# Basic C / C++

FRA 142 Computer Programming for Robotics and Automation Engineering II

**Suriya Natsupakpong, PhD**

Institute of Field Robotics (FIBO)

King Mongkut's University of Technology Thonburi (KMUTT)

## Difference Between Python and C++

- Python uses Garbage Collection whereas C++ does not.
- C++ is a statically typed language, while Python is a dynamically typed language.
- Python is easier to use than C++.
- Python is run through an interpreter, whilst C++ is pre-compiled. Hence, C++ is faster than Python.
- C++ supports pointers and incredible memory management.
- Python supports very fast development and rapid, continuous language development.
- Python has less backwards compatibility.
- Majority of all applications are built from C++.
- Majority of all 3D applications offer Python access to their API's.
- Python code tends to be 5 to 10 times shorter than that written in C++.

http://www.differencebetween.info/difference-between-cplusplus-and-python

## Difference Between Python and C++

- In Python, there is no need to declare types explicitly.
- Smaller code size in Python leads to "rapid prototyping", which offers speed of development.
- Python requires an engine to run.
- Python is interpreted each time it runs.
- Python is hard to install on a Windows box and thus makes distribution of the program problematic.
- C++ is a pure binary that links to existing libraries to assist the coding.
- In Python, variables are in scope even outside the loops in which they are first instantiated.
- In Python, a function may accept an argument of any type, and return a value of any type, without any kind of declaration beforehand.
- Python provides flexibility in calling functions and returning values.
- Python looks cleaner, is object oriented, and still maintains a little strictness about types.

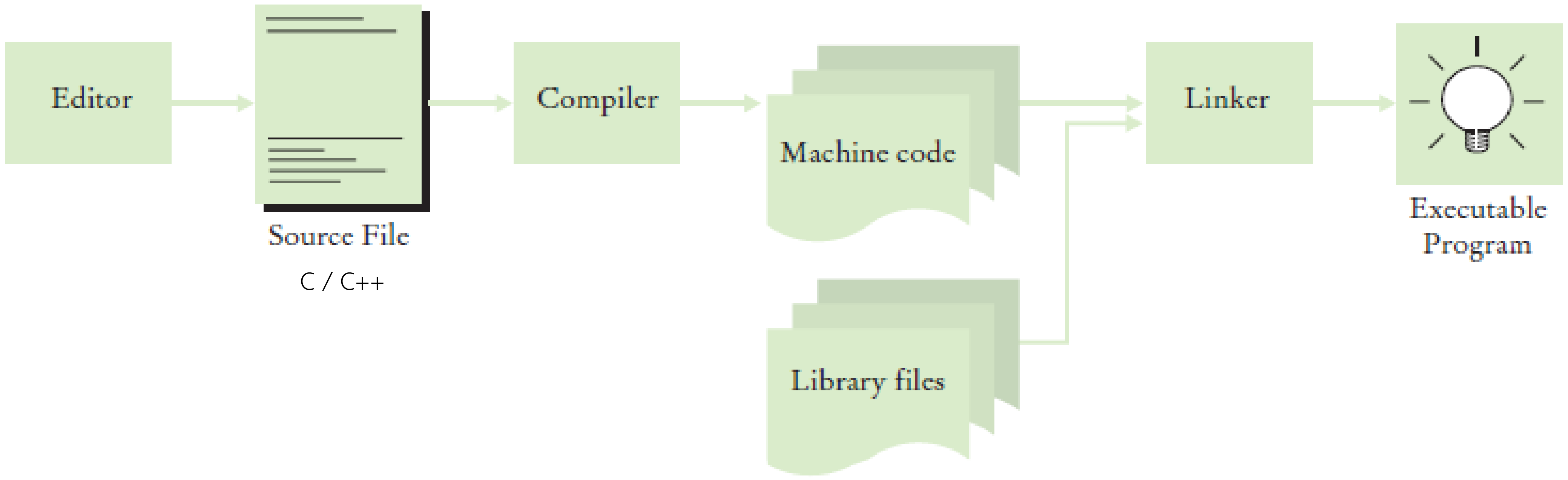
http://www.differencebetween.info/difference-between-cplusplus-and-python

## Interpreter VS Compiler

| Interpreter | Compiler |
|---|---|
| Translates program one statement at a time. | Scans the entire program and translates it as a whole into machine code. |
| It takes less amount of time to analyze the source code but the overall execution time is slower. | It takes large amount of time to analyze the source code but the overall execution time is comparatively faster. |
| No intermediate object code is generated, hence are memory efficient. | Generates intermediate object code which further requires linking, hence requires more memory. |
| Continues translating the program until the first error is met, in which case it stops. Hence debugging is easy. | It generates the error message only after scanning the whole program. Hence debugging is comparatively hard. |
| Programming language like Python, Ruby use interpreters. | Programming language like C, C++ use compilers. |

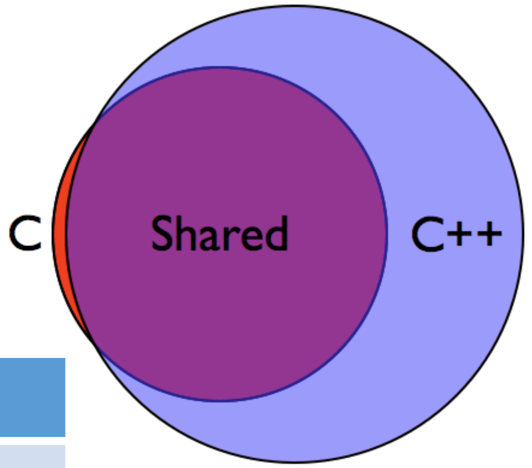
https://www.programiz.com/article/difference-compiler-interpreter

# From C/C++ Source Code to Executable Program



Cay S. Horstmann. 2010. *C++ for Everyone* (2nd ed.). Wiley Publishing.

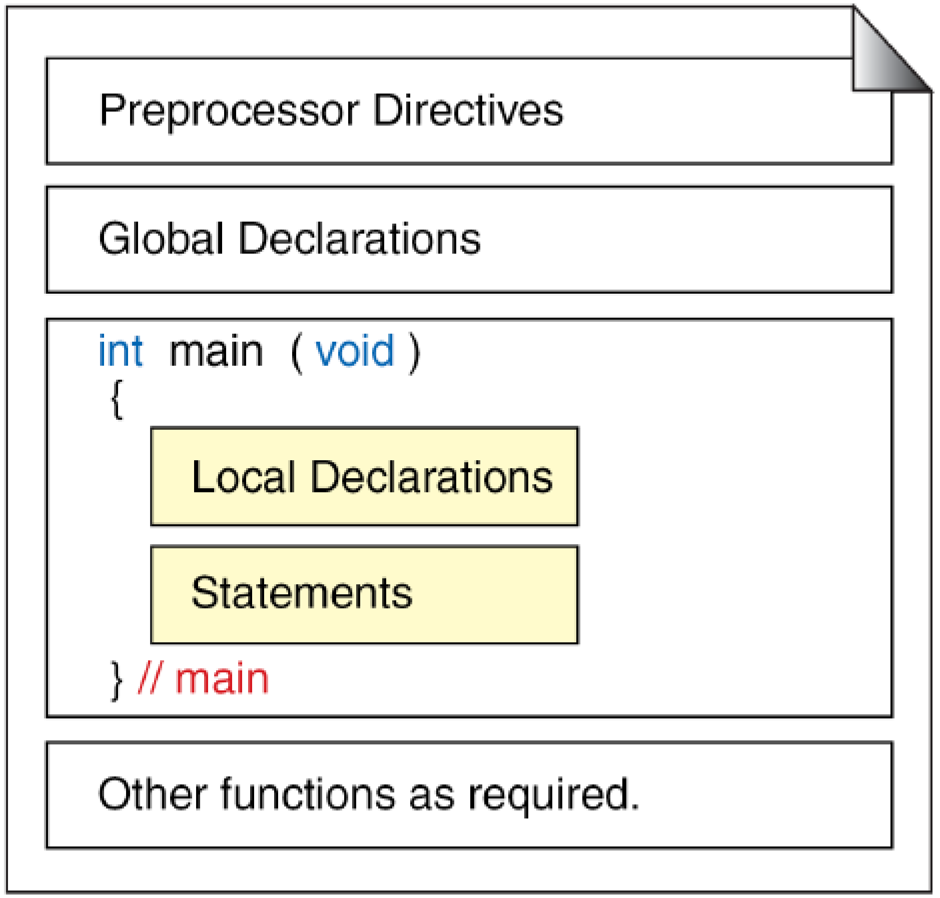
## C VS C++



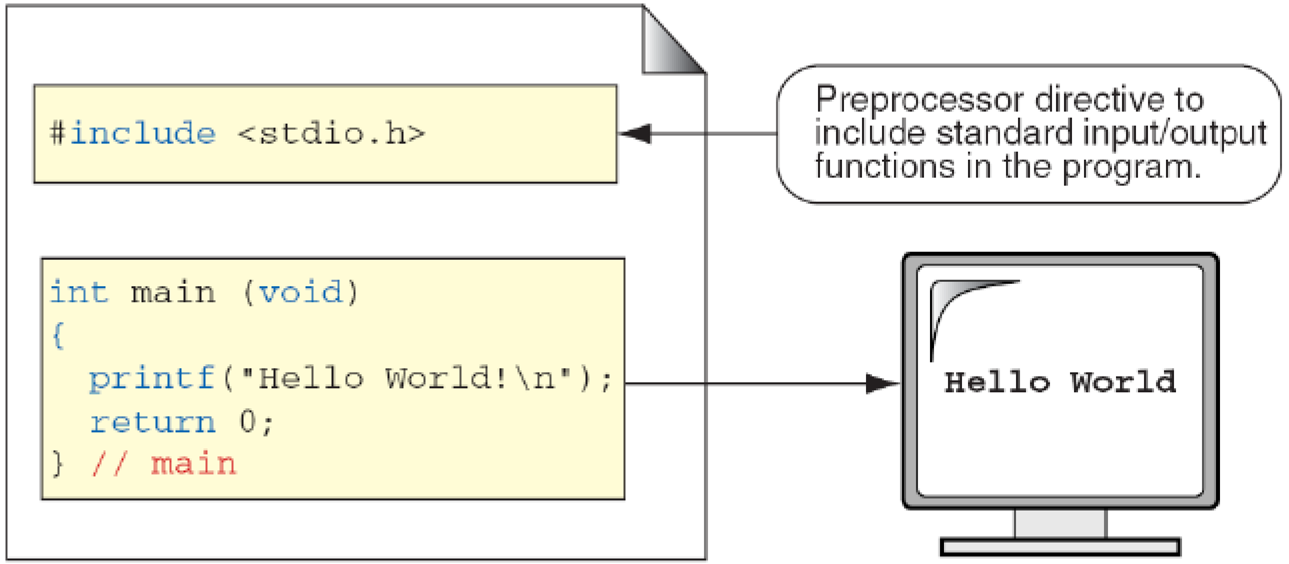
| C | C++ |
|---|---|
| C is a procedural (aka structural) programming language. | In addition to begin procedural, C++ is also an **object oriented programming language**. |
| In C language, the solution is achieved through a sequence of procedures or steps. Therefore, C is a function driven language. | C++ can model the whole solution in terms of objects and that makes the solution better organized. C++ is an object driven language. |
| Concept of **virtual functions** is not present in C. | C++ offers the facility of using virtual functions. |
| Operator overloading is not possible in C. | C++ allows **operator overloading**. |
| Data in C functions is not secured. Data can be easily accessed by other external functions. | All the data in C++ can be put inside objects. This provides better data security. |
| C is a *middle level language*. | C++ is a **high level language**. |

