Operator Overloading

One of the most powerful feature of C++ is Operator Overloading.

Operator functions defines the operation that the fn. will perform relative to the class upon which it will work.

Creating a Member Operation Function

Syntax of member operation function

Return-type class_name::operator#(arg-list){

//operations

}

# is **place holder**, we have to place a operator in place of # that we want to overload. For e.g.:- operator+()

See program 15.1

In case of unary operator arg-list is empty and in case of binary operator arg-list has right operand as argument.

Left operator will call the operator function and right operand will be passed as argument.

For e.g. :-

a+b;   //a is left operand and b is right operand.

Note:- . :: .* ?   These operator cannot be overloaded.

All the operator functions are inherited by the derived class except the operator=().

The above is true only for operator fn. That are member of the class.

Operator Overloading using friend function

Friend function are not member of the class, They can be used to create operator function, hence we have to pass the reference of the object.

In case of friend operator function, we will have to pass reference of object in case of unary operator and in case of binary operator we will have to pass reference of both the objects.

See program 15.2

Note:- There is no advantage of the friend operator function over member operator function except one i.e.:-

Ob+100; //This is correct in case of member operator function.

100+Ob; //But this is not correct in case of member operator function because 100 cannot call the member operator function as it is not an object.

But this is not a problem in case of friend operator function, as we pass both the object in the function and we can use function overloading to make it possible.

Operator Overloading

In case friend operator function of unary operator, reference of the object must be given to ensure changes take place.

Overloading new and delete operators

void pointer in C / C++

A void pointer is a pointer that has no associated data type with it. A void pointer can hold address of any type and can be typcasted to any type.

```
int a = 10;
char b = 'x';

void *p = &a;  // void pointer holds address of int 'a'
p = &b; // void pointer holds address of char 'b'
```

Advantages of void pointers:
1) malloc() and calloc() return void * type and this allows these functions to be used to allocate memory of any data type (just because of void *)

```
int main(void)
{
    // Note that malloc() returns void * which can be
    // typecasted to any type like int *, char *, ..
    int *x = malloc(sizeof(int) * n);
}
```

Note that the above program compiles in C, but doesn't compile in C++. In C++, we must explicitly typecast return value of malloc to (int *).

Some Interesting Facts:
1) void pointers cannot be dereferenced. For example the following program doesn't compile.

```
#include<stdio.h>

int main()

{

        int a = 10;

        void *ptr = &a;

        printf("%d", *ptr);

        return 0;

}
```

Output:

```
Compiler Error: 'void*' is not a pointer-to-object type
```

Operator Overloading

The following program compiles and runs fine.

```
#include<stdio.h>
int main()
{
    int a = 10;
    void *ptr = &a;
    printf("%d", *(int *)ptr);
    return 0;
}
```

Output:

```
10
```

2) The C standard doesn't allow pointer arithmetic with void pointers. However, in GNU C it is allowed by considering the size of void is 1. For example the following program compiles and runs fine in gcc.

```
#include<stdio.h>

int main()

{

        int a[2] = {1, 2};

        void *ptr = &a;

        ptr = ptr + sizeof(int);

        printf("%d", *(int *)ptr);

        return 0;

}
```

Output:

```
2
```

Malloc() vs new

NEW	MALLOC
calls constructor	Does not calls constructors
It is an operator	It is a function
Returns exact data type	Returns void *

on failure, Throws	On failure, returns NULL
Memory allocated from free store	Memory allocated from heap
can be overridden	cannot be overridden
size is calculated by compiler	size is calculated manually

Note:- size_t is an alias for unsigned int, it is used to allocate the maximum memory allocated by system, used with new operator. New operator calls constructor automatically as delete calls destructor automatically.

```
//Allocate memory to an object, constructor called automatically
void *operator new(size_t size){
return pointer_to_memory
}
//Delete an object, destructor called automatically
Void operator delete(void *p){
}
```

See program 15.3

Special Operators:- "[]", ",", "()", "->"

There is only one restriction to these operators and i.e. They must be nonstatic member function. They cannot be **friend.**

Overloading [] :-

[] is considered as Binary Operator in C++. Given an object O[5] is translated into this call to the function O.operator[](3) .

See program 15.4.1

Overloading () :-

() is used for function calls. We can easily overload this function.

See program 15.4.2

Overloading -> :-

-> is used with pointer to point to a data member. It is considered as unary operator in C++

Operator Overloading

See Program 15.4.3

Overloading , :-

, is used to separate objects/variables in C++. It is considered as binary operator in C++.

See program 15.4.4