Lab Sheet 2
Additional Features in C++

Manipulators

Manipulators are operators that are used to format the data display. The most commonly used manipulators are endl and setw. The endl manipulator, when used in an output statement, causes a linefeed to be inserted. It has same effect as using the newline character "\n" in C. For example

```
...

cout<< "LODGING" <<lexp<<endl;

cout<< "CLOTHING" <<cexp<<endl;

cout<< "TRAVELING" <<texp<<endl;

.....
```

If we assume that values of variable lexp, cexp and texp are 2000, 800 and 2500 respectively, then output appears is

```
LODGING 2000

CLOTHING 800

TRAVELING 2500
```

It is not the ideal output. setw(n) manipulator eliminate this problem by specifying filed width. The value is right justified within field. For example

```
...

cout<<setw(11)<< "LODGING" <<setw(8)<<lexp<<endl;

cout<<setw(11)<< "CLOTHING" <<setw(8)<<cexp<<endl;

cout<<setw(11)<< "TRAVELING" <<setw(8)<<texp<<endl;
```

.....

output of this section is

LODGING 2000

CLOTHING 800

TRAVELING2500

Namespace

The namespace feature in C++ allow us to specify a scope with a name, that is, namespace is a named scope. The namespace feature help us to reduce the problem of polluting the namespace. Namespace is defined as follows.

```
namespace nsn
{
   int item;
   void showitem(int it)
   {
      cout<<it;
   }
}
```

The elements from the namespace are used as

```
cout<<nsn::item;
```

or it can be accessed by including the particular element as

```
using nsn::item;
```

after inclusion the statements

```
cout<<item;  //correct
showitem(item);  //not correct
```

We can also include every thing from the namespace as

```
using namespace nsn;
```

After this inclusion every element from the namespace can be used directly as

```
cout<<item;
```

```
showitem(item);
```

Pass by reference

When an address of the variable/object is passed, the function works directly on the actual variable/object used in the call. This means that any changes made to the variable/object inside the function will reflect in actual variable/object. For example

```
 return_type function_name(datatype &, datatype &)
```

Return by reference

The primary reason of returning the value by reference is to use a function call on the leftside of the equal sign. For example
max(a,b)=-1

Here the function should return reference to variables, but not the values, then the function call will yield a reference to either a or b depending on their values.

Structure with function

The value of structure variable can be passed as a

parameter to a function. For example

```
struct stu
{
    ...
};


void show(stu);
```

Overloaded function

Overloaded function appears to perform different operation depending on the kind of data sent to it. It performs one operation on one kind of data but another operation on a different kind. The reason behind using the overloaded function is because of its convenience to use the same function name for different operation.

```
void convert();      //takes no argument
```

```
voi                    //tak on
d   convert(int n);    es    e  argument of type int
voi convert(float,i    //tak tw argument of type
d   nt);               es    o  float and int
```

It uses the number of arguments, and their data types, to distinguish one function from another.

Inline function

We know that function save memory space but take some extra time. If the functions are short, we may put in the function directly in the line with the code in the calling program. But the trouble with repeatedly inserting the same code is that you lose the benefits of the program organization and clarity that come with using function. If the function is very short, the instructions necessary to call it may take up as much space as the instructions within the function body, so that there is not only a time penalty but a space penalty as well. The solution to this is the inline function. This kind of function is written like a normal function in the source file but complies into inline code instead of into a function. Beside of this the source file remains well organized and easy to read, since the functions are shown as a separate entity. Functions that are very short, say one or two statements are candidates to be inlined.

inline(keyword) float(retutn type) convert(int n)

i.e

inline float convert(int n)

Here all we need is the keyword *inline* in the

function definition.

Default Arguments

The function can be called without specifying all its arguments. This won't work on just any function that is the function declaration must provide default values for those arguments that can be missed.

Let us take an example

```
void dearg(char
='?',int=25);                    //declaration with default
int main()                       arguments
{
   dearg();
   dearg('<');
   dearg('>',30);
   return 0;
}
void dearg(char ch,int n)
{
     ...
```

```
}
```

Here the function dearg( ) takes two arguments. It is called three times from main( ) program. The first time it is called with no arguments, second time with one argument and third time with two. The first and second provides default arguments, which will be used if the calling function doesn't supply them. The default argument follows an equal sign, which is placed directly after name. It is also possible to use variable name. If one argument is missing it assumed to the last argument as we show in second.

Exercises

Use manipulators where seems to be useful.