http://techwelkin.com/difference-between-c-and-c-plus-plus

# C Template



```
int  main  ( void )
{
    Local Declarations
    Statements
} // main
```


Preprocessor directive to include standard input/output functions in the program.

```
#include <stdio.h>

int main (void)
{
  printf("Hello World!\n");
  return 0;
} // main
```

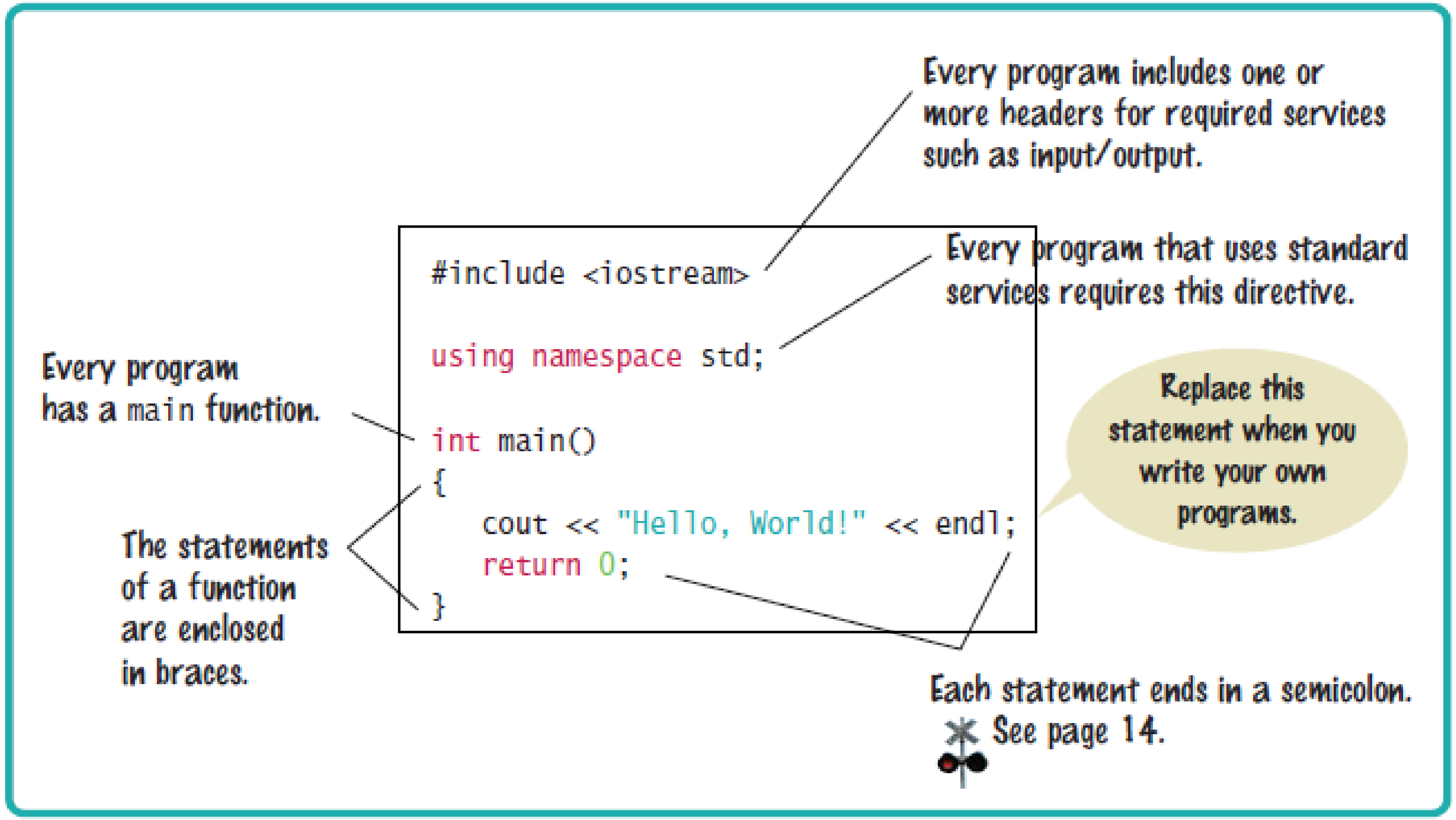
```
Hello World
```

# Hello World : The First C / C++ Program

In C

```
#include <stdio.h>

int main()
{
   printf("Hello, World!\n");
   return 0;
}
```

In C++



```
#include <iostream>

using namespace std;

int main()
{
   cout << "Hello, World!" << endl;
   return 0;
}
```

Every program includes one or more headers for required services such as input/output.
Every program that uses standard services requires this directive.
Every program has a main function.
Replace this statement when you write your own programs.
The statements of a function are enclosed in braces.
Each statement ends in a semicolon. See page 14.

Cay S. Horstmann. 2010. *C++ for Everyone* (2nd ed.). Wiley Publishing.

## Task 2-1

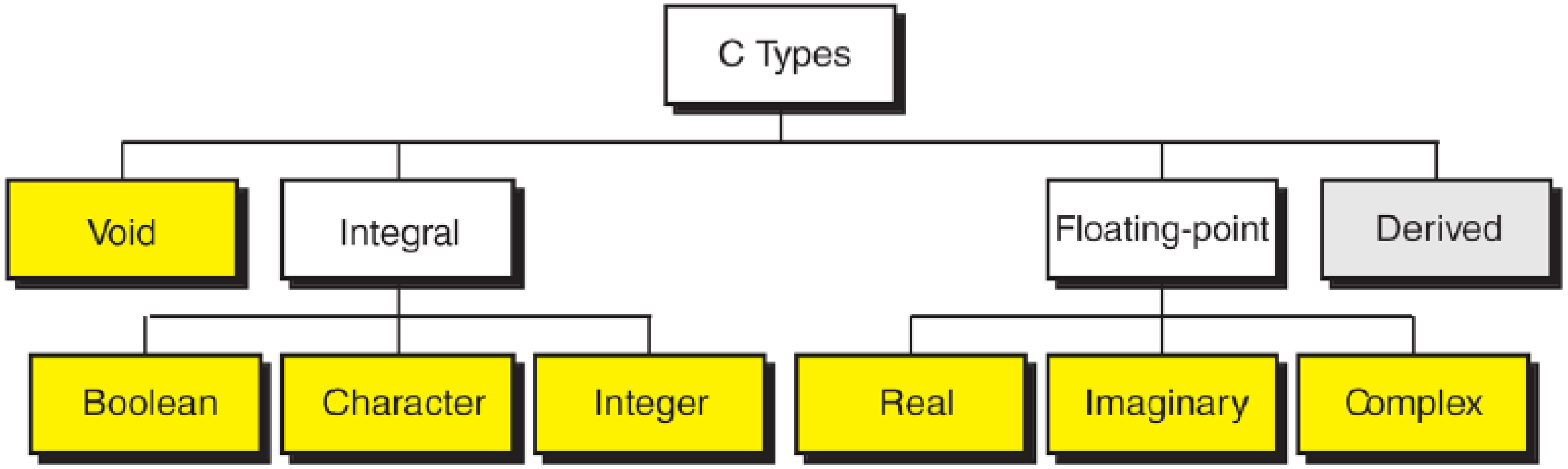
- Write the C/C++ program to Introduce yourself.

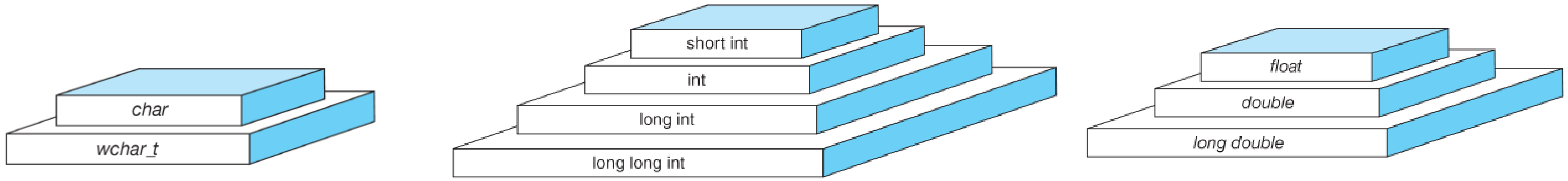
https://cpp.sh/

## C / C++ Concept

- Every C / C++ language program must have at least one function, namely **main()**. It is the first function called from the system starting the program.
- A semicolon **;** is used to indicate the end of an expression.
- Braces **{ }** are used to present the beginning and the end of the function contents.
- Double quotes **“ ”** are used to mark the beginning and the end of a text string.
- Slash-slash **//** slash-star/star-slash **/*….*/** are used as comment delimiters.

