Scope

- The **scope of a declaration** is the block of codes where the identifier introduced by that declaration refers to the object being declared.

- A declaration associates an identifier with an object, but the same identifier could be used in different declarations.

- Identifiers may not be recognized throughout the entire program

- A **global declaration** is a declaration that is made outside any block.

- Objects that are declared inside the boby of a function are called **local** to the function.

- The scope of a formal argument name includes the body of its function.

Scope

Example:

```
int  y = 38
                                        scope of y
void  f (int, int);
                                        scope of f
void  main ( ) {

      int  z = 47;
                                        scope of z
      {

            const  int  a = 90;
                                        scope of a
            z  +=  a;
      }
      z  =  2 * y;
      f (1, 2);
}

void  f (int  s,  int  t)  {
                                        scope of s & t
      int  r = 12;
                                        scope of r
      s  = r + t;

      int  i = 27;
                                        scope of i
      s  += i;
}
```

Scope

- Example

```
#include <iostream.h>

int Number; //global variable

void Increment(int Number) {
    Number = Number + 1;
    cout << Number << endl;
}

void main() {
  Number = 1;
  Increment(Number);
  cout << Number << endl;
  return 0;
}
```

- o *scope* of <u>***Number***</u> does not include function *Increment*
- o *scope* of ***Number*** includes only the function *Increment*

Referring to Global Variables inside function

- Example

```
#include <iostream.h>

int Number; //global variable

void Increment(int Num) {
    Num = Num + 1;
    cout << Num << endl;
    Number = Number + 1;
}

void main() {
  Number = 1;
  Increment(Number);
  cout << Number << endl;
  return 0;
}
```

- o ***Number*** inside function *Increment* is referring to global variable
- o Lead to some of the most difficult to find bugs in programming

Global V ariable & Pass-by-Reference

- Example

```
#include <iostream.h>

int Number; //global variable

void Increment(int& Num) {
    Num = Num + 1;
    cout << Num << endl;
    Number = Number + 1;
}

void main() {
  Number = 1;
  Increment(Number);
  cout << Number << endl;
  return 0;
}
```

- o When *Increment* is called, *Num* refers to global variable *Number*
- o *"Number = Number + 1"* also refers to global variable *Number*

Scope & Argument Passing

- Example

```
#include <iostream.h>
int A,B,C,D;

void Two(int A, int B, int& D) {
     B = 21; D = 23;
     cout << A << " " << B << " "
          << C << " " << D << endl;
}

void One(int A, int B, int& C) {
     int D;    // Local variable
     A = 10; B = 11; C = 12; D = 13;
     cout << A << " " << B << " "
          << C << " " << D << endl;
     Two(A,B,C);
}

void main() {
     A = 1; B = 2; C = 3; D = 4;
     One(A,B,C);
     cout << A << " " << B << " "
          << C << " " << D << endl;
     Two(A,B,C);
     cout << A << " " << B << " "
          << C << " " << D << endl;
}
```

```
10 11 12 13
10 21 23 23
1 2 23 4
1 21 23 23
1 2 23 4
```

Static objects

- A local object is created when its definition is executed. When the function containing its definition finishes execution, the **lifetime** of the object is over.

```
int f (int x) {
    int y = 3;
    y = y + x;
    return y;
}

cout << f(1) << endl;          // output is 4

cout << f(1) << endl;          // output is 4
```

- The lifetime of a **static object** begins with the execution of the first call to the function in which it is defined. A static object exists and maintains its value from one execution of the function to the next.

```
int f (int x) {
    static int y = 3;
    y = y + x;
    return y;
}

cout << f(1) << endl;          // output is 4

cout << f(1) << endl;          // output is 5

cout << f(2) << endl;          // output is 7
```

Basic Types

- ***char*** - a character in the ASCII character set
- ***int*** - whole number (-32768 to 32767)
- ***float*** - real number

  — char , int - ***ordinal*** types

Why Types?

