Basic Logics

Review

- A variable is a name referencing to a memory location.
- A data type defines: How the values are stored and how the operations on those values are performed.
- 4 primitive data types in C: **int**, **char**, **float**, **double**
- Data stored are in binary format
- Declare a variable in C: **data_type identifier [=initialValue];**
- An identifier begin with a letter or ‘_’, the later symbols can be letters, digits and ‘_’
- An user-defined identifier must not a C keyword.
- Expression is a valid association of constants, variables, operators and functions.

Objectives

After studying this section, you should be able to:

- How to develop a C- program?

  → **Logic constructs**

- When I develop a C-program, what are things that I should follow?

  → **Programming styles**

- How I can understand a program?

  → **Walkthroughs and debug**

Contents

- Logic constructs
- Programming Styles
- Walkthroughs

Logic Constructs

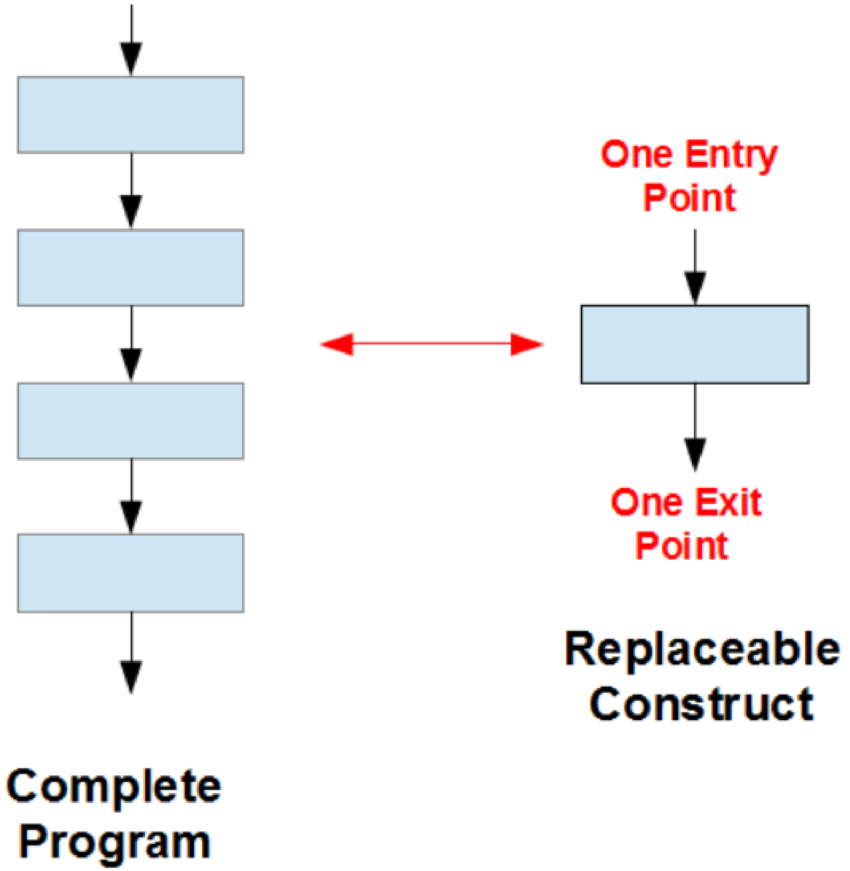
Logic constructs

Logic constructs: Expressions enable us to write programs that perform calculations and execute statements in a sequential order.