## Type


C Types
Void
Integral
Floating-point
Derived
Boolean
Character
Integer
Real
Imaginary
Complex


char
wchar_t
short int
int
long int
long long int
float
double
long double

# Variable Types and Sizes

| Type | Typical Bit Width | Typical Range |
|---|---|---|
| char | 1byte | -128 to 127 or 0 to 255 |
| unsigned char | 1byte | 0 to 255 |
| signed char | 1byte | -128 to 127 |
| int | 4bytes | -2147483648 to 2147483647 |
| unsigned int | 4bytes | 0 to 4294967295 |
| signed int | 4bytes | -2147483648 to 2147483647 |
| short int | 2bytes | -32768 to 32767 |
| unsigned short int | 2bytes | 0 to 65,535 |
| signed short int | 2bytes | -32768 to 32767 |
| long int | 8bytes | -2,147,483,648 to 2,147,483,647 |
| signed long int | 8bytes | -9,223,372,036,854,775,808 to 9,223,372,036,854,775,807 |
| unsigned long int | 8bytes | 0 to 18,446,744,073,709,551,615 |
| float | 4bytes | +/- 3.4e +/- 38 (~7 digits) |
| double | 8bytes | +/- 1.7e +/- 308 (~15 digits) |
| long double | 8bytes | +/- 1.7e +/- 308 (~15 digits) |
| wchar_t | 2 or 4 bytes | 1 wide character |

## Rules for Identifiers

1. First character must be alphabetic character or underscore. It may not have a space or a hyphen
2. Must consist only of alphabetic characters, digits, or underscores.
3. First 63 characters of an identifier are significant.
4. Cannot duplicate a keyword.
5. C / C++ is a case-sensitive language.

| Valid Names | | Invalid Name | |
|---|---|---|---|
| A | // Valid but poor style | $sum | // $ is illegal |
| student_name | | 2names | // First char digit |
| _aSystemName | | Sum-salary | // Contains hyphen |
| _Bool | // Booleand System id | Stdnt Nmbr | // Contains spaces |
| INT_MIN | // System Defined Value | Int | // Keyword |

## Keywords / Reserve Words in C \ C++

- For C

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| auto | const | double | float | int | short | struct | unsigned |
| break | continue | else | for | long | signed | switch | void |
| case | default | enum | goto | register | sizeof | typedef | volatile |
| char | do | extern | if | return | static | union | while |

- For C++

| | | | | |
|---|---|---|---|---|
| asm | dynamic_cast | namespace | reinterpret_cast | try |
| bool | explicit | new | static_cast | typeid |
| catch | false | operator | template | typename |
| class | friend | private | this | using |
| const_cast | inline | public | throw | virtual |
| delete | mutable | protected | true | wchar_t |

## Variables

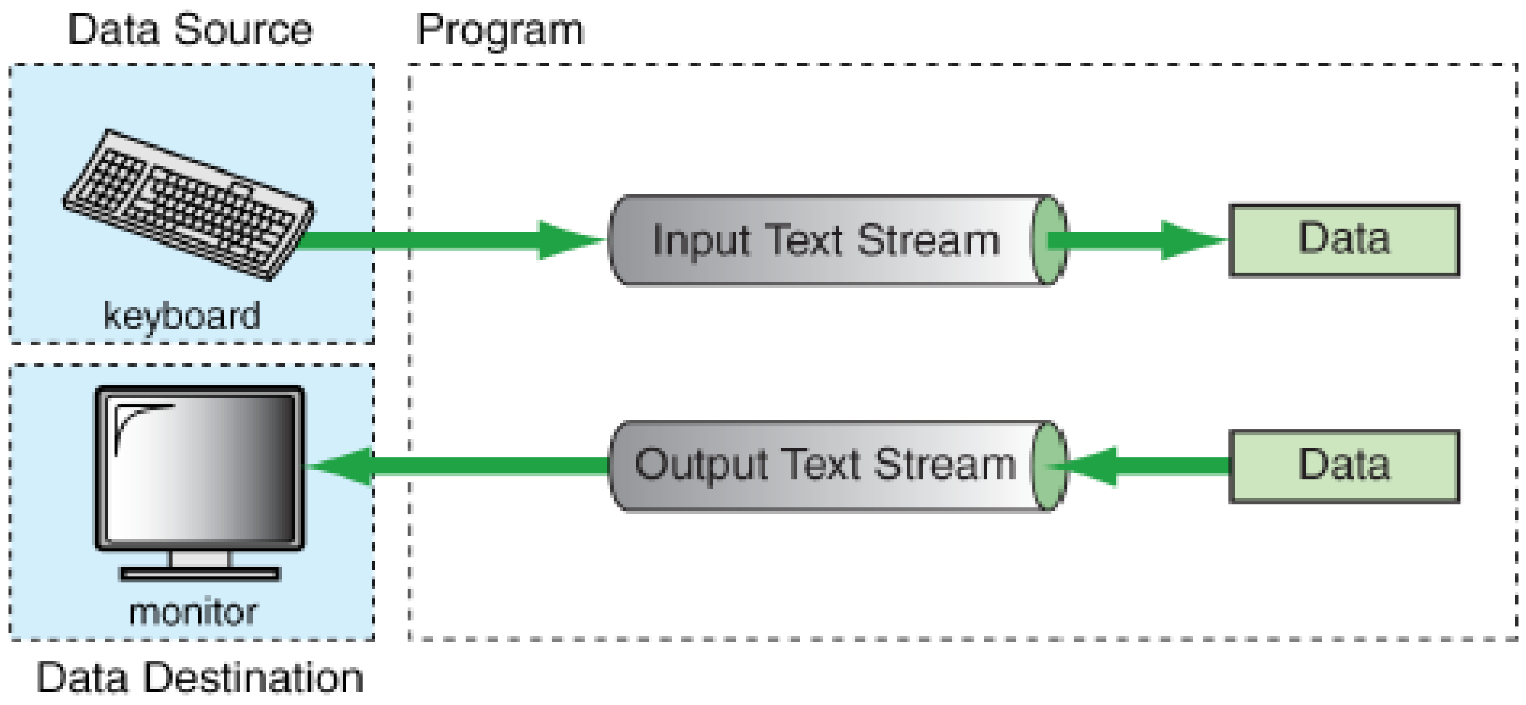
- Variables are names memory locations that have a type, such as integer or character, which is inherited from their type. The type determines the values that a variable may contain and the operations that may be used with its values.
- When a variable is defined, it is not initialized. We must initialize any variable requiring prescribed data when the function starts.

```
/* type variable_list; */
int     i, j, k;
char    c, ch;
float   f, salary;
double  d;
/* type variable_name = value; */
extern int d = 3, f = 5;    // declaration of d and f.
int d = 3, f = 5;           // definition and initializing d and f.
byte z = 22;                // definition and initializes z.
char x = 'x';               // the variable x has the value 'x'.
```

## Constants

- Constants are data values that cannot be changed during the execution of a program. Like variables, constants have a type.
- Use single quotes ‘ ‘ for character constants
- Use double quotes “ “ for string constants

## Input / Output



## C : scanf() & printf() functions

**`int scanf(const char *format, ...)`**

reads input from the standard input stream stdin and scans that input according to format provided.

**`int printf(const char *format, ...)`**

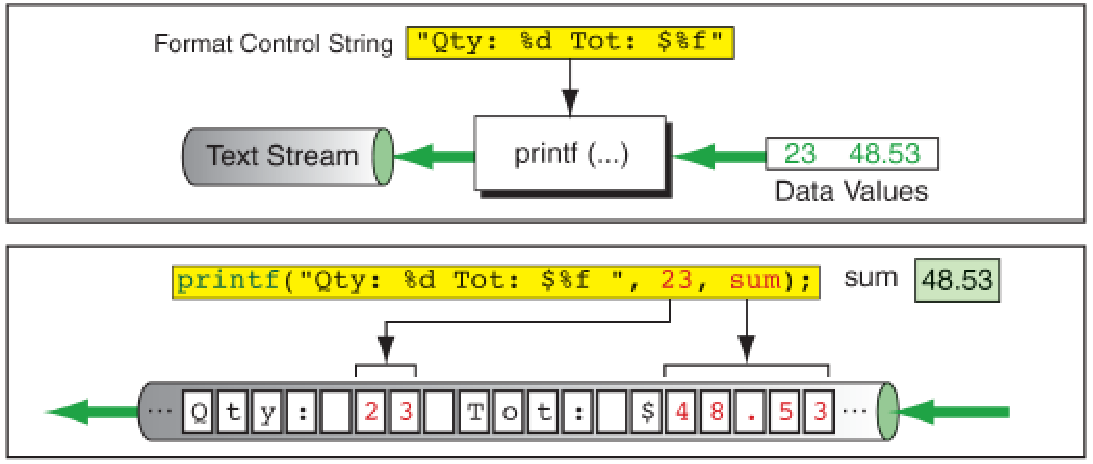
writes output to the standard output stream stdout and produces output according to a format provided.

The format can be a simple constant string, but you can specify %s, %d, %c, %f, etc., to print or read strings, integer, character or float respectively

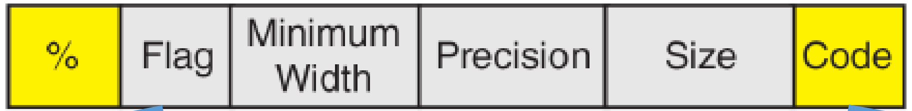
```
#include <stdio.h>
int main( )
{
   char str[100];
   int i;

   printf( "Enter a value :");
   scanf("%s %d", str, &i);

   printf( "\nYou entered: %s %d ", str, i);

   return 0;
}
```

# Output

(a) Basic Concept



(b) Implementation



| Flag Type | Flag Code | Formatting |
|---|---|---|
| Justification | None | right justified |
| | – | left justified |
| Padding | None | space padding |
| | 0 | zero padding |
| Sign | None | positive value: no sign<br>negative value: – |
| | + | positive value: +<br>negative value: – |
| | **Space** | positive value: space<br>negative value: – |

| Type | Size[a] | Code | Example |
|---|---|---|---|
| `char` | None | `c` | `%c` |
| `short int` | `h` | `d` | `%hd` |
| `int` | None | `d` | `%d` |
| `long int` | None | `d` | `%ld` |
| `long long int` | `ll` | `d` | `%lld` |
| `float` | None | `f` | `%f` |
| `double` | None | `f` | `%f` |
| `long double` | `L` | `f` | `%Lf` |

a. Size is discussed in the next section.

