CEN-201
Object Oriented Programming (OOP)

- Textbook
- Teach Yourself C++
- Helbert Schildt

CEN-201
Object Oriented Programming (OOP)

- Machine Language (The first programming Language)

- Assemly (Allows long program to be written)

- High level languages (Fortran, Cobol, Algol, Basic)

- Structured Programming Languages (Algol, Pascal, C) :Relies on well defined control structure, code blocks, the absence of the GOTO, and standalone subroutines. The essence of structured programming is the reduction of a program into its constituent elements.

Object Oriented Programming (C++, JAVA, PASCAL, …)

- To allow more complex programs to be written, a new approach to the job of programming was needed.
- Object-Oriented Programming encourages you to decompose a problem into its constituent parts.
- Each component becomes a self-contained object that contains its own instructions and data that relate to that object.

Three Key Features of OOP

- *Encapsulation* is the mechanism that binds together code and the data it manipulates. When data and code are linked together, an object is created.

- *Polimorfizm* is the quality that allows one name to be used for two or more related but technically different purposes. For example in C, the absolute value action requires three distinc function names: **abs(), labs**, and **fabs().** In C++ each function can be called by the same name, such as **abs(). In C++,** it is possible to use one possible name for many differnt purposes. This called *function overloading*. The advantage of polymorfism is that it helps to reduce complexity by allowing one interface to specify a general class action.

Polymorfism can be applied to operators, too. This type polymorfism called operator overloading. (operators: +, -, *,++,-- etc.)

Inheritance is the process by which one object can aquire the properties of another. It allows an object to support the concept of *hierarchical classification*.

For example, think about the description of a house. A house is part of the general class called building. In turn, building is part of the more general class structure, which is part of even more general class of objects that we call man-made. In each case, the child class inherits all those qualities associated with the parent and adds to them its own defining characteristics.

Example: Person(general class), students, employees, staff , acedemic members (derived classes).

Two Versions of C++

```
/*A traditional style C++ program. The header is a file name*/
#include <iostream.h>
int main()
{/* program code */
return 0;
}

/*A modern style C++ program that uses the new style
headers and a namespace. Headers are abstractions that
might be mapped to files.*/
#include <iostream>
using namespace std;
int main()
{
/* program code*/
return 0;
}
```

New style C++ headers
<iostream>
<fstream>
<vector>
<string>

C headers that are used

in C	in C++
<math.h>	<cmath>
<string.h>	<cstring>

C++ Console I/O

C++ is a superset of C, all elements of C language are also contained in C++. All C programs are also C++ programs. “printf()” and “scanf()” are I/O functions in C. In C++ I/O is performed using I/O operators. The output operator is “<<“ and the input operator is “>>”

cout<< : to output to the console,
cin>> : to input a value from keyboard.

Outputing in C

```
printf(“%d\n”,255);                 prints an integer.
printf(“%f\n”,2.55);                prints a float.
printf(“%s\n”,”text yazı yazar”);   prints a text.
```

It is possible to output more than one value in a single I/O expression.

Example:

```
#include <iostream>
using namespace std;
int main () {int i,j; double x;float y;
cout<<”This string is output to the screen. \n”;
//or
cout<<” This string is output to the screen. .”<<endl;
i=5;j=3;x=1,2;y=4.5;
cout<<3.7<<“ “<<i<<“ “<<j<<” “<<x<<“ “<<y<<endl;
cout<<endl;} /*Next line*/
```

cout << → ekrana yazdırm

cin >> → kullanıcıdan veri alma

endl → satır atlama

Example: This program promts the user for an integer value:

```
#include <iostream>
using namespace std;
int main()
{
int i; float j
cout<<”Enter an integer value: ”;
cin>>i;
cout<<”Here is your number  : ”<<i<<endl;

return 0;
}
```

Classes: A First Look

The class is the mechanism that is used to create objects. The class is declared using the class keyword. The syntax of a class declaration is similar to that of structure. Its general form is shown below:

```
class class_name
{
private:
// private functions and variables
public:
// public functions and variables
} object_list;
```

The class_name becomes a new type that is used to declare objects of classes. By default, all functions and variables declared inside a class are privite to that class.

The private members are accessible only by other members of that class.
To declare public class members, the **public** keyword is used, followed by a colon. All functions and variables declared after the public specifier are accessible both by other members of the class and by any other part of the program.