- **Structured Programming**
- **Logic constructs:**
  - Sequence constructs
  - Selection constructs
  - Iteration constructs

1. Structured Programming

- Structure of a program code should be organize in a manner so that it is understandable, testable and readily modifiable.
- It consists of simple logical constructs, each of which has one entry point and one exit point.
- The beginning step for developing a program is DESIGN using
  - **Pseudo-coding or**
  - **Flow charting**



Structured Programming - Pseudo-code

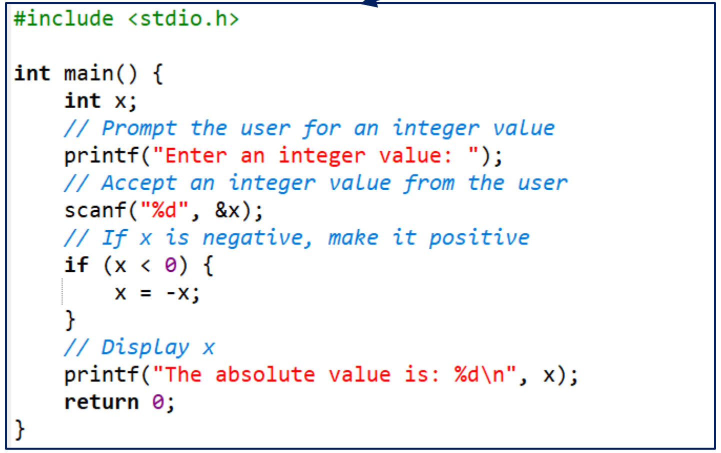
- **Pseudo-code** is a set of shorthand notes in a human (non-programming) language that itemizes the key steps in the sequence of instructions that produce a programming solution.
- **Example 1**: Calculating the absolute value of an integer inputted from the keyboard.

```
✓ Prompt the user for an integer value
✓ Accept an integer value from the user and store it in x
✓ If  x  is negative then x = -x
✓ Display x
```

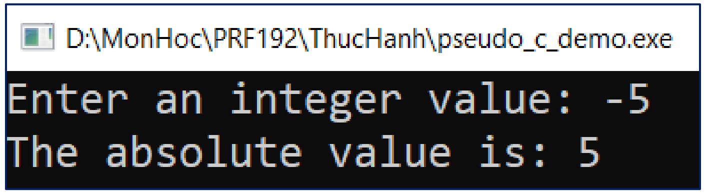
Pseudo-code → a program

- ✓ Prompt the user for an integer value
- ✓ Accept an integer value from the user and store it in x
- ✓ If x is negative then x = -x
- ✓ Display x

C



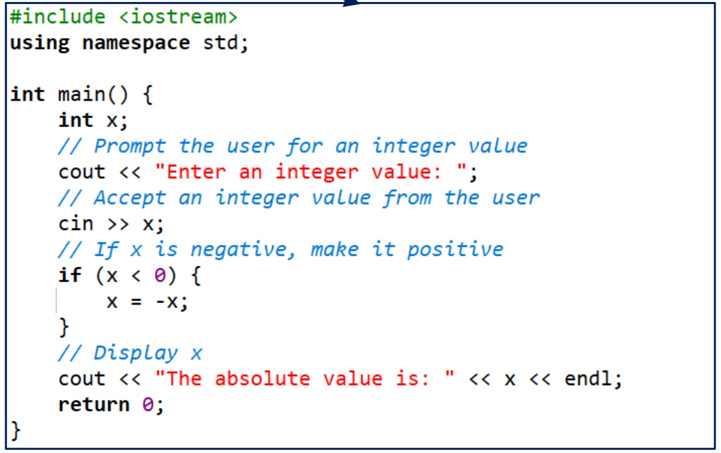
```c
#include <stdio.h>

int main() {
    int x;
    // Prompt the user for an integer value
    printf("Enter an integer value: ");
    // Accept an integer value from the user
    scanf("%d", &x);
    // If x is negative, make it positive
    if (x < 0) {
        x = -x;
    }
    // Display x
    printf("The absolute value is: %d\n", x);
    return 0;
}
```



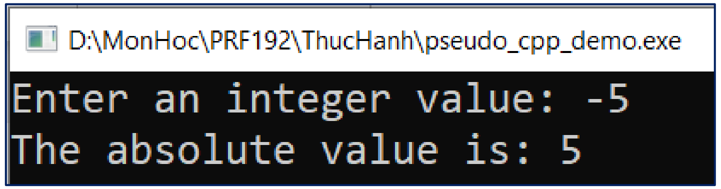
D:\MonHoc\PRF192\ThucHanh\pseudo_c_demo.exe

```
Enter an integer value: -5
The absolute value is: 5
```