## Output Examples

```
1 printf("%d%c%f", 23, 'z', 4.1);
  >> 23z4.100000
2 printf("%d %c %f", 23, 'z', 4.1);
  >> 23 z 4.100000
3 int num1 = 23;
  char zee = 'z';
  float num2 = 4.1;
  printf("%d %c %f", num1, zee, num2);
  >> 23 z 4.100000
4 printf("%d\t%c\t%5.1f\n", 23, 'z', 14.2);
  printf("%d\t%c\t%5.1f\n", 107, 'A', 53.6);
  printf("%d\t%c\t%5.1f\n", 1754, 'F', 122.0);
  printf("%d\t%c\t%5.1f\n", 3, 'P', 0.1);
  >> 23      z      14.2
     107     A      53.6
     1754    F      122.0
     3       P      0.1
```

## Output Examples

```
5 printf("The number%dis my favorite.", 23);
  >> The number23is my favorite.
6 printf("The number is %6d", 23);
  >> The number is     23
     ^^^^^^^^^^^^^^^^^^^^
7 printf("The tax is %6.2f this year.", 233.12);
  >> The tax is 233.12 this year.
8 printf("The tax is %8.2f this year.", 233.12);
  >> The tax is   233.12 this year.
     ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
9 printf("The tax is %08.2f this year.", 233.12);
  >> The tax is 00233.12 this year.
10 printf("\"%8c   %d\"", 'h', 23);
  >> "       h   23‟
     ^^^^^^^^^^^^^^^
```

## Output Examples

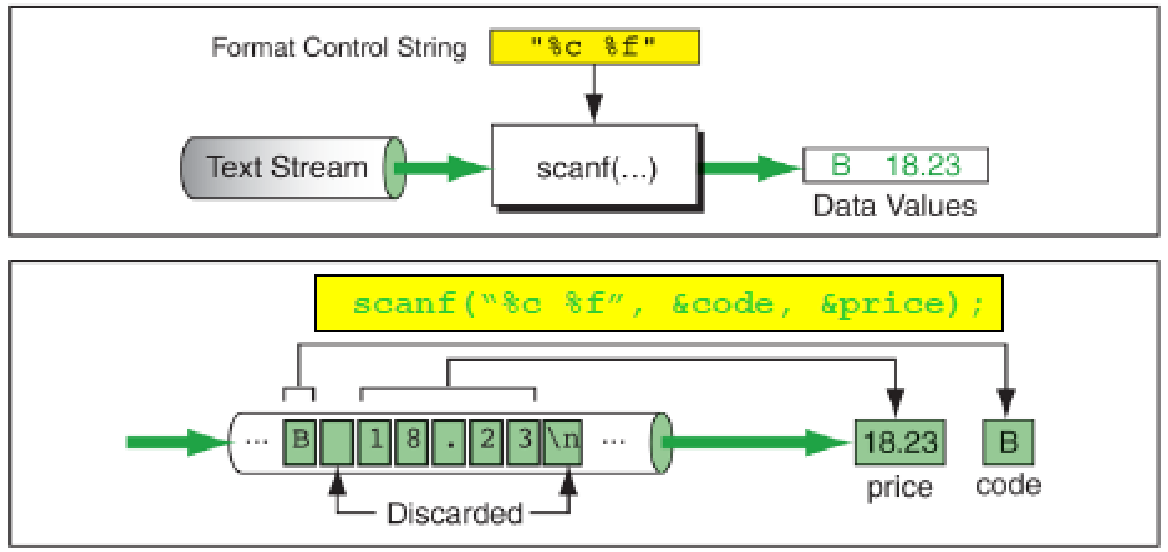
```
11 printf("This line disappears.\r...A new line\n");
   printf("This is the bell character \a\n");
   printf("A null character\0kills the rest of the line\n");
   printf("\nThis is \'it\' in single quotes\n");
   printf("This is \"it\" in double quotes\n");
   printf("This is \\the escape character it self\n");
   >> ...A new line
      This is the bell character
      A null character

      This is 'it' in single quotes
      This is "it" in double quotes
      This is \the escape character it self
12 printf("|%-+8.2f| |%0+8.2f| | %-0+8.2f|", 1.2, 2.3, 3.4);
   >> |+1.20   | |+0002.30| | +3.40   |
      ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
```

## Common Output Errors

```
1 printf("%d %d %d\n", 44, 55);
  >> 44 55 1638280
2 printf("%d %d\n", 44, 55, 66);
  >> 44 55
3 float x = 123.45;
  printf("The data are: %d\n", x);
  >> The data are: -1073741824
```

## Input

(a) Basic Concept



(b) Implementation

| | | | | | |
|---|---|---|---|---|---|
| % | Flag | Maximum Width | | Size | Code |

## Input Examples

```
1 214 156 14z
  scanf("%d%d%d%c", &a, &b, &c, &d);

2 214 156 14 z
  scanf("%d%d%d %c", &a, &b, &c, &d);

3 2314 15 2.14
  scanf("%d %d %f", &a, &b, &c);

4 14/26 25/66
  scanf("%2d/%2d %2d/%2d", &num1, &den1, &num2, &den2);

5 11-25-56
  scanf("%d-%d-%d", &a, &b, &c);
```

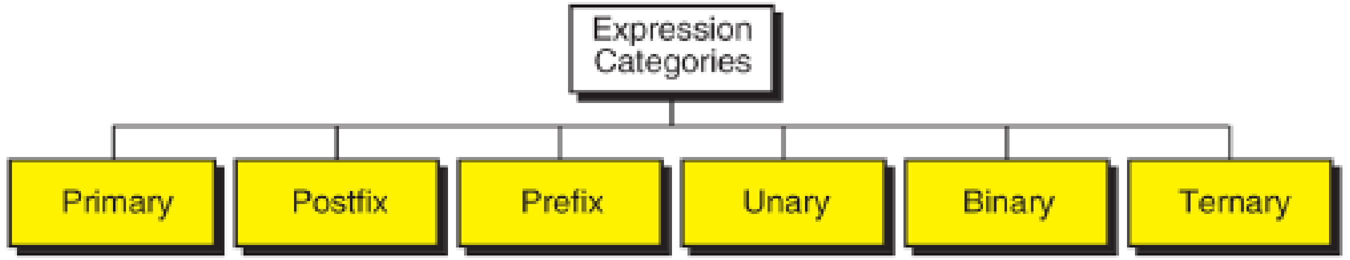
## Common Input Errors

```
1 int a = 0;
  scanf("%d",a);
  printf("%d\n",a);
  >> 234 (Input)
     0 (Output)

2 float a = 2.1;
  scanf("%5.2f", &a);
  printf("%5.2f", a);
  >> 74.35 (Input)
     2.10 (Output)
```

```
3 int a;
  int b;
  scanf("%d%d%d", &a, &b);
  printf("%d %d\n", a, b);
  >> 5 10 (Input)
     5 10 (Output)

4 int a = 1;
  int b = 2;
  int c = 3;
  scanf("%d%d",&a, &b, &c);
  printf("%d %d %d\n", a, b, c);
  >> 5 10 15 (Input)
     5 10 3 (Output)
```

## Expression

- An expression is a sequence of operands and operators that reduces to a single value.



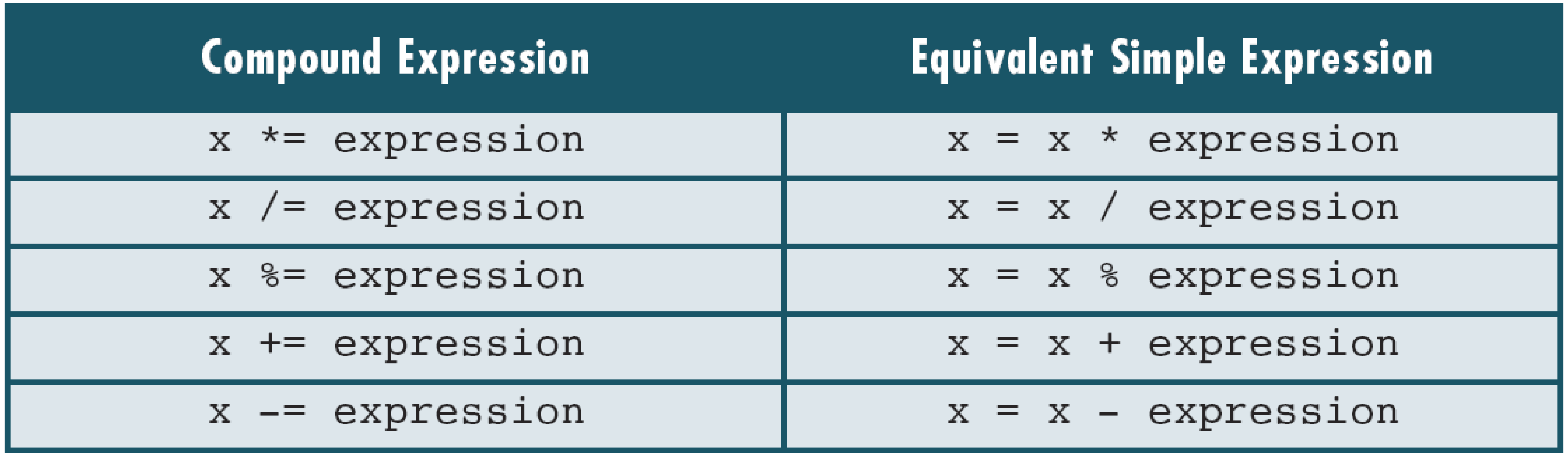
- (a++) has the same effect as (a = a + 1)

  x = a++ : x=a and then a = a+1

- (++a) has the same effect as (a = a + 1)

  x = ++a : a = a+1 and then x = a

## Expression



| Compound Expression | Equivalent Simple Expression |
|---|---|
| `x *= expression` | `x = x * expression` |
| `x /= expression` | `x = x / expression` |
| `x %= expression` | `x = x % expression` |
| `x += expression` | `x = x + expression` |
| `x -= expression` | `x = x - expression` |

```
a * 4 + b / 2 -c * b
d = a * 4 + b / 2 -c * b
f = a * 4 + b++ / 2 -c * --b
```

