or Java. The only different is the syntax. Examine the example below.

Example 9.23. Simple Array.

```
#include <iostream>
using namespace std;

int main() {
   // declaration array A with 5 type elements int
   int A[5];

   cout<<"A[C]"<<"          "<<"B"<<endl;
   // Input value to array element
   for (int C=0; C<5; C++) {
      double B = 5;
      A[C] = C;
      B = A[C]/B;
     cout<<A[C]<<"            "<<B<<endl;
  }
  return 0;
}
```

In the above example, we declare array named A with data type int. C is the variable counter. To put value in an array, examine the statement A [C] = C. C is the index for variable A. Same as VB or Java, the starting default of the array index is 0. The execution result is shown in Figure 9.6.



```
A[C]   B
0      0
1      0.2
2      0.4
3      0.6
4      0.8


Terminated with return code 0
Press any key to continue ...
```

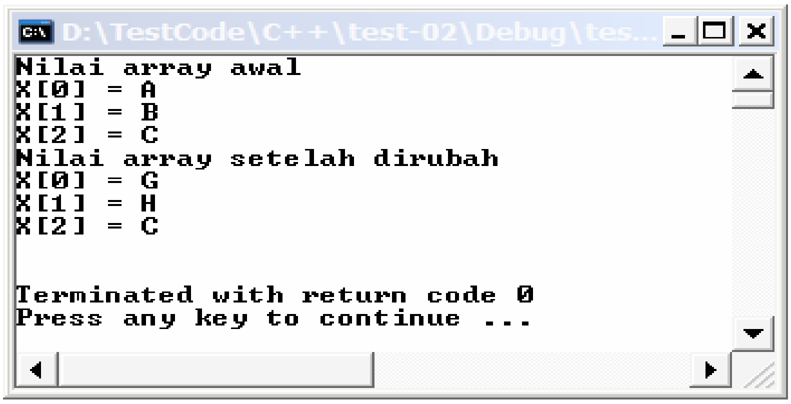
Figure 9.6. Execution result of a simple array program

In the above Example 9.23, we declare array without initialization value. But in fact we can direct give value together with declaration array variable. We may later change the value in other program or when program is in process. Examine the Example 9.24 .

Example 9.24. Array declaration and initialization.

```
#include <iostream>
using namespace std;

int main() {
  // Declaration and  array initiation
  char X[3] = { 'A', 'B', 'C' };
  // Showing early value array element
  cout<<"Value array early"<<endl;
  cout<<"X[0] = "<<X[0]<<endl;
  cout<<"X[1] = "<<X[1]<<endl;
  cout<<"X[2] = "<<X[2]<<endl;
  // Change element to-1 and to-2
  X[0] = 'G';
  X[1] = 'H';
  // Showing change value array element
  cout<<"Array value after change"<<endl;
  cout<<"X[0] = "<<X[0]<<endl;
  cout<<"X[1] = "<<X[1]<<endl;
  cout<<"X[2] = "<<X[2]<<endl;
  return 0;
}
```

The execution result is shown in Figure 9.7.



```
Nilai array awal
X[0] = A
X[1] = B
X[2] = C
Nilai array setelah dirubah
X[0] = G
X[1] = H
X[2] = C


Terminated with return code 0
Press any key to continue ...
```

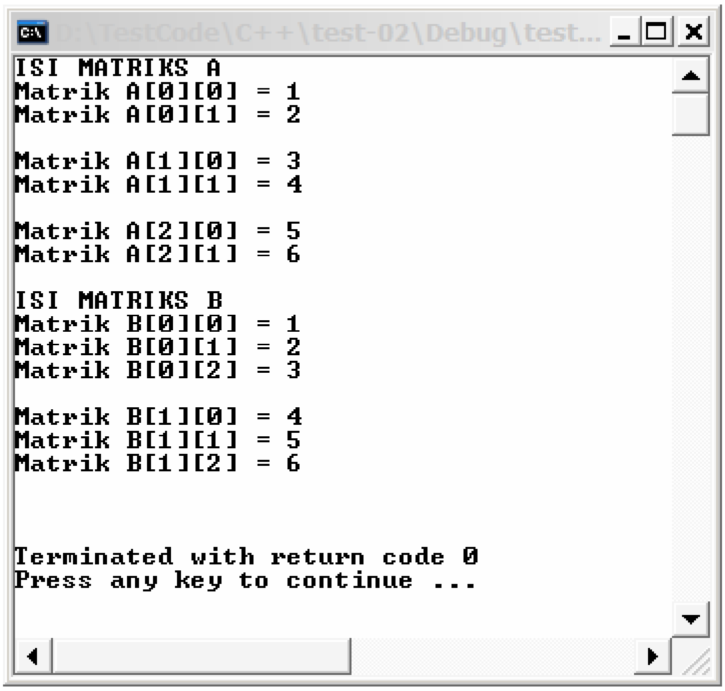
Figure 9.7. Execution result from array declaration and initialization.