Example:

```
#include <iostream>
using namespace std;
class myclass {
private:
int a;
public:
void set_a(int num);
int get_a();
};
```

```
void myclass::set_a(int num)
{
a=num;
}
int myclass::get_a()
{
return a;
}

int main()
{
myclass object_1,object_2;
object_1.set_a(15);object_2.set_a(22);
cout<<”The value of a in object_1 : ”<<object_1.get_a()<<endl;
cout<<” The value of a in object_2 : ”<<object_2.get_a()<<endl;
return 0;
}
```

In the previos example, “a” is defined as a private member. Therfore “a” can not be accessible outside of the object. Examples that are given below are wrong.

```
object_1.a=10; //wrong
object_2.a=15; //wrong
```

If “a”’ is defined as a public member, “a”‘ can be accessible outside of the object.

Example:

```
#include <iostream>
using namespace std;
class myclass {
public:
int a;
};
int main()
{
myclass object_1,object_2;
object_1.a=15;object_2.a=22;
cout<<” The value of a in object_1 : ”<<object_1.a<<endl;
cout<<” The value of a in object_1 : ”<<object_2.a<<endl;
return 0;
}
```

Some Differences Between C and C++

```
C                    C++
----------------------------------
char f1(void);     char f1();
```

In C++, if the function doesn’t take any parameter, there is no need to use “void”.

In C++, all functions must have a protype.

In C++, if a function is declared as returning a value, it must return a value.

C++ defines **bool** data type, which is used to store Boolen (true/false) values. **True** is any nonzero value and **false** is zero.

Example: (bool)

```
#include <iostream>
using namespace std;
int main()
{
bool outcome;
if(outcome) cout<<”true”;
else cout<<”false”;
return 0;
}
```

Introducing Function Overloading

Example: In C, the standard library contains the functions **abs(), labs()** and **fabs()** which return the absolute value of an integer, a long integer, and a floating point value, respectively. In C++, one function name can be used for these three functions.

```
#include <iostream>
using namespace std;
int abs1(int n);
long abs1(long n);
double abs1(double n);
int main()
{int x=5; long y=10; double z=20.12;
cout <<”integer number:”<<abs1(x)<<endl;
cout<<”long integer number:“<<abs1(y)endl;
cout<<”double (reel) number :”<<abs1(z)<<endl;
return 0;
}
```

```
//for integer abs ()
int abs1(int n)
{
return n<0 ? -n:n;
}
//for long integer abs ()
long int abs1(long int n)
{
return n<0 ? -n:n;
}
//for double abs ()
double abs1(double n)
{
return n<0 ? -n:n;
}
```

Example: (function overloading)

```
#include <iostream>
using namespace std;
void date(char *date);//date as a string
void date(int month, int day, int year);//date as numbers

int main()
{
date(“8/23/2010”);
date(8,23,2010);
return 0;
}
// date as a string
void date(char *date)
{
cout<<”date :”<<date<<endl;
}
```

```
// date as numbers
void date(int month, int day, int year)
{
cout<<”Date :”<<month<<”/”<<day<<”/”<<year<<endl;
}
```

Example:

```
#include <iostream>
using namespace std;
f1(int a);//takes one argument
f1(int a, int b);//takes two arguments
int main()
{
f1(10);
f1(3,7);
return 0;
}
//bir parametre
void f1(int a)
{
cout<<”one argument :”<<a<<endl;
}
void f1(int a,int b)
{
cout<<”The first parameter :”<<a<<”The second parameter :”<<b;
cout<<endl
```

Chapter 2
Introducing Classes

Constructer: Creates objects

Destructor: Destroys objects

Example:

```
#include <iostream>
using namespace std;
class myclass {
private:
int a;
public:
myclass();//constructor
~myclass();//destructor
void set_a(int num);
int get_a();
};
```

```
void myclass::set_a(int num)
{
a=num;
}
int myclass::get_a()
{
return a;
}
myclass::myclass()
{cout<<”In Constructor”<<endl;}
myclass::~myclass()
{cout<<”In Destructor”<<endl;}
int main()
{
myclass object_1,object_2;
object_1.set_a(15);object_2.set_a(22);
return 0;
}
```

Example:

```
#include <iostream>
#include <cstring>
#include <cstdlib>
using namespace std;
#define SİZE 255;
class strtype {
private:
char *p;
int len;
public:
strtype();//constructor
~strtype();//destructor
void set(char *ptr);
void show();
};
```

```
//İnitialize a string object (constructor)
strtype::strtype()
{
p=(char *) malloc(SIZE);
if (!p)
{cout<<”Allocation error ”<<endl;  exit(1);}
*p='\0';
len=0;
}
//Free memory when destroying string object
strtype::~strtype()
{cout<<”Freenig p”<<endl; free(p);}
void strtype::set(char *ptr)
{if (strlen(ptr) >=SIZE)
{cout<<”String too big”;  exit(1);}
strcpy(p,ptr);
len=strlen(p);
}
```

```
void strtype::show()
{
cout<<p<<"  Length of the string is :"<<len<<endl;
}
int main() {
strtype s1,s2;
s1.set("This is a test");
s2.set("I like C++");
s1.show();
s2.show();
return 0;
}
```