## Increment and Decrement Operators

- Increment operator : ++

  Pre-increment : ++variable

  Post-increment : variable++

- Decrement operator : --

  Pre-increment : --variable

  Post-increment : variable--

```
x = 5;
y = ++x;
```

[redacted]

```
x = 5;
y = x++;
```

[redacted]

```
a = 5;
b = 2 + (--a);
```

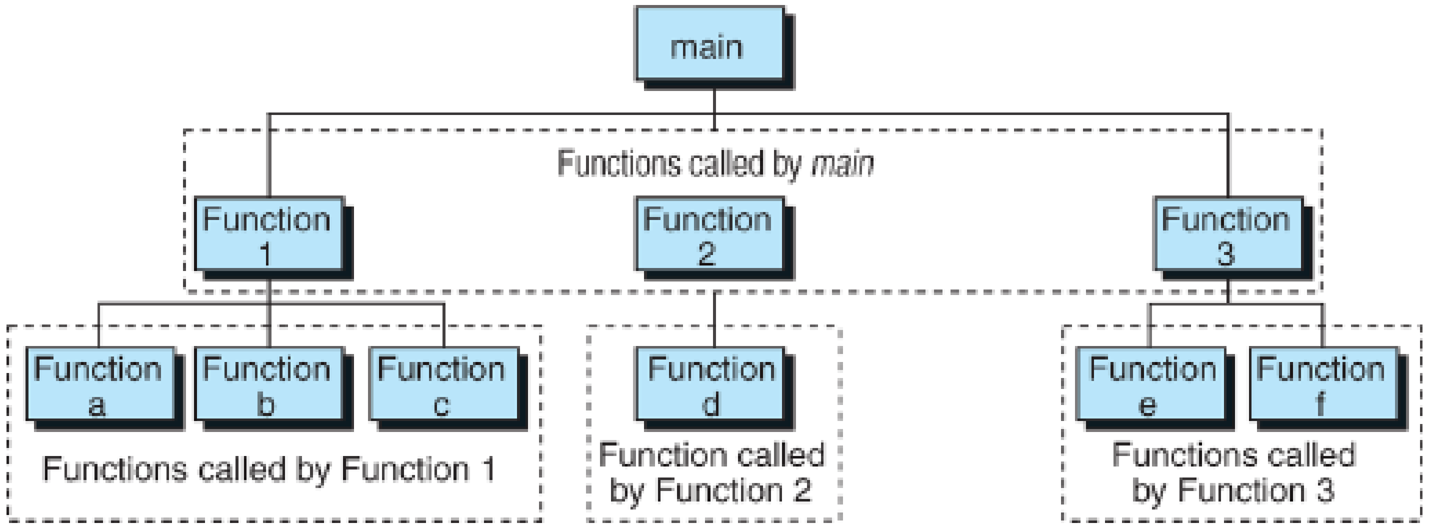
[redacted]

```
a = 5;
b = 2 + (a--);
```

[redacted]

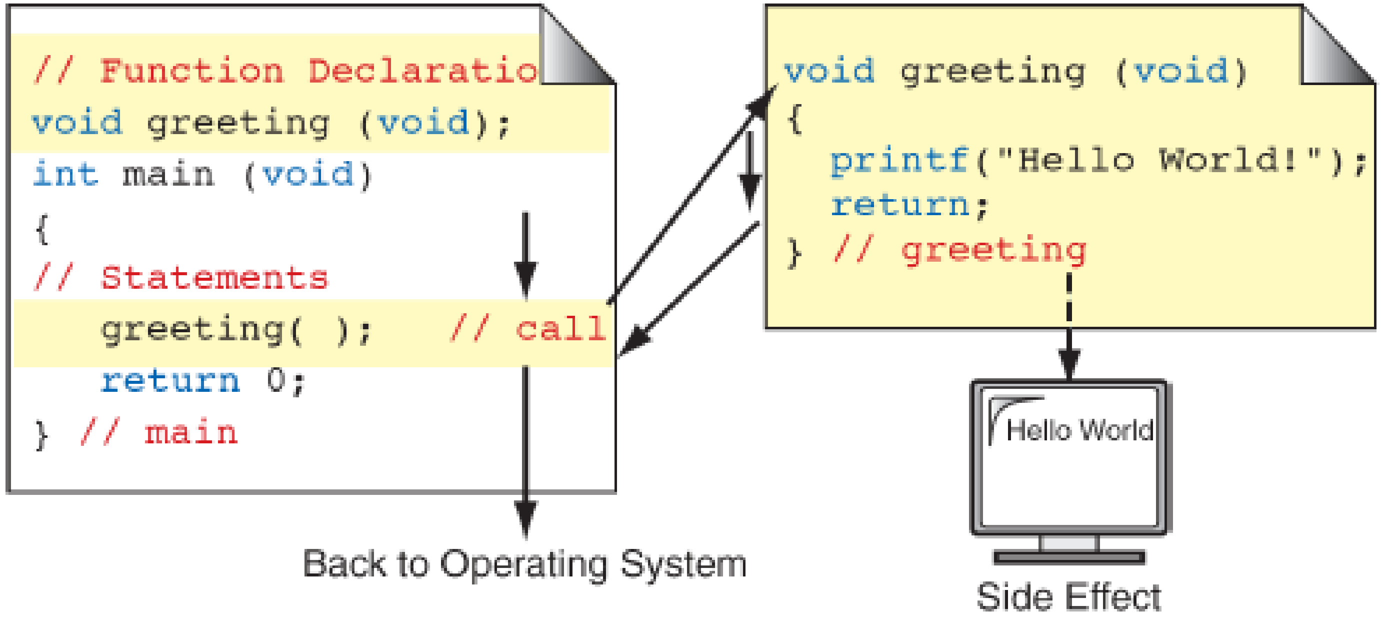
## Function in C

- A C program is made of one or more functions, one and only one of which must be named main.
- The execution of the program always starts with main, but it can call other functions to do some part of the job.


main
Functions called by *main*
Function 1
Function 2
Function 3
Function a
Function b
Function c
Functions called by Function 1
Function d
Function called by Function 2
Function e
Function f
Functions called by Function 3

## User-Defined Function

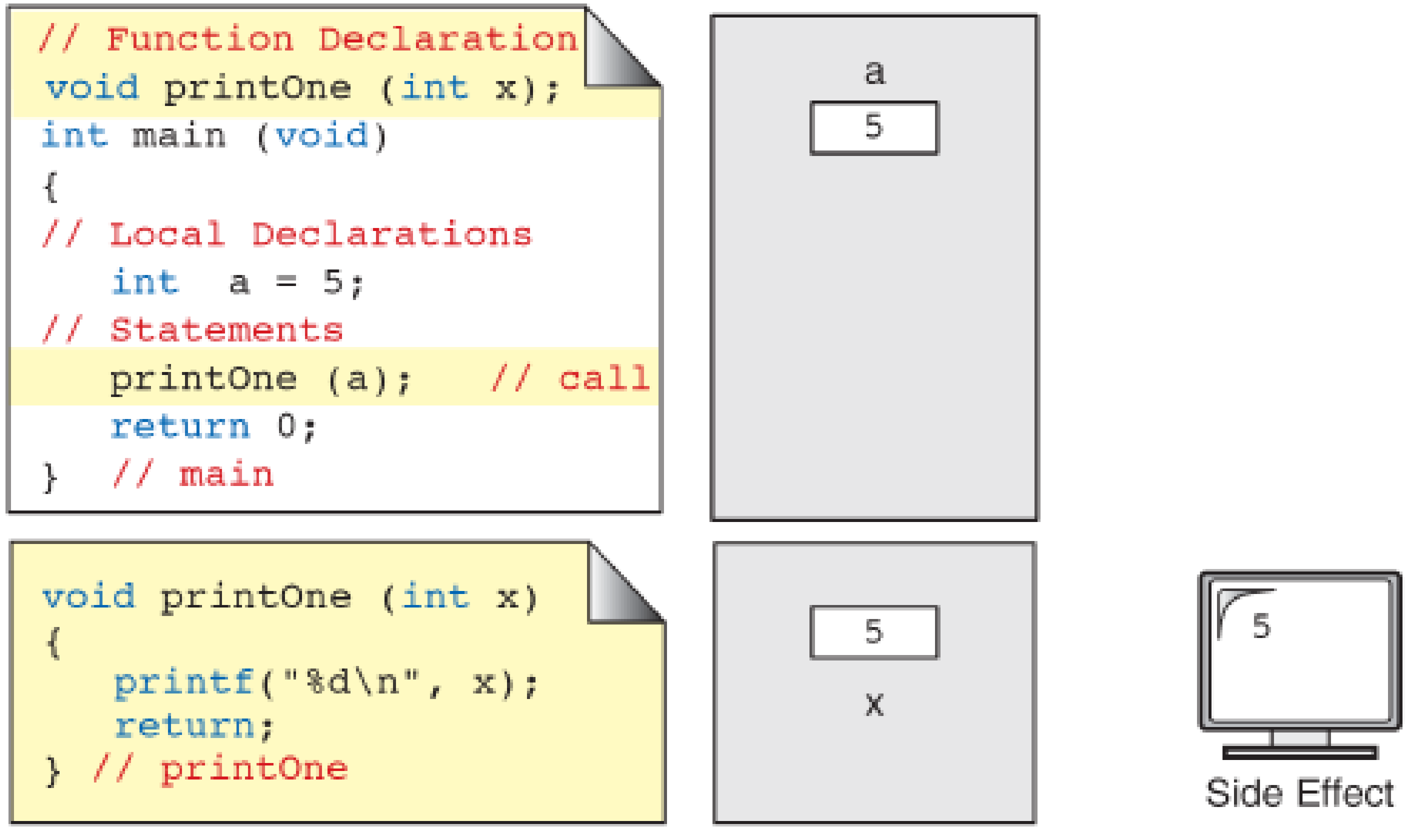
- Functions must be both declared and defined. The function declaration gives the whole picture of the function that needs to be defined later. The function definition contains the code for a function.



```
// Function Declaratio
void greeting (void);
int main (void)
{
// Statements
    greeting( );    // call
    return 0;
} // main
```

```
void greeting (void)
{
   printf("Hello World!");
   return;
} // greeting
```