C++ also provides feature for multidimensional array. Multidimensional array declaration is shown in Example 9.25.

Example 9.25. Multidimensional array declaration.

```
#include <iostream>
using namespace std;

int main() {
  // Declaration  and initialization array multidimensional
  int Matrix_A[3][2] = { {1,2}, {3,4}, {5,6} };
  int Matrix_B[2][3] = { {1,2,3}, {4,5,6} };

  // Declaration variable for repetition  index
  int j, k;

  // Showing value by keeping array A element
  cout<<"CONTENTS MATRIX A"<<endl;
  for (j=0; j<3; j++) {
    for (k=0; k<2; k++) {
    cout<<"Matrix A["<<j<<"]["<<k<<"] = "<<Matrix_A[j][k]<<endl;
   }
   cout<<endl;
  }

  // Showing value by keeping array B element
  cout<<"CONTENTS MATRIX B"<<endl;
  for (j=0; j<2; j++) {
   for (k=0; k<3; k++) {
     cout<<"Matrix B["<<j<<"]["<<k<<"] = "<<Matrix_B[j][k]<<endl;
   }
   cout<<endl;
  }
  return 0;
}
```

In the above example, there are two (2) multidimensional arrays, namely, Matrix A and Matrix B. Both are two dimensional matrix. Matrix A has 3 rows and 2 columns whereas Matrix B has 2 rows and 3 columns. Value fills in is done during declaration. However, we may fill it using keyboard via cin cmmand. The execution result is shown in Figure 9.8.



```
ISI MATRIKS A
Matrik A[0][0] = 1
Matrik A[0][1] = 2

Matrik A[1][0] = 3
Matrik A[1][1] = 4

Matrik A[2][0] = 5
Matrik A[2][1] = 6

ISI MATRIKS B
Matrik B[0][0] = 1
Matrik B[0][1] = 2
Matrik B[0][2] = 3

Matrik B[1][0] = 4
Matrik B[1][1] = 5
Matrik B[1][2] = 6



Terminated with return code 0
Press any key to continue ...
```

Figure 9.8. Result execution of multidimensional array.

9.4. CLASS

Class concept in C++ is the same as described in Chapter 8. This concept is derived from the object oriented programming paradigm. If you understand well the object oriented programming and Java class programming, then class in C++ concept is not a problem. You much understand, Java adopt most of the C++ programming concepts, including class. The main difference perhaps only in the syntax. In the section, we will not explain the object oriented programming, but the actual implementation in C++.

9.4.1. Class Declaration

Same as Java, C++ class creation uses class as keyword. In a class, there is data and method that will be used by an object (instance). Data and method is usually called class member. Method in C++ has same form as function.

Example 9.26. Declaration and using class

```
#include <iostream>
```

```
using namespace std;

class Rectangle {
    int x, y;
  public:
    void set_value (int,int);
    int wide() {return (x*y);}
};

void Rectangle::set_value (int a, int b) {
  x = a;
  y = b;
}

int main () {
  Rectangle pp1, pp2;
  pp1.set_value(3,4);
  pp2.set_value(7,12);
  cout << “Wide pp1 : “ << pp1.wide()<<endl;
  cout << “Wide pp2 : “ << pp2.wide()<<endl;
  return 0;
}
```

Example above example, the class declared named Rectangle and has two class data member, namely, x and y and two methods, namely, set_value and wide (). The two (2) methods has public access. Same as java, there are three (3) access rights for data or method in class, namely, public, private and protected. Public means class member can access from outside class. Private means class member can only be accessed in class. Protected means class member can be accessed by subclass from class, but not from outside class.

Method set_value has two (2) arguments / parameters with data type int but does not have return value, so we use key word void. Method wide() does not have argument but returns value. To return value on a Method we use data type in front of method name.