C++



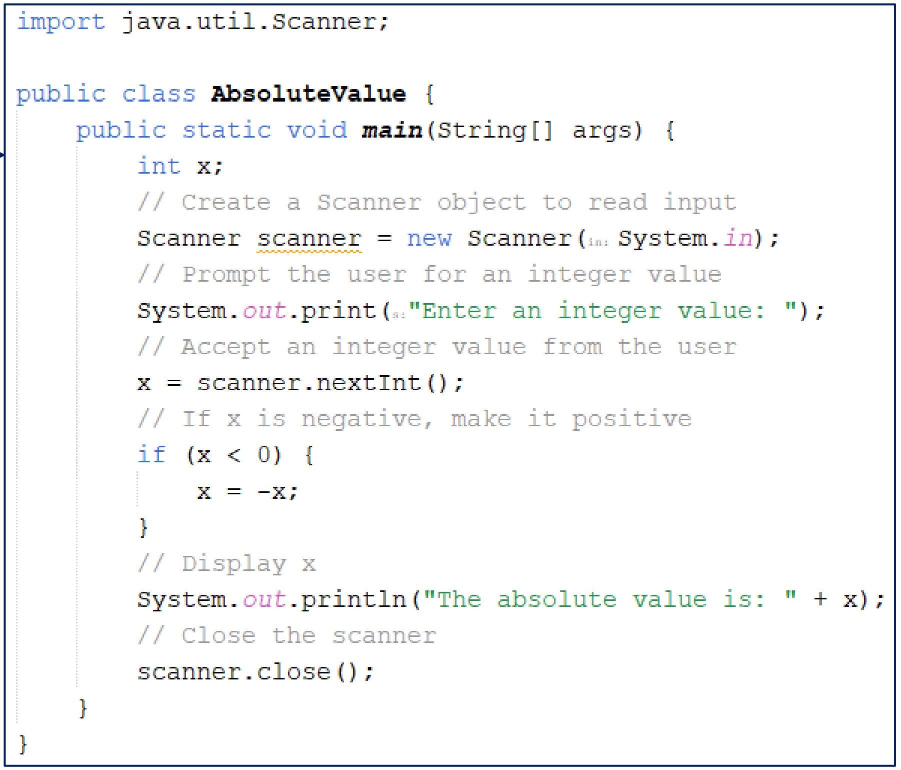
```cpp
#include <iostream>
using namespace std;

int main() {
    int x;
    // Prompt the user for an integer value
    cout << "Enter an integer value: ";
    // Accept an integer value from the user
    cin >> x;
    // If x is negative, make it positive
    if (x < 0) {
        x = -x;
    }
    // Display x
    cout << "The absolute value is: " << x << endl;
    return 0;
}
```



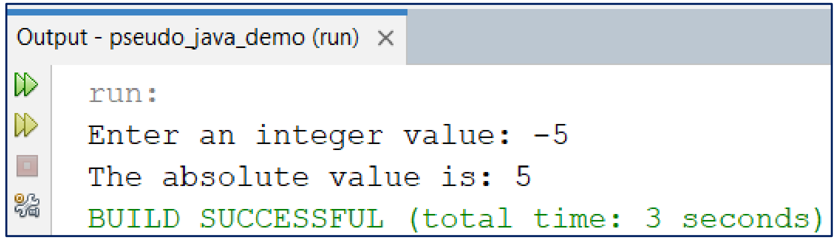
D:\MonHoc\PRF192\ThucHanh\pseudo_cpp_demo.exe

```
Enter an integer value: -5
The absolute value is: 5
```