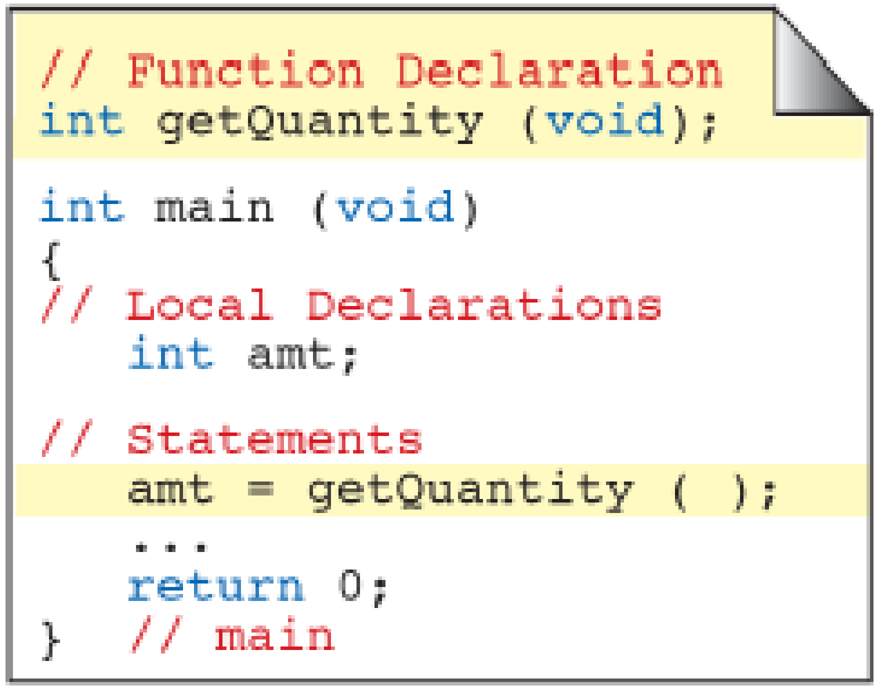
## void Function with Parameters



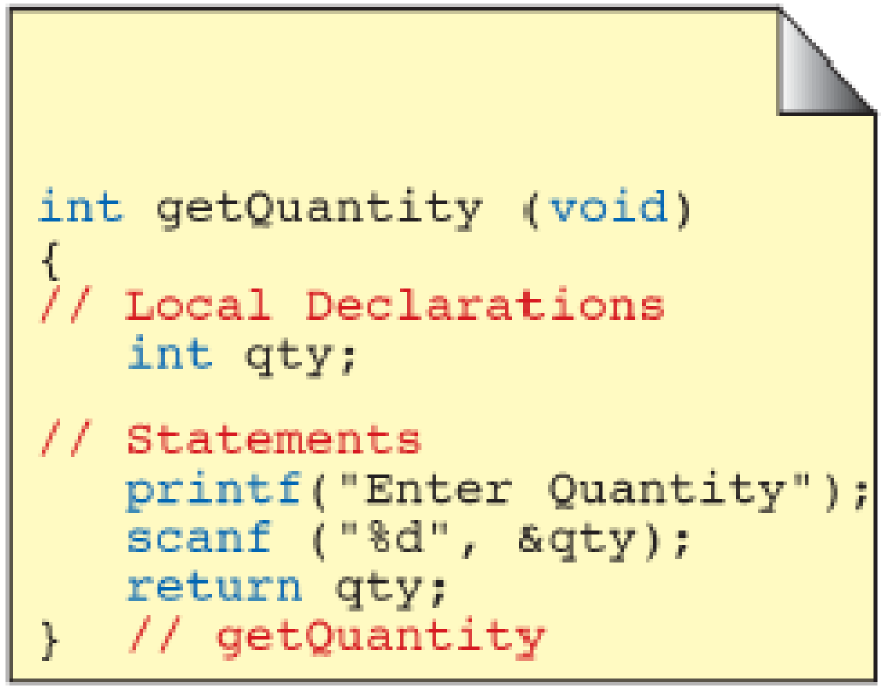
```
// Function Declaration
void printOne (int x);
int main (void)
{
// Local Declarations
   int  a = 5;
// Statements
   printOne (a);    // call
   return 0;
}  // main
```

```
void printOne (int x)
{
   printf("%d\n", x);
   return;
} // printOne
```

## Non-void Function without Parameters

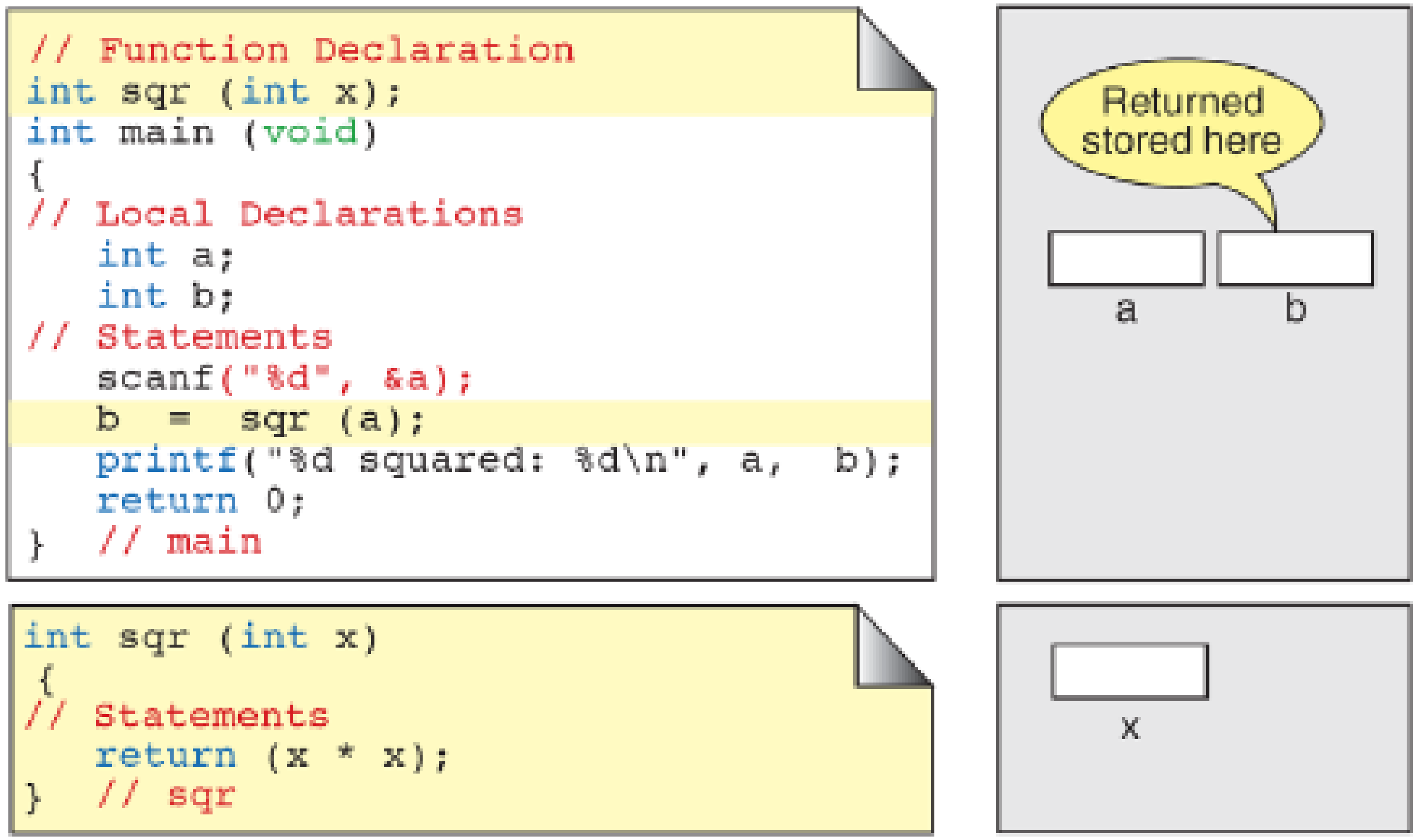


```
// Function Declaration
int getQuantity (void);

int main (void)
{
// Local Declarations
   int amt;

// Statements
   amt = getQuantity ( );
   ...
   return 0;
}  // main
```



```
int getQuantity (void)
{
// Local Declarations
   int qty;

// Statements
   printf("Enter Quantity");
   scanf ("%d", &qty);
   return qty;
}  // getQuantity
```

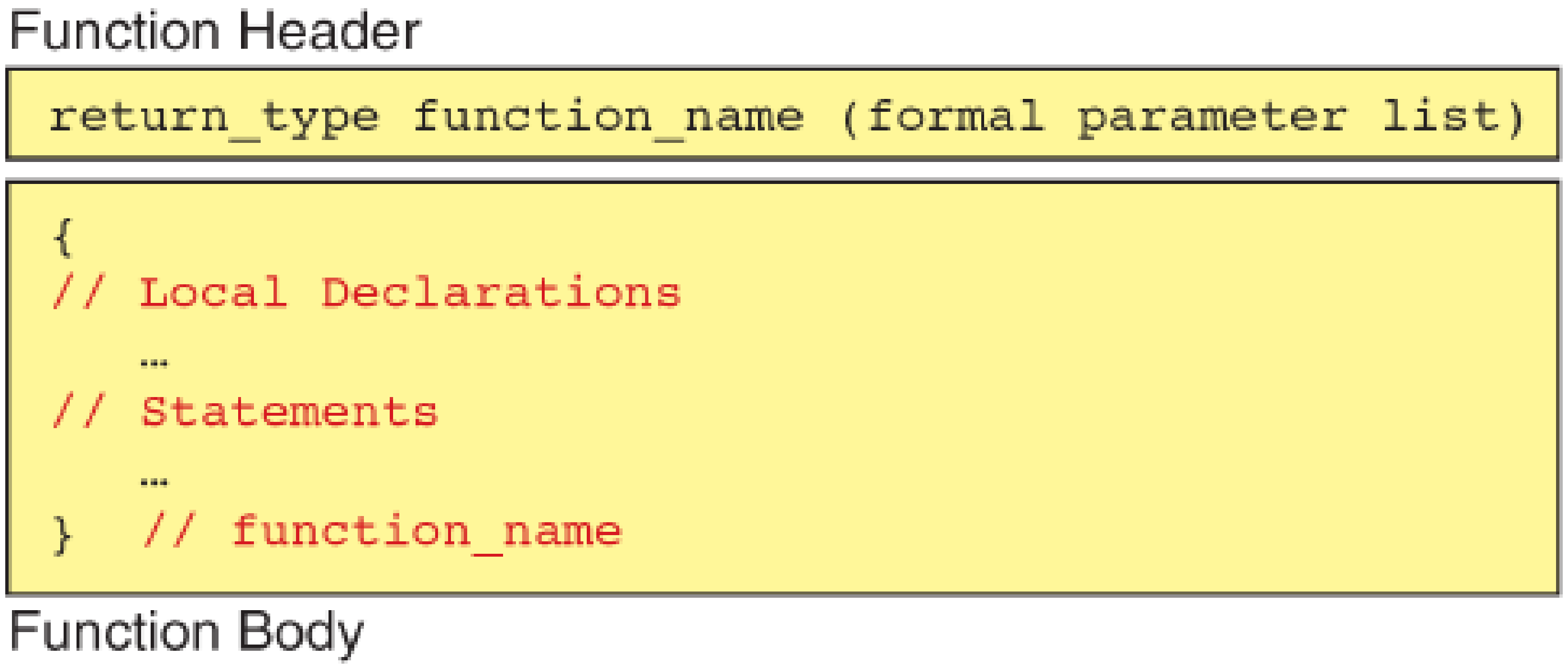
## Calling a Function that Returns a Value



```
// Function Declaration
int sqr (int x);
int main (void)
{
// Local Declarations
   int a;
   int b;
// Statements
   scanf("%d", &a);
   b  =  sqr (a);
   printf("%d squared: %d\n", a,  b);
   return 0;
}  // main
```

```
int sqr (int x)
 {
// Statements
   return (x * x);
}  // sqr
```