In C++ we can implement method in or outside the class , but method declaration must be in side the class. We generally put method implementation outside the class (outside sign {}). Examine the above example. Declaration of method set_value is inside class Rectangle but the implementation is outside the class. Whereas method wide(), both the declaration and the implementation are inside the class. Implementation method outside the class uses sign::, also to define outside class member, (Examine the statement void Rectangle::set value (int a, int b)).

After class is formed, we can use and make object which is an instance of the class.

Examine the beginning part is starts with int main (). In this part, we create two (2) objects, namely, pp1 and pp2 as instance from Rectangle class. Then we use these objects to call method set_value and wide() from Rectangle class. Type above program and run it. How is results?

Similar to Java, a class have generally a constructor that use to initiate variable or allocate memory. Constructor has same name as the class. In a class, it would be better to add method destructor. Destructor is the opposite from constructor. The main goal is to return variable value to the original form and release memory to be free from variable usage. Method destructor has same name as the name class, with prefix ~. Examine how to use constructor and destructor.

Example 9.27. Constructor and destructor.

```
#include <iostream>
using namespace std;
class Rectangle {
    int *length, *wide;
  public:
    Rectangle (int,int);
    ~Rectangle ();
    int wide () {return (*length * *wide);}
};
Rectangle::Rectangle (int a, int b) {
   length = new int;
   wide= new int;
   *length = a;
   *wide = b;
}
Rectangle::~Rectangle () {
   delete length;
   delete wide;
}
int main () {
   Rectangle pp1 (3,4), pp2 (5,6);
   cout << "Wide pp1: " << pp1.wide() << endl;
   cout << "wide pp2: " << pp2.wide() << endl;
   return 0;
}
```

9.4.2. Inheritance

C++ provides facilities inheritance in class. Inheritance process in C++ is more complicated compare with Java. This is because C++ gives possibility inheritance with access right consideration. There are two (2) access rights in inheritance superclass to subclass, namely, public and private.

If the inheritance class is public from superclass, the requirements are as follow:

- Part of public in superclass will stay on as public in subclass.
- Part of protect in superclass will stay on as protect in subclass.
- Part of private in superclass can not access by subclass.

If the inheritance class is private from superclass, the requirements are as follow:

- Part public in superclass will become private in subclass.
- Part protect in superclass will become private in subclass.
- Part private in superclass can not access by subclass. See example:

Example 9.28. Inheritance

```
#include <iostream>
using namespace std;

class CPolygon {
  protected:
    int width, height;
  public:
    void set_values (int a, int b)
      { width=a; height=b;}
  };
class CRectangle: public CPolygon {
  public:
    int area ()
      { return (width * height); }
  };
class CTriangle: public CPolygon {
  public:
    int area ()
      { return (width * height / 2); }
  };
int main () {
  CRectangle rect;
  CTriangle trgl;
  rect.set_values (4,5);
  trgl.set_values (4,5);
  cout << rect.area() << endl;
  cout << trgl.area() << endl;
  return 0;
}
```

In above program, CPolygon is superclass, while CRectangle and CTriangle are subclass. In Cpolygon class, variable width and height are declared as protected, and may only be accessed by the subclass. In addition, this class has method set_values. Both two variables and method will be inheritance to the subclass such as CRectangle and CTriangle. Please note how CRectangle and CTriangle is declared as the descendant class from CPolygon by using keyword public. Please change public in the declaration of Ctriangle class to Ctriangle class: private CPolygon. You will see the following error in compiling,

```
Compiling source file(s)...
oo-test.cpp
oo-test.cpp: In function `int main()':
oo-test.cpp:9: error: `void CPolygon::set_values(int, int)'
is inaccessible
oo-test.cpp:28: error: within this context
oo-test.cpp:28: error: `CPolygon' is not an accessible base of `CTriangle'
```

Why there is an error in compilation? This is because the above rule. Method set_values in class CPolygon declare as public, but inheritance to class CTriangle as private. It will change method from public to private in class CTriangle. As you may understand, if class member has private access right then it can not be accessed from outside.