Java



```java
import java.util.Scanner;

public class AbsoluteValue {
    public static void main(String[] args) {
        int x;
        // Create a Scanner object to read input
        Scanner scanner = new Scanner(System.in);
        // Prompt the user for an integer value
        System.out.print("Enter an integer value: ");
        // Accept an integer value from the user
        x = scanner.nextInt();
        // If x is negative, make it positive
        if (x < 0) {
            x = -x;
        }
        // Display x
        System.out.println("The absolute value is: " + x);
        // Close the scanner
        scanner.close();
    }
}
```

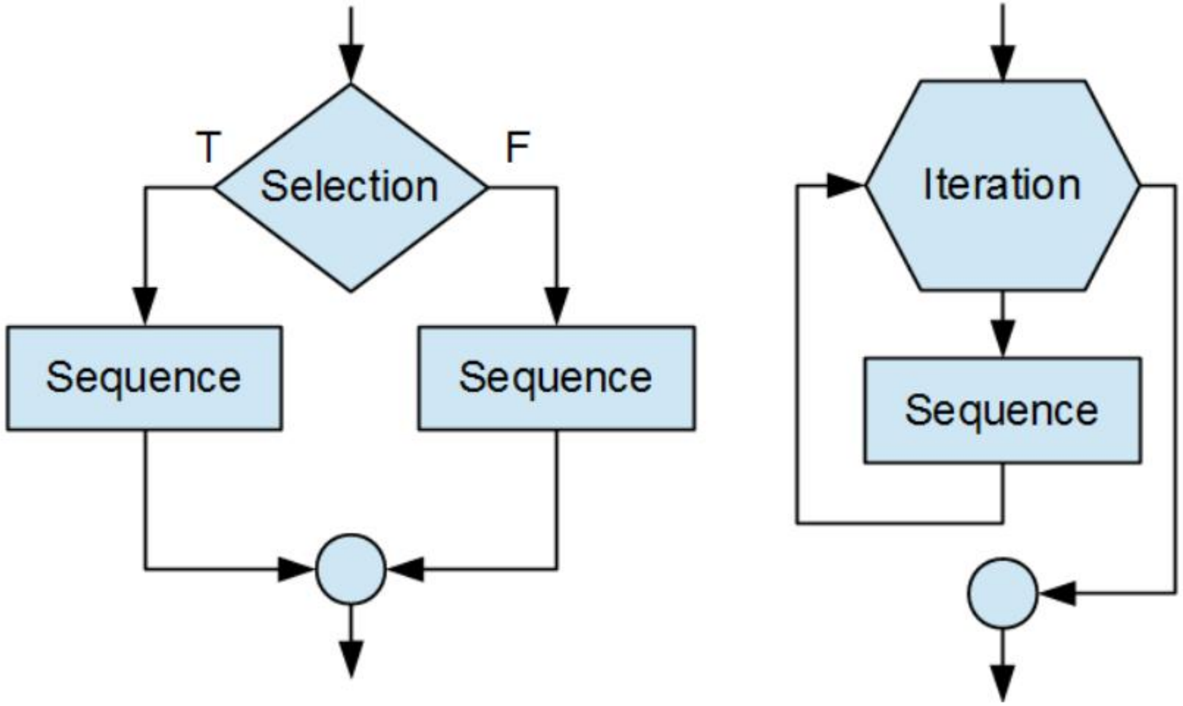


Output - pseudo_java_demo (run) ×

```
run:
Enter an integer value: -5
The absolute value is: 5
BUILD SUCCESSFUL (total time: 3 seconds)
```

