## Scope of Variables

- Global variables

  - Declared outside of any function and are known (*visible*) to any function in the program

- Local variables

  - Declared inside a function and are known (*visible*) to this function only

  - The formal parameters of a function are local variables for the function

  - Variables declared inside a block are also local (they are legal inside the block only !!)

- *Scope* of a variable

  - The section of the program where it is legal to use the variable

(valid everywhere)
**global variable**

```
#include <iostream.h>

float  avg;

void compute(int, int, int&);

void main()
{
  int x, y, sum;

  cout << "Enter two values ";
  cin >> x >> y;
  compute(x,y,sum);
  cout << "The sum is " << sum << endl;
  cout << "The average is " << avg << endl;

  if( avg > 10.5)
  {
    float diff;

    diff = x - y;
    cout << "The difference is " << diff;
  }

}

void compute(int num1, int num2, int& res)
{
  int x;

  x=10;
  res = num1 + num2 + x;
  avg = res/3.0;
}
```

**local variables**
(valid only inside the main )

**local variable**
(valid only inside the if statement )

**local variables**
(valid only inside the compute_sum )

- What if a variable is declared twice, once as global and once as local

  - The local declaration takes precedence over the global declaration (i.e., the local variable *hides* the global variable)

```
#include <iostream.h>
void SomeFunc (float);
const int a=17;
int b, c;

void main()
{
 b=4;
 c=6;
 SomeFunc(42.8);
}

void SomeFunc(float c)
{
 float b;

 b=2.3;
 cout << << a << b << c;
}
```

- Misuse of global variables

  - Rather than passing variables to a function, it is possible to make all variables global ones !!! **Do not do this** !!!

**(x,y,sum are global)**

```
#include <iostream.h>

int x, y, sum;
void compute_sum();

void main()
{
 cout << "Enter two values ";
 cin >> x >> y;
 compute_sum();
 cout << "The sum is" << sum;
}

void compute_sum()
{
 sum = x + y;
}
```

**(only sum is global)**

```
#include <iostream.h>

iint sum;
void compute_sum(int,int);

void main()
{
 int x,y;
 cout << "Enter two values ";
 cin >> x >> y;
 compute_sum(x,y);
 cout << "The sum is" << sum;
}

void compute_sum(int num1, int num2)
{
 sum = num1 + num2;
}
```

- Disadvantages of using global variables in a function

  - Functions which use global values are not *self-contained* !!

  - A function is *self-contained* if it does not rely on anything that is declared outside of it

    *Self contained functions can easily be integrated into different programs

    *Incorporating a function which uses global variables into a program requires us to incorporate the declarations of the global variables into the program as well

  - More difficult to *track down errors*

  - *Naming conflict*: some local variables might have the same name with some global variables !!

**these declarations must also be incorporated into the program**

**function to compute the area of a triangle**

```
void tri_area()
{
  area = 0.5 * base * height;
  cout << area;
}
```

```
#include <iostream.h>

float  base, height, area;
void tri_area();

void main()
{

 cout << "Enter the base and height ";
 cin >> base >> height;
 tri_area();
}

void tri_area()
{
  area = 0.5 * base * height;
  cout << area;
}
```

- Advantages of using global variables in a function

  - Useful when many functions need to share the same data

  - Less memory space (no memory is allocated for formal parameters)

  - Faster (no actual parameters are copied to the formal parameters)

- Design guidelines

  - Avoid using global variables for the functions that are intended to be reused in other programs later

  - If you use global variables, avoid using common names for them, to reduce the chance of naming conflicts

## Lifetime of Variables

- What is the lifetime of a variable ?

  - The period of time (during program execution) when a variable has memory allocated to it

- What is the lifetime of a global variable ?

  - It is the lifetime of the entire program

- What is the lifetime of a local variable ?

  - It is the lifetime of the function or the block within which the local variable has been declared

- Storage categories

  - Automatic variables
  - Static variables

- Initialization in declarations

  - Combine declaration and initialization in one statement

```
#include <iostream.h>

void main()
{
 int x=5;
 int y, product;

 cout << "Enter a number";
 cin >> y;
 product = x * y;
 cout << "The result is" << product << endl;
}
```

## Value returning functions (revisited)

- When is it appropriate to use them ?

  - When there is only one result to be returned by the function

  - When the result is going to be used within an expression

- Examples

  - Write a function to compute the factorial of a number (*n!=1 x 2 x 3 ... x n*)

  - Write a function named *distance* that accepts the rectangular coordinates of two points (x1,y1) and (x2,y2) and calculates and returns the distance between the two points

  - Write a function to compute the roots of the quadratic equation

**Recommended exercises**

- *Quick Check* excersices: 1-6

- *Exam preparation* excersices: 1, 3, 4, 8-10