More about functions

Lecture - 5

Scope

- Unary scope resolution operator
- Namespaces
- Inline functions
- References and reference parameters

Unary scope resolution operator

- It is possible to declare local and global variables of the same name

- Unary scope resolution operator(::) to access a global variable when the local variable of the same name is in scope

Example

```
#include <iostream>
int number=7;
int main()
{
double number=10.5;
std::cout<<“local “<<number;
std::cout<<“global “<<::number;
}
```

Namespaces

- A program includes many identifiers defined in different scopes
- A variable of one scope can overlap with the variable of the same name in a different scope, creating a naming conflict
- C++ attempts to solve this problem with namespaces

Namespaces (contd..)

- Each namespace defines a scope in which identifiers and variables are placed
- Defining namespaces
  - Use the keyword namespace
- Accessing namespace members with quantified names
  - Scope resolution ::

Example

```
#include <iostream>
int integer1=98; // global variable
namespace example
{ double pi=3.141;
int integer1=8;
void printvalues();
} // namespace ends here
void main()
{ std::cout<<“global variable”<<integer1;
std::cout<<“pi “<<example::pi;
example::printvalues();
}// main closing

void example::printvalues()
{  std::cout<<integer1;
std::cout<<“ global integer “<<::integer1;
}
```

Inline functions

- Function calls involve execution-time overhead
- qualifier inline advises compiler to generate a copy of the function's code in place to avoid a function call
- Compiler can ignore the inline qualifier

Example

- #include<iostream>

```
inline double cube(double side)
{ return side*side*side; }
void main()
{ double sidevalue ;
 cin>>sidevalue; cout<<“cube”<<cube(sidevalue); }
```

References and reference parameters

- Two ways to pass arguments to functions
  - ***Pass by value –*** a copy of argument's value is made and passed to the called function. Changes to copy do not affect original value
  - ***Pass by reference –*** called function has the ability to access the caller's data directly.

Example

```
#include <iostream>
Int squareByValue(int);
void squareByReference(int &);
Int main()
{
int x=2, y=4;
cout<<" square of x
   "<<squareByValue(x);
cout<<" x after call "<<x;
squareByReference(y);
cout<<" square of y "<<y;


return 0;
} // main closing
```

```
int squareByValue(int
    number)
{
 return number*number;
}



void squareByReference(int
    &numberRef)
{
 numberRef =
    numberRef*numberRef ;
}
```

References as aliases

- References can also be used as aliases for other variables within a function

```
int main()
{  int count = 1 ;
       int &cref=count ;
       cref++;
       std::cout<<count ;
}
```

• reference variables must be initialized in declarations and cannot be reassigned

Returning a reference from a function

- Problem – the function variables do not have scope outside the function, and memory is de-allocated (***dangling references***)

- The variable should be declared as ***static***

Assignment

- Difference between Inline and Macros