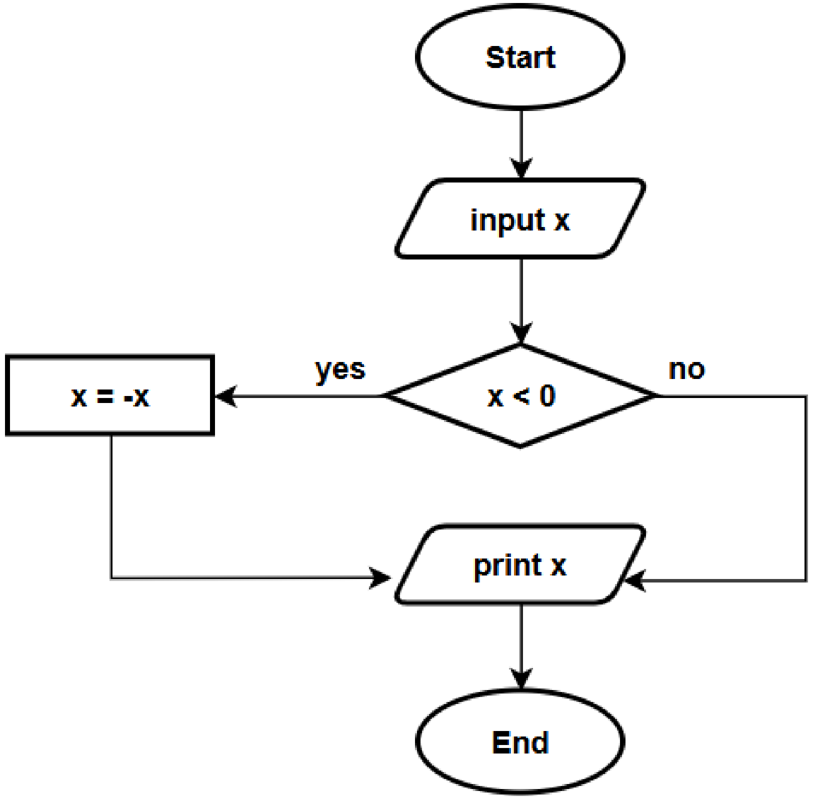
Structured Programming – flowcharting

- A ***flow chart*** is a set of conventional symbols connected by arrows that illustrate the flow of control through a programming solution. Popular sets of symbols for sequences, selections and iterations are shown below:

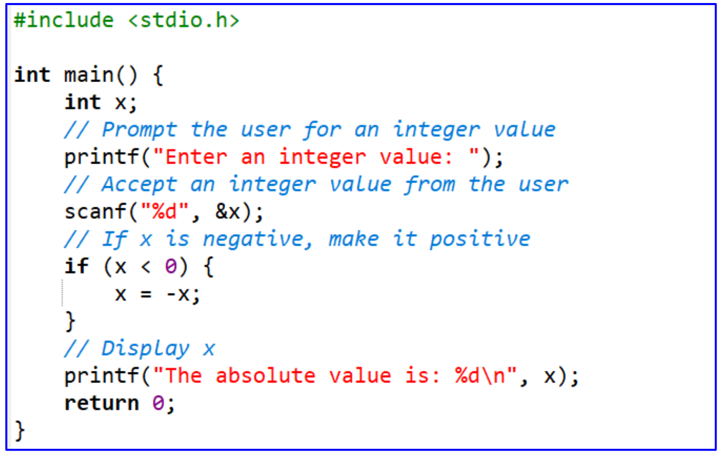


Flowchart - Example

- Calculating the absolute value of an integer inputted from the keyboard.