Constructors That Take Parameters

Example:

```
#include <iostream>
using namespace std;
class myclass {
private:
int a;
public:
myclass(int x); //Constructor
void show();
};
myclass::myclass(int x)
{cout<<”constructor”<<endl; a=x;
void myclass::show(); {cout<<a<<endl}
int main()
{myclass object_1(5); object_1.show;return 0;}
```

Introducing Inheritance

Example:

```
#include <iostream>
using namespace std;
class B {   //Define base class
private:
int i;
public:
void set_i(int n);
int get_i();
};
class D :public B {  //Define derived class
{
private:
int j;
public:
void set_j(int n);
int mul(); };
```

```
void B::set_i(int n)
{i=n;}
int B::get_i()
{return i;}
void D::set_j(int n)
{j=n;}
int D::mul()
{return j*get_i();}
İnt main ()
{ B ob1; D ob2;
ob1.set_i(10);
ob2.set_i(20); ob2.set_j(30);
cout<<ob1.get_i<<endl;
cout<<ob2.get_i<<endl;
cout<<ob2.mul<<endl;
return 0;
}
```

Object Pointers

So far, you have been accessing members of an object by using the dot operator.

When a pointer is used, the arrow operator (→) rather than the dot operator is employed. To obtain the address of an object, precede the object with the & operator, just as you do when taking the address of any other type of variable.

Example

```
#include <iostream>
using namespace std;
class myclass  {
private: int a;
public:
myclass (int x);  //constructor
int get();
};
```

```
myclass::myclass(int x)
{a=x; }
int myclass::get()
{
return a;
}
int main()
{
myclass ob_1(125); //create object
myclass *p; //create pointer to object
p=&ob_1;
cout<<”Value using object”<<ob_1.get()<<endl;
cout<<”Value using pointer”<<p->get()<<endl;
return 0;
}
```

“myclass *p;” creates a pointer to an object of **myclass**. The address of **ob** is put into **p** by using this statement “p=&ob_1;”

Classes, Structures, and Union are Related

As you have seen, the class is syntactically similar to the structure. In fact, the only difference between a structure and a class is that, by default, the members of a class are private but the members of a structure are public. The expanded syntax of a structure is shown here:.

```
struct type-name
{
//public function and data members
private :
//private function and data members
}object-list;
```

Example: A program that uses struct to create a class:

```
#include <iostream>
#include <cstring>
using namespace std;
struct st_type {  st_type (double b, char *n);//constructor
void show();
private: double balance; char name[40]; };
st_type::st_type(double b,char* n)
{balance=b;strcpy(name,n);  }
void st_type::show()
{cout<< “Name : “<<name<<endl;
cout<< “Balance : “<<balance<<endl;}
int main()
{st_type acount_1(125,”Ahmet”);
st_type acount_2(250,”Ayşe”);
acount_1.show()
acount_2.show();
return 0;}
```

Union

•In a union all data members share the same memory location. In C++, a union defines a class type that can contain both functions and data as members.
•A union is like a structure in that, by default all members are public.
•Unions can contains constructor and destroctor functions.
•Unions can not inherit any other class an they can not be used as base class for any other type.
•Unions must not have any static members. They also must not contain any object that has a constructor or destructor.
•Unions can not have virtual member functions.

There is a special type of union in C++ called anonymous union. An anonymous union does not have a type name, and no variables can be declared for this sort of union. An anonymous union tells the compiler that its members will share the same memory location.

Integer number

The value of w of an N bit integer $a_{N-1}a_{N-2}\cdots a_0$ is given by

$$w = -a_{N-1}2^{N-1} + \sum_{i=0}^{N-2} a_i 2^i$$

Single Precision binary Floating-point format (32 bit):

The value of w of an 32 bit float $a_{31}a_{30}\cdots a_0$ is given by

$$w = (-1)^{a_{31}}\left(1 + \sum_{i=1}^{23} a_{23-i} 2^{-i}\right) * 2^{E-127}$$

$$E = \sum_{i=23}^{30} a_i 2^{i-23}$$ $E = 00_H$ and $E = FF_H$ are interpreted specially.