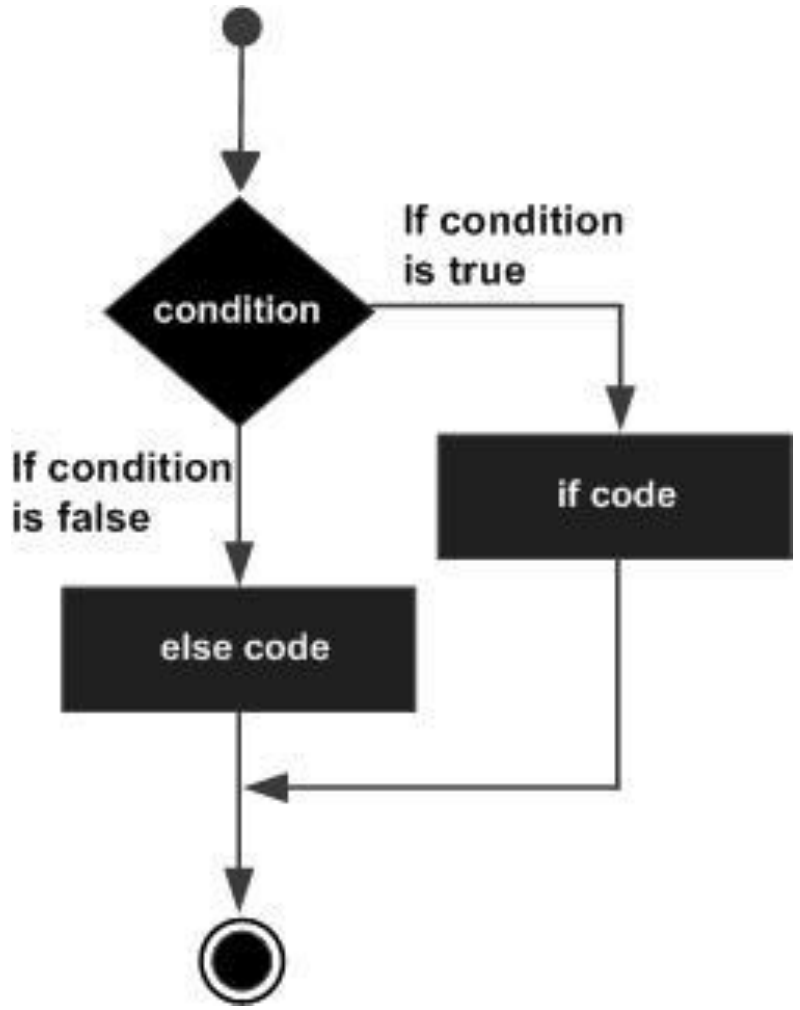
## Function Definition



```
return_type function_name (formal parameter list)
```

```
{
// Local Declarations
   ...
// Statements
   ...
}  // function_name
```

## Decision Making Structure : if



C programming language assumes any **non-zero** and **non-null** values as **true**, and if it is either **zero** or **null**, then it is assumed as **false** value.

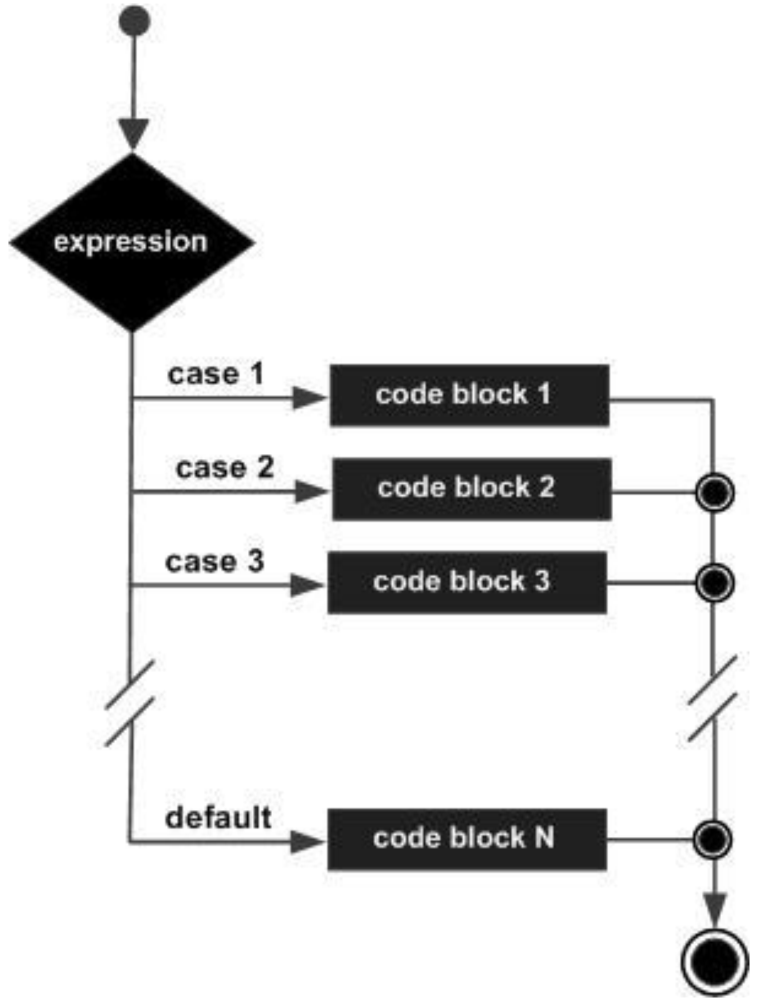
```
if(boolean_expression)
{
   /* statement(s) will execute if the boolean expression is true */
}
else
{
  /* statement(s) will execute if the boolean expression is false */
}
```

**The ? : Operator:**

The value of a ? expression is determined like this: Exp1 is evaluated. If it is true, then Exp2 is evaluated and becomes the value of the entire ? expression. If Exp1 is false, then Exp3 is evaluated and its value becomes the value of the expression.

```
Exp1 ? Exp2 : Exp3;
```

## Decision Making Structure : switch



```
switch(expression){
    case constant-expression  :
       statement(s);
       break; /* optional */
    case constant-expression  :
       statement(s);
       break; /* optional */

    /* you can have any number of case statements */
    default : /* Optional */
       statement(s);
}
```

## Task 2-2

```
Please enter temperature : 30.0 C
30.00 C = 303.15 K = 86.00 F
```

จงเขียนโปรแกรมแปลงหน่วยอุณหภูมิ โดยรับค่าอุณหภูมิและหน่วยของอุณหภูมิจากผู้ใช้ และกำหนดให้

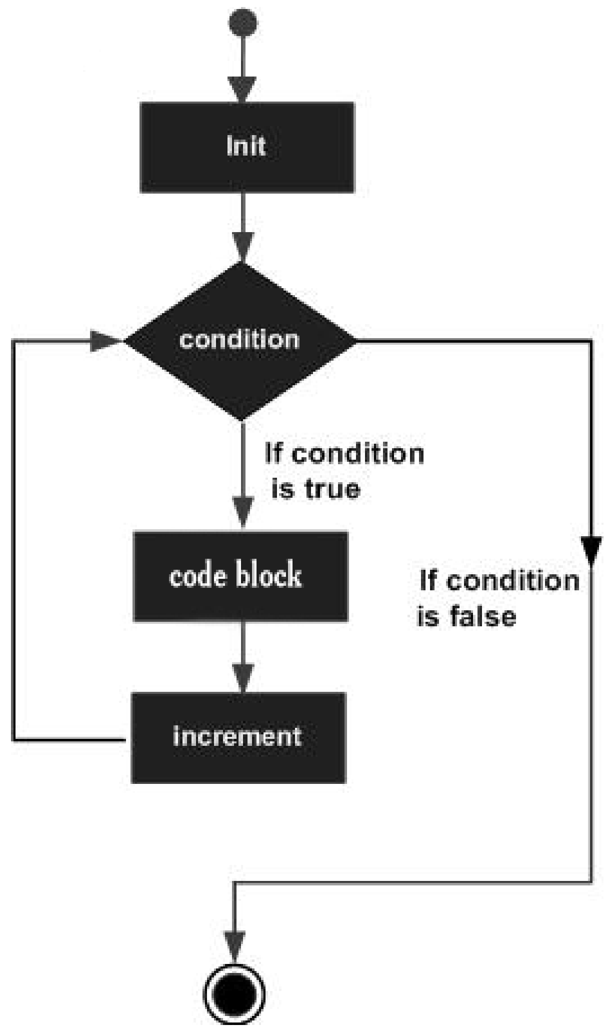
| | | |
|---|---|---|
| 30 C | หมายถึง | 30 องศาเซลเซียส |
| 274.5 K | หมายถึง | 274.5 องศาเคลวิน |
| 85.3 F | หมายถึง | 85.3 องศาฟาเรนไฮต์ |