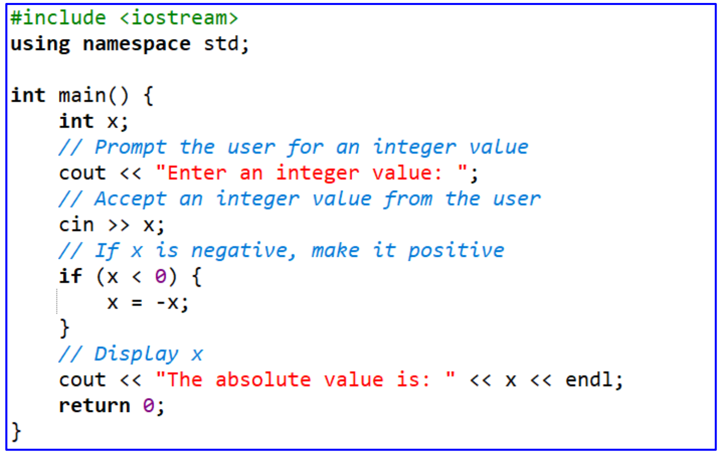


C

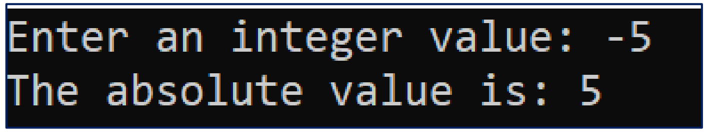


```c
#include <stdio.h>

int main() {
    int x;
    // Prompt the user for an integer value
    printf("Enter an integer value: ");
    // Accept an integer value from the user
    scanf("%d", &x);
    // If x is negative, make it positive
    if (x < 0) {
        x = -x;
    }
    // Display x
    printf("The absolute value is: %d\n", x);
    return 0;
}
```

C++



```cpp
#include <iostream>
using namespace std;

int main() {
    int x;
    // Prompt the user for an integer value
    cout << "Enter an integer value: ";
    // Accept an integer value from the user
    cin >> x;
    // If x is negative, make it positive
    if (x < 0) {
        x = -x;
    }
    // Display x
    cout << "The absolute value is: " << x << endl;
    return 0;
}
```

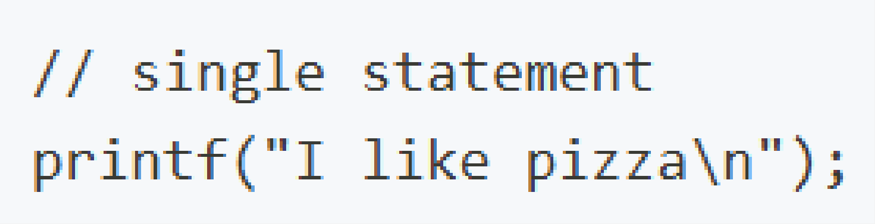


```
Enter an integer value: -5
The absolute value is: 5
```

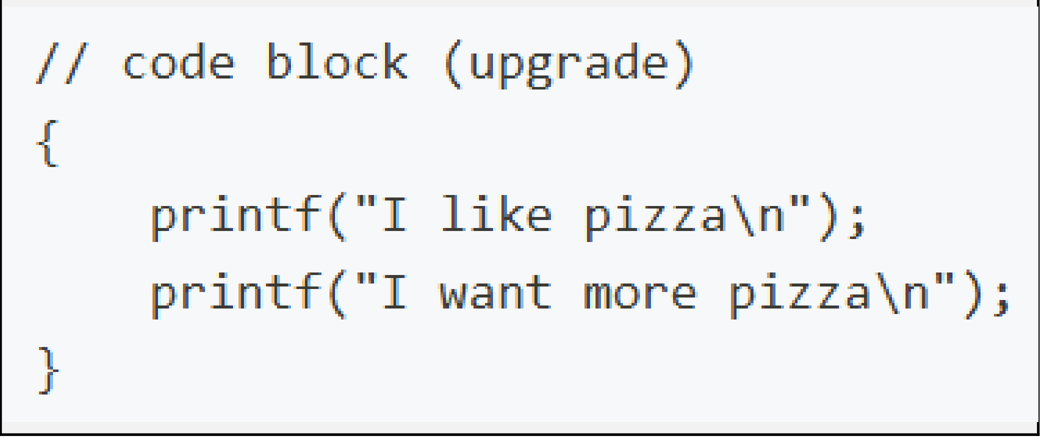
2. Sequence Contructs

- A **sequence** is either a simple *statement* or a *code block*.

Simple Statements	**Syntax:**	**Example:**

	Expression;	
		`// single statement`
		`printf("I like pizza\n");`