9.4.3. Polymorphism

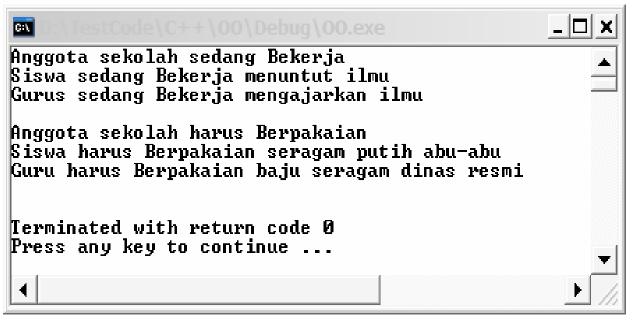
In C++, to apply polymorphism, we must use special function named virtual. Put the virtual function in superclass, so we can re-define it in subclass. Examine the following example:

Example 9.29. Using virtual function.

```
#include <iostream>
using namespace std;

class MemberSchool {
  char* name;
  char* address;
public:
  void SetName(char* N) {
     name = N;
  }
```

```
   void SetAddress(char* A) {
      address = A;
   }
   char* GetName() {
      return name;
   }
   char* GetAddress() {
      return address;
   }
   // Create virtual function
   virtual void Working() {
      cout<<"Working"<<endl;
   }
   virtual void Dressing() {
      cout<<"Dressing"<<endl;
   }
};

class Student: public MemberSchool {
   char* Faculty;
   char* Program;
  int semester;
public:
  void SetJurusan(char* J) {
     Jurusan = J;
  }
  void SetProgram(char* P) {
    Program = P;
  }
  void SetSemester(int smt) {
    semester = smt;
  }
  char* GetFaculty() {
     return Faculty;
  }
  char* GetProgram() {
     return Program;
  }
  int GetSemester() {
     return semester;
  }
  // override on function Working
  void Working() {
     cout<<"Work demanding knowledge"<<endl;
  }
  // override on function Dressing
  void Dressing() {
     cout<<"Dressing uniform white gray"<<endl;
  }
};

class Teacher: public Member School {
  char*  position;
  char* expertise;
public:
  void SetPosition(char* jbt) {
     position = jbt;
  }
  void SetExpertise(char* khl) {
     expertise = khl;
  }
  char* GetPosition() {
     return position;
  }
  char* GetExpertise() {
     return expertise;
  }
  // override on function Working
  void Working() {
     cout<<"Work teaching knowledge"<<endl;
  }
  // override on function Dressing
   void Dressing() {
       cout<<"Dressing formal uniform"<<endl;
   }
};

// Main function
int main() {
   // instantion on class MemberSchool, Student and Teacher
   MemberSchool As;
   Student Sw;
   Teacher Gr;

   // Calling function Working from each class
   cout<<"Member school in ";
   As.Working();
   cout<<"Student in ";
   Sw.Working();
   cout<<"Teacher in ";
   Gr.Working();
   cout<<'\n';
   // Calling function Dressing from each class
   cout<<"Member much be ";
   As.Dressing();
   cout<<"Student must be ";
   Sw.Dressing();
   cout<<"Teacher much be ";
   Gr.Dressing();
   return 0;
}
```

In the above source code, there are two (2) virtual methods, namely, Working and Dressing. These methods will be used in subclass in different applications. Please examine the content of each method in each subclass. This is normally called as overriding. The review overiding concept in Chapter 8. Compare overriding concept between Java and C++. As we execute the above program, the result is shown in Figure 9.9. Overloading may also be done using virtual function. Please review the difference between overriding and overloading as described iin Chapter 8.



```
Anggota sekolah sedang Bekerja
Siswa sedang Bekerja menuntut ilmu
Gurus sedang Bekerja mengajarkan ilmu

Anggota sekolah harus Berpakaian
Siswa harus Berpakaian seragam putih abu-abu
Guru harus Berpakaian baju seragam dinas resmi


Terminated with return code 0
Press any key to continue ...
```

Figure 9.9. Execution results of virtual function and overriding.

In the above Example 9.29, virtual function is completed with the content of the function. C++ also provides pure virtual function that only declared as a function with empty content. This concept is similar to interface in Java. The pure virtual function will be inheritance into classes. The advantage in using pure virtual function is in the freedom in defining functions in the descendant class. The pure virtual function is usually used in abstract class. An abstract class has at least one pure virtual function. We should not create an object directly from an abstract class.

Polymorphism concept in C++ is based on the virtual function, pure virtual function, overriding, overloading, and abstract class. Examine the following polymorphism example.