Double Precision binary Floating-point format (64 bit):

The value of w of an 64 bit double $a_{63}a_{62}\cdots a_0$ is given by

$$w = (-1)^{a_{63}}\left(1+\sum_{i=1}^{52} a_{52-i}2^{-i}\right)*2^{E-1023}$$

$E = \sum_{i=52}^{62} a_i 2^{i-52}$ $E = 000_H$ and $E = 7FF_H$ are interpreted specially.

Example

```
#include <iostream>
using namespace std;
union union_x {
int x;
double y;
void show_data();
};
void union_x::show_data()
{cout<<”The value of x : “<<x<<endl;
cout<<”The value of y : “<<y<<endl;}
int main()
{union_x union_1;
 union_1.x=15;
union_1.show_data();
 union_1.y=20;
union_1.show_data();
return 0; }
```

Example: (anonymous union)

```
union
{
int i;
char ch[4];
}
int main()
{i=10;
cout<<“The value of i :”<<i<<“The value of ch :”<<ch<<endl;
ch[0]='x';
cout<<“The value of i :”<<i<<“The value of ch : ”<<ch<<endl;
retun 0;
}
```

In-Line Functions

•In C++, it is possible to define functions that are not actually called but, rather, in line, at the point of each call (similar to macro in C).

•The advavtage of in-line functions is that they have no overhead associated with the function call and return mechanism.

•The disadvantage of in-line functions is that if they are too large and called too often, your program grows larger.

Declaration : Add the ***inline*** specifier infront of the function.

inline specifier is a request, not a command, to the complier.

Example:

```
#include <iostream>
using namespace std;
inline int even (int x)
{if (!(x%2))
cout <<i<<” is an even number”<<endl;
else <<i<<” is an odd number”<<endl;
}
int main()
{ int i;
i=5;
even(i);
i=6;
even(i);
}
```

Automatic In-Lining

If a member function’s definition is short enough, the definition can be included inside the class declaration. Doing so causes the function to automaticially become an in-line function, if possible.

Example:

```
#include <iostream>
using namespace std;
class samp {
private: int i,j;
public :
samp(int a, int b);
void divisible () {if  (!(i%j)) cout<<i“ divisible by “<<j<<endl;
else cout<<i“ is not divisible by “<<j<<endl;}}
samp::samp(int a, int b){i=a;j=b}
int main()
{samp ob_1(10,2), ob2_2(10,3);
ob_1.divisible();
ob_2.divisible();}
```

Chapter 3 A Closer Look at Classes

Assigning Objects: One object can be assigned to another provided that both objects are of the same type. By default, when one object is assigned to another, a bitwise copy of all the data members is made.

Example:

```
#include <iostream>
using namespace std;
class myclass  {private: int a,b;
public:
void set(int i,int j) {a=i;b=j}
void show() {cout<<a”    “<<b<<endl;}
};
int main()
{
myclass ob_1,ob_2;
ob_1.set(10,4);
ob_2=ob_1;//assign ob_1 to ob_2
ob_1.show();
ob_2.show();
return 0;
}
```

Example (this program contains an error)

```
#include <iostream>
#include <cstring>
#include <cstdlib>
using namespace std;
class strtype {
private:
char *p;
int len;
public: double y; void set(char *ptr,double x);
strtype(char *ptr);//constructor
~strtype();//destructor
void show();};
strtype::strtype(char *ptr) { len=strlen(ptr);
p=(char *) malloc(len+1);
if (!p) {cout<<”Allocation error”<<endl;  exit(1)}
strcpy(p,ptr);}
//destructor
strtype::~strtype()
{cout<<"freeing p.   The value of y is :"<<y<<endl;free(p);}
void strtype::show() cout<<p<<"  The length of the string is : "<<len<<"
The value of y is : "<<y<<endl;
```

```
void strttype::set(char *ptr,double x){ y=x;
if (strlen(ptr)>=len)
{len= strlen(ptr)+1;     free(p);p=(char *) malloc(len);
if (!p)  {cout<<"Allocation error "<<endl;  exit(1);}}
len= strlen(ptr)+1;
strcpy(p,ptr);}
int main() {strtype s1,s2;
s1.set(“This is a test”);
s2.set(“I like C++”); s1.show();
s2.show();
s2=s1; //This generates an error
s1.show();
s2.show(); return 0;}
```

After assigning s1 to s2, both s1’s p and s2’s p will point to the same memory. Thus, when these objects are destroyed, the memory pointed to by s1’s p is freed twice and the memory originally pointed to by s2’ p is not freed at all.

Passing Objects to Functions

In C++, objects can be passed to functions as arguments in just the same way that other types of data are passed. As with other types of data, by default all objects are passed by value to a function.