โปรแกรมจะคำนวณและแสดงผลลัพธ์ในการแปลงเป็นหน่วยอุณหภูมิในรูปแบบที่เหลือ (เช่น ใส่ C จะแสดงค่า K และ F)
โดยใช้สมการดังนี้

$$K = C + 273.15 \qquad F = \frac{9}{5}C + 32 \qquad C = \frac{5}{9}(F - 32)$$

$$C = K - 273.15 \qquad F = 1.8K - 459.69 \qquad K = \frac{F + 459.69}{1.8}$$

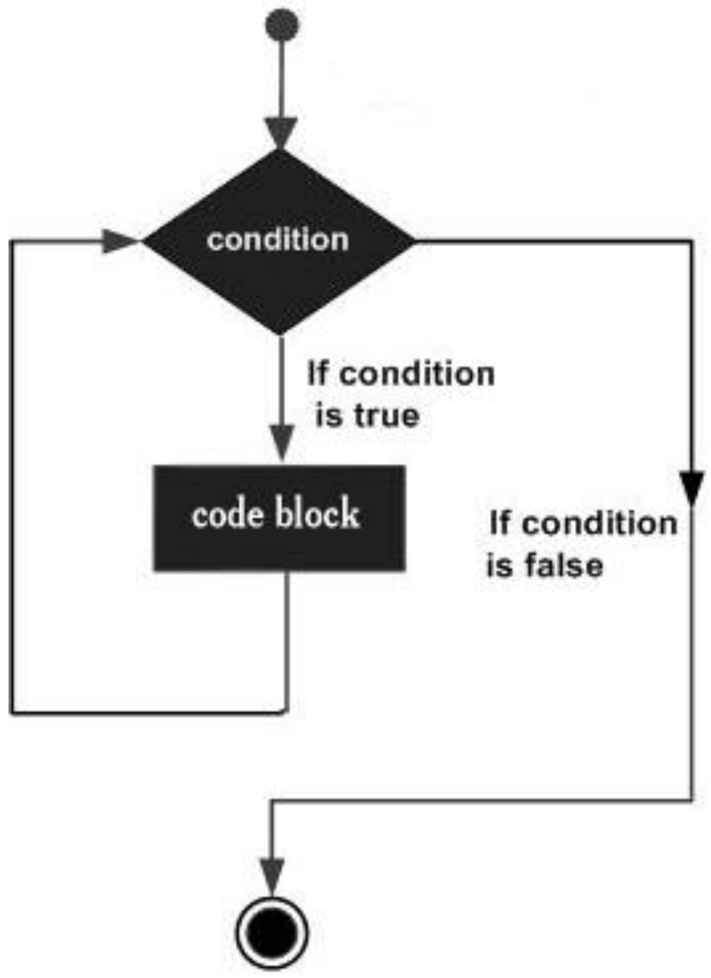
## Loop Structure : for



```
for ( init; condition; increment )
{
   statement(s);
}
```

```
#include <stdio.h>

int main ()
{
   /* for loop execution */
   for( int a = 10; a < 20; a = a + 1 )
   {
      printf("value of a: %d\n", a);
   }

   return 0;
}
```

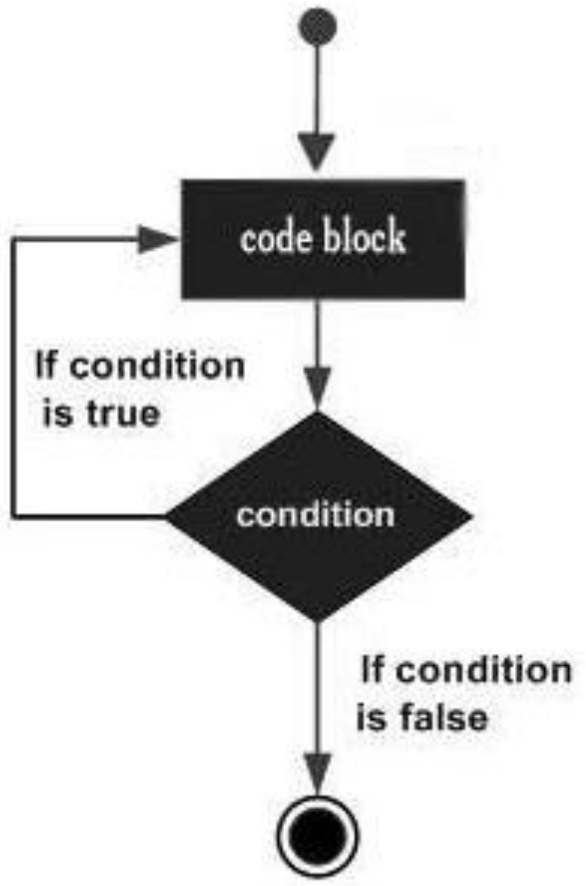
## Loop Structure : while



```
while(condition)
{
   statement(s);
}
```

```
#include <stdio.h>

int main ()
{
   /* local variable definition */
   int a = 10;

   /* while loop execution */
   while( a < 20 )
   {
      printf("value of a: %d\n", a);
      a++;
   }

   return 0;
}
```

# Loop Structure : do...while



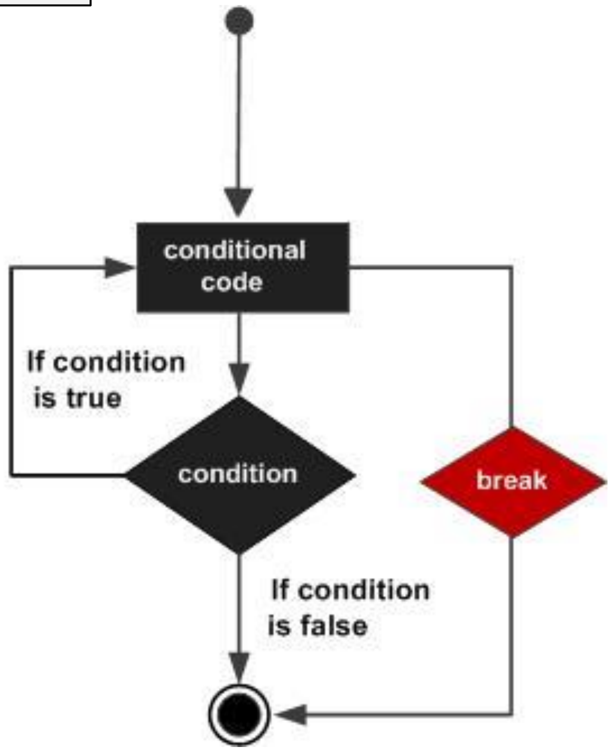
```
do
{
   statement(s);

}while( condition );
```

```
#include <stdio.h>

int main ()
{
   /* local variable definition */
   int a = 10;

   /* do loop execution */
   do
   {
       printf("value of a: %d\n", a);
       a = a + 1;
   }while( a < 20 );

   return 0;
}
```

# Loop Control Statements

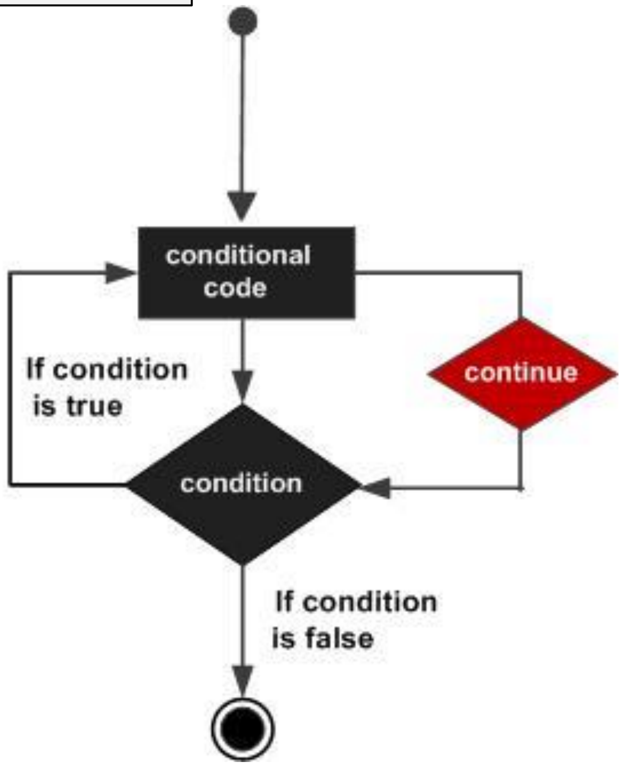
## break statement

```
break;
```



## continue statement

```
continue;
```



## Nest Loop Structure

```
for ( init; condition; increment )
{
   for ( init; condition; increment )
   {
      statement(s);
   }
   statement(s);
}
```

```
while(condition)
{
   while(condition)
   {
      statement(s);
   }
   statement(s);
}
```

```
do
{
   statement(s);
   do
   {
      statement(s);
   }while( condition );

}while( condition );
```

## Task 2-3

จงเขียนโปรแกรมแสดงสัญลักษณ์ “*” เป็นรูปสามเหลี่ยมดังแสดงในตัวอย่าง โดยที่ผู้ใช้สามารถป้อนเลือกชนิดของสามเหลี่ยม และความสูงของสามเหลี่ยมที่จะแสดงได้

(ตัวอย่าง)

```
Please enter the type of triangle : 1
Please enter the height of triangle : 5
*
* *
* * *
* * * *
* * * * *
* * * *
* * *
* *
*
```

```
Please enter the type of triangle : 2
Please enter the height of triangle : 5
        *
      * * *
    * * * * *
  * * * * * * *
* * * * * * * * *
```

```
Please enter the type of triangle : 3
Please enter the height of triangle : 5
    *
   * *
  * * *
 * * * *
* * * * *
 * * * *
  * * *
   * *
    *
```

```
/*****************************************************/
/* Program       : Task 2-3                          */
/* Description   : program for printing triangle     */
/* Programmer    : student-id1 name1 surname1        */
/*               : student-id2 name2 surname2        */
/* Group         : group-name (optional)             */
/* Section       : A or B                            */
/* Date          : 22/01/2018                        */
/*****************************************************/

#include <stdio.h>

// main program
int main()
{
    // print the greeting message
    printf("Welcome to FRA142 : Computer Programming for Robotics Engineering II\n");
    return 0;
}
```