Example 9.30. Applying polymorphism.

```
#include <iostream>
using namespace std;

class CPolygon {
   protected:
     int width, height;
   public:
     void set_values (int a, int b)
        { width=a; height=b; }
     virtual int area (void) =0; // pure virtual function
     void printarea (void)
        { cout << this->area() << endl; }
   };
class CRectangle: public CPolygon {
   public:
     // overriding function area
     int area (void)
        { return (width * height); }
    };
class CTriangle: public CPolygon {
    public:
      // overriding function area
      int area (void)
         { return (width * height / 2); }
    };
int main () {
    CRectangle rect;
    CTriangle trgl;
    CPolygon *ppoly1 = &rect;                //  defining object pointer
    CPolygon *ppoly2 = &trgl;                //   defining object pointer
    ppoly1->set_values (4,5);
    ppoly2->set_values (4,5);
    ppoly1->printarea();
    ppoly2->printarea();
    return 0;
}
```

In the above example, CPolygon class is an abstract class and has pure virtual function, namely, area. Examine how to declare pure virtual function with a beginning statement virtual. This function does not have any contents but only mark = 0. We can not make an object directly from Cpolygon class. But we can make object pointer to allocate memory base on CPolygon class. In Cpolygon class a keyword this is used. The this keyword is used as a pointer to the class. Statement this->area() is the same as CPolygon->area (). Thus, same as calling the virtual function are in the class.

In the above code, we create two (2) pointer variables *ppoly1 and *ppoly2 that has the same value as the address of variable rect and trgl. Note the mark * and & to point to memory address. Execute the program and examine the result.

9.5. OBJECT ORIENTED APPLICATION DESIGN

Basic concept of object oriented programming has been studied in Chapter 8. Its application in Java is also described in Chapter 8. While its implementation in C++ is in this Chapter. In this section, we will discuss the application design in object oriented.

In this example, we will solve one of the problem in book store, namely, supply / stock, If we closely examine, a book store will sell many things, including, novel, books, magazines, office stationery, etc.. Each has its own brand, price, and other characteristics. Moreover each materials also has relations with pricing, discount, number of purchases and storage. Availability of supplies in book store determine by incoming materials from ordering and outgoing as being sold. Each out going materials should automatically decrease the supply.

The beginning of the step in object oriented application is to create the abstraction of the problem. Examine closely the problem, we can create the main class of all available things. This is due to the special characteristics of the materials, as well as the general characteristic for all materials. For example, price is general characteristic for all materials. However, certain brand is specific to a certain materials, especially stationery. Moreover, title is a characteristic belongs to books. Confined by this condition we can make problem abstraction as follow.

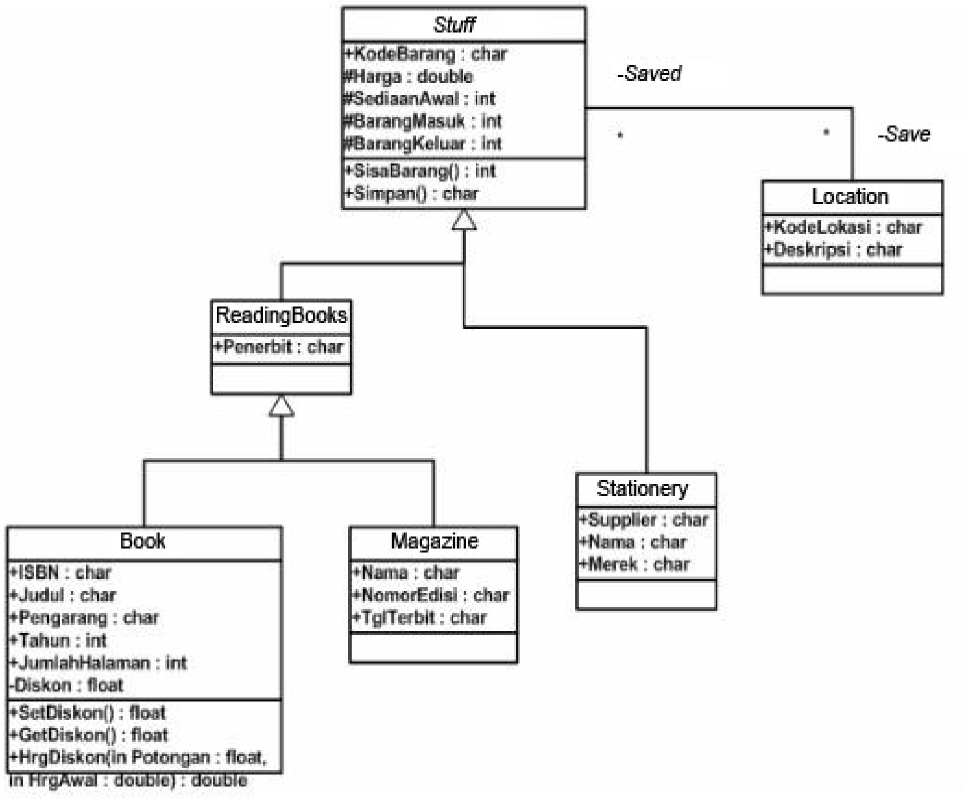


Figure 9.10. Case abstraction in book store supply materials.