Example:

```
#include <iostream>
using namespace std;
class samp {
private:
int i;
public :
samp(int n) {i=n}
void set_i(int n) {i=n;}
int get_i(){return i;}
};
```

```
void sqrt_it(samp o)
{
o.set_i(o.get_i()*o.get_i());
cout<<”The value of a (in function sqrt_it() )
:”<<o.get_i()<<endl;
}
int main() {
samp a(10);
sqr_it(a);
cout<<”The value of a : ”<<a.get_i()<<endl;
return 0;
}
```

Output;

```
The value of a (in function sqrt_it()) :100
The value of a : 10
```

Example:

```
#include <iostream>
using namespace std;
class samp {
private:
int i;
public :
samp(int n) {i=n}
void set_i(int n) {i=n;}
int get_i(){return i;}};
void sqrt_it(samp *o){o->set_i(o->get_i()*o->get_i());
cout<<”i of the object that passed to the function by a pointer : ”<<o-
>get_i()<<endl;}
int main() {samp a(10);
sqr_it(&a);
cout<<”i of the object a : ”<<a.get_i()<<endl;
return 0;}
```

Output;

```
i of the object that passed to the function by reference : 100
i of the object a : 100
```

When a copy of an object is made when being passed to a function, it means that a new object comes into existence. When a copy of an object is made to be used in a function call the constructor function is not called. However, when the function terminates and the copy is destroyed, the destroctor function is called.

Example:

```
#include <iostream>
using namespace std;
class samp {
private:
int i;
public :
samp(int n) {i=n; cout<<”Constructing object”<<endl;}
~samp() { cout<<”Destructing object”<<endl; }
int get_i(){return i;}
};
```

```
int sqrt_it(samp o)
{
return (o.get_i()*o.get_i());
}
int main()
{
samp a(10);
cout<<sqr_it(a)<<endl;
return 0;
}
```

Output;

```
Constructing object
Destructing object
100
Destructing object
```

Returning Objects from Functions

•Functions can return objects. When an object is returned by a function, a temporary object is automatically created which holds the return value. After the value returned, this object is destroyed.

•The destruction of the temporary object might cause unexpected side effect in some situations.

•**Example:**

```
#include <iostream>
using namespace std;
class samp {
private:
char s[80];
public :
void show() {cout<<s<<endl;}
void set(char *str){strcpy(s,str)i;}
};
```

```
samp input() //return an object of the type samp.
{ char s[80];
samp str;
cout<<”;Enter a string :”;
cin>>s;
str.set(s);
return str;}
int main ()
samp ob_1;
ob_1=input();
ob_1.show();
return 0;
}
```

•You must be carefull about returning objects from functions if those functions contain destructor functions because the returned object goes out of scope as soon as the value returned to the calling routine.
•If the object returned by the function has a destructor that frees dynamically allocated memory, that memory will be freed even though the object that is assigned the return value is still using it.

Example:

```
#include <iostream>
using namespace std;
class samp {
private:
char *s;
public :
samp() {s='\0';}
~samp() {if (s) free(s); cout<<”Freeing s”<<endl;}
void show() {cout<<s<<endl;}
void set(char *str);};
```

```
samp::set(char *str)
{ s=(char *) malloc(strlen(str)+1);
If (!s) {cout<<”Allocation error”<<endl; exit(1);}
strcpy(s,str);
}
samp input() //return an object of the type samp.
{ char s[80]; samp str;
cout<<”Enter a string:”;
cin>>s;
str.set(s);
return str;}
int main ()
samp ob_1;
ob_1=input(); //This causes an error.
ob_1.show();
return 0;
}
```

The output of this program:

```
Enter a string : Hello
Freeing s
Freeing s
Hello
Freeing s
Null pointer assignment
```

An Introduction to Friend Functions

A friend is not a member of a class but still has access to its privite elements.
Two reasons:
•Operator overloading
•You may want one function to have access to the privite members of two or more differnt classes.
Example:

```
#include <iostream>
using namespace std;
class myclass
{private:
Int n,d;
public :
myclass(int i,int j) {n=i;d=j;}
friend isfactor(myclass ob1);};
```

```
int isfactor(myclass ob1)
{
if (ob1.n%ob1.d) return 1;
else return 0;}
int main()
{
myclass ob_1(10,2), ob_2(10,3);
if isfactor(ob_1) cout<<”2 is a factor of 10”<<endl;
else cout<<” 2 is not a factor of 10 ”<<endl;
if isfactor(ob_2) cout<<” 3 is a factor of 10 “<<endl;
else cout<<” 3 is not a factor of 10”<<endl;
}
```

- A friend function is not inherited
- A friend function can be friends with more than one class
- A function of a class can be friend of another class.