1. Write a program to set a structure to hold a date (mm,dd and yy), assign values to the members of the structure and print out the values in the format 11/28/2004 by function. Pass the structure to the function

```
#include<iostream>
#include <iomanip>
using namespace std;
int main()
{
    int d,m,y;
    cout<<"Enter day.\t";
    cin>>d;
    cout<<"Enter month.\t";
    cin>>m;
    cout<<"Enter year.\t";
    cin>>y;
    cout<<setw(2)<<setfill('0')<<d<<"/";
    cout<<setw(2)<<setfill('0')<<m<<"/";
    cout<<setw(4)<<setfill('0')<<y;
    return 0;
}
```

2. Write a program using the function overloading that converts feet to inches. Use function with no argument, one argument and two arguments. Decide yourself the types of arguments. Use pass by reference in any one of the function above.

```
#include<iostream>
using namespace std;
float conversion();
float conversion(float);
void conversion (float,float &);
int main()
{
    float ft;
    cout<<"Enter number in feet:\t"<<endl;
    cin>>ft;
    float ans;
    float &a=ans;
           cout<<"answer from function with no
parameter:\t"<<conversion ()<<endl;
         cout<<"answer from function with single
parameter:\t"<<conversion (ft)<<endl;
    conversion(ft,a);
          cout<<"answer from function with two
parameter:\t"<<ans<<endl;
}
void conversion (float h,float &b )
{
    b=h*12;
}
float conversion(float h)
{
    return h*12;
}
float conversion()
{
    float temp;
     cout<<"Enter value of feet for function with no
argument:\t"<<endl;
    cin>>temp;
    return temp*12;
}
```

3. Define two namespaces: Square and Cube. In both

the namespaces, define an integer variable named

```
    "num" and a function named "fun". The "fun"
    function in "Square" namespace, should return the
    square of an integer passed as an argument while
    the "fun" function in "Cube" namespace, should
    return the cube of an integer passed as an
    argument. In the main function, set the integer
    variables "num" of both the namespaces with
    different values. Then, compute and print the
    cube of the integer variable "num" of the
    "Square" namespace using the "fun" function of
    the "Cube" namespace and the square of the
    integer variable "num" of the "Cube" namespace
    using the "fun" function of the "Square"
    namespace.
#include<iostream>
#include<math.h>
using namespace std;
namespace cube
{
    int num;
    int fun(int a)
    {
        return a*a*a;
    }
}
namespace square
{
    int num;
    int fun(int b)
    {
        return b*b;
    }
}
int main()
{
    square::num=5;
    cube::num=10;
    cout<<"square of "<<cube::num<<" is
"<<square::fun(cube::num)<<endl;
    cout<<"cube of "<<square::num<<" is
"<<cube::fun(square::num)<<endl;
}
```

4. Write a function that passes two temperatures by reference and sets the larger of the two numbers to 100 by using return by reference.

```
#include<iostream>
using namespace std;
int &ma(int &,int &);
int main()
{
    int t1,t2;
    t1=95;
    t2=94;
    int &a=t1;
    int &b=t2;
    cout<<"Previous value of t1 and t2
are:\t"<<t1<<'\t'<<t2<<endl;
    ma(a,b)=100;
    cout<<"after value of t1 and t2
are:\t"<<t1<<'\t'<<t2<<endl;
}
int &ma(int &x,int &y)
{
    if(x>y)
    {
        return x;
    }
    else
    {
        return y;
    }
}
```

5. Assume that employee will have to pay 10 percent income tax to the government. Ask user to enter the employee salary. Use inline function to display the net payment to the employee by the company.

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

```
    int s;
    cout<<"Enter your salary:\t"<<endl;
    cin>>s;
    cout<<"Net payment to the employee = "<<net(s);
}
inline float net(int a)
{
    return(a-(a*0.1));
}
```

6. Write a program that displays the current monthly salary of chief executive officer, information officer, and system analyst, programmer that has been increased by 9, 10, 12, and 12 percentages respectively in year 2010. Let us assume that the salaries in year 2009 are
   Chief executive officer Rs. 35000/m
   Information officer Rs. 25000/m
   System analyst Rs. 24000/m
   Programmer Rs. 18000/m
   Make function that takes two arguments; one salary and other increment. Use proper default argument.

```
#include<iostream>
using namespace std;
float increment(float a,float b=12);
int main()
{
    float salary,increase;
    cout<<"New salary of chief executive officer is
"<<increment(35000,9)<<endl;
    cout<<"New salary of information officer is
"<<increment(25000,10)<<endl;
    cout<<"New salary of chief system analyst is
"<<increment(24000)<<endl;
    cout<<"New salary of chief Programmer is
"<<increment(18000)<<endl;
}
float increment(float a, float b)
{
    return(a+a*(b/100));
}
```