- values are stored internally in a computer as binary numbers
- computers are comfortable to add a *float* to a *char*
- protects the programmer from making mistakes
  — a type defines the ***range of values*** that an object can take on, and the ***operations*** that apply to it

Data T ype : Char

- Constants

```
const char star = '*';
```

- Variable Declaration

```
char letter;
```

- Variable Assignment

```
letter := 'A';
```

- Character I/O

```
cin << A << B << C << D;
```

INPUT: 1X3Y

```
A = '1', B = 'X', C = '3', D = 'Y'
```

```
cout << A << B << C << D << endl;
```

OUTPUT:
1X3Y

Character Set

- 7-bit ASCII character set: 128 characters
- begins with 32 non-printing control characters
  - Of interests:
    - 0 (\0) — NUL character, commonly used as end of string character
    - 7 (\7) — BEL character, for programs with bells and whistles
    - 8 (\10) — BS character (\b), backspace character
    - 10 (\12) — LF character (\n), line feed character
    - 13 (\15) — CR character (\r), carriage return character
- space (32) is the first printable ASCII character
- NUL is the 0th ASCII character
- 'A' through 'Z' have code values 65 through 90
- 'a' through 'z' have code values 97 through 122
- '0' through '9' have code values 48 through 57

Some Character Functions from ctype.h

- isupper(c) — returns nonzero if c is an uppercase letter
- islower(c) — returns nonzero if c is a lowercase letter
- isdigit(c) — returns nonzero if c is digit character
- isalpha(c) — returns nonzero if either islower(c) or isupper(c) returns nonzero
- isalnum(c) — returns nonzero if either isalpha(c) or isdigit(c) returns nonzero
- isspace(c) — returns nonzero if c is a space, newline, formfeed, carriage return, tab or vertical tab
- tolower(c) — if c is uppercase, returns lowercase; otherwise, returns c
- toupper(c) — if c is lowercase, returns uppercase; otherwise, returns c

Relational Operators with Ordinal Type

- 'A' through 'Z' have code values 65 through 90

  ```
  'A' < 'B' < ••• < 'Z'
  ```

- 'a' through 'z' have code values 97 through 122

  ```
  'a' < 'b' < ••• < 'z'
  ```

- '0' through '9' have code values 48 through 57

  ```
  '0' < '1' < ••• < '9'
  ```

Using Ordinal Type in a For Loop

- <counter variable> can have any ordinal (counting) type, including ***int*** and ***char*** . (~~*float*~~ is forbidden.)

  o Example

  ```
  for (Char CharCounter = 'a';
            CharCounter <= 'e';
            CharCounter++)
    cout << CharCounter;

  for (Char CharCounter = 'E';
            CharCounter >= 'A';
            CharCounter--)
    cout << CharCounter;
  ```

Example

```
//File islower.cpp
#include <iostream.h>

int islower(char c) {
     if (c>='a' && c<='z')
          return 1;
     else
          return 0;
}

void main() {
     char c;
     cin >> c;
     if (islower(c))
          cout << "\'" << c << "\'" << " "
               << "is a lower case letter"
               << endl;
     else
          cout << "\'" << c << "\'" << " "
               << "is a not a lower case letter"
               << endl;
     return 0;
}
```

Standard I/O Streams

- ***stream*** - a sequence of characters
    - o ***cout*** - output stream
    - o ***cin*** - input stream

- ***input/output operator***
    - ***o*** ***output operator (<<)***
        - convert typed data to a stream of characters and insert them into an output stream
    - ***o*** ***input operator (>>)***
        - extract information from an input stream of characters and convert them into typed data

Output conversion (from internal representation to characters)

```
cout << "Enter " << setw(3) << n
     << " float values separated by a blank.";
```

E	n	t	e	r		2	0	3		f	l	o	a	t	•••

Input conversion

```
cin >> month >> day >> year;
```

1	2			2	4		9	6						

Streams as a Sequence of Lines

- Internal representation of a character stream

<65><65><32><66><66>**<u><13><10></u>**<48><49><32><50><51>**<u><13><10></u>**<52>

end of line ***end of line***

- Display on TERMINAL

```
AA BB
01 23
4
```