In Figure 9.10, we create six (6) classes to facilitate the above case. There are two (2) relations described in the Figure, namely, relations inheritance with arrow mark and association (relations between two or more class) with line without arrow. Material is superclass and described in abstract class. Bahan Bacaan and Alat Tulis is a subclass of Barang, whereas Buku and Majalah are subclass from Bahan Bacaan. Location is a class to refer the storage.
In the above case, we may apply the inheritance and polymorphism concept as well as use of pure virtual function in class Barang. These functions may be inheritance to be implemented in the subclasses. The code design is shown in Table 9.3.

Examining the diagram and table 9.3, we may write the source code. There are six (6) classes in the code. Each class may be written in a separate file or collected in a singe file. Then the implementation from these class may be written in a single file. To link these files and call class from different file, we may use preprocessor #include follows by the file name.

Table 9.3. Class, Function and Parameter on bookstore stock application.

Class Name	Variable	Function	Parameter	Comments
Barang	KodeBrg Harga SediaanAwal BrgKeluar BrgMasuk	-SisaBarang -Simpan	pAwal, bKeluar, bMasuk kodeLok	Kelas Abstrak
BahanBacaan	Penerbit			Subclass dan mewarisi anggota kelas Barang
Buku	ISBN Judul Pengarang Tahun JumlahHalaman Diskon	-setDiskon -getDiskon -HrgDiskon	Hrg, Potongan	Subclass dari BahanBacaan dan mewarisi anggota kelas tersebut
Majalah	Nama NoEdisi TglTerbit			Subclass dari BahanBacaan dan mewarisi anggoota kelas tersebut
AlatTulis	Supplier			
	Nama Merek			
Lokasi	KodeLokasi Descripsi			

9.6. SUMMARY

- The general program structure in C++ includes file registration, function definition, main() section, and code block.
- Data type primitive in C++ consists of int, long, float, double, char, boolean, and short. The available composite data type is struct, enum and array.

- The operator type are assignment operator, unary operator, binary operator, relational operator, bitwise operator and ternary operator.
- Selection Control structure may be done with if (without then) and switch... case. While in looping, C++ provides several commands, namely, for, while, and do-while.
- The input and output statements is done with commands cout and cin as stream that includes in class iostream.
- Function may or may not returns value. Parameter in function may be input parameter, output parameter or both of them.
- Pointer is a variable. However, different from normal variable, Pointer keep the memory address but not the actual value.
- C++ supports object oriented programing with class, inheritance, virtual function, overriding, overloading and polymorphism.

9.7. EXERCISE

1. Create a C++ program to calculate the average value of 34, 56, 91, 11, and 22.
2. Create a C++ program to determine the rice after discount for the followings.

```
Input Price / unit                   : Rp. 20.000
Number Of things buy                 : 5
------------------------------------- -------------------------------------
Total cost before discount           :Rp.100000
discounts (10%)                      :Rp.  10000
Net price                            :Rp.  90000
```

3. Someone has a saving account in bank and starting balance at Rp. 10,000.-. He allows to store or withdraw money from his saving account. Create a C++ program for Banking transaction. The main menu should be as follows:

```
-------------------------------------
PT. BANK ABC
-------------------------------------
Balance:....
Transaction Menu
1. Deposit savings
2. Take savings
3. Exit
Menus (1/2/3)? ...
```

Requirement:
Bank made policy that minimum balance in account is Rp 10,000.-

If the customer chooses option 1 then the customer will be asked to enter the amount of Rupiah to be stored. If customer chooses option 2 then customer will be asked to enter the amount of Rupiah to be withdraw. However, if the balance is less than Rp. 10000, the program will refuse. (Note: please use cin command to get input from keyboard)

4. Create a program use function to calculate the final mark in a class. There are two (2) arguments in the function, namely, mid test mark and final exam mark. The program output should gives A if the average mark is the same or larger than 80, and B if it less then 80 and same or larger than 70, and C if it is less then 70.

5. Examine problem no 5 in Chapter 8. Create abstraction diagram as well as the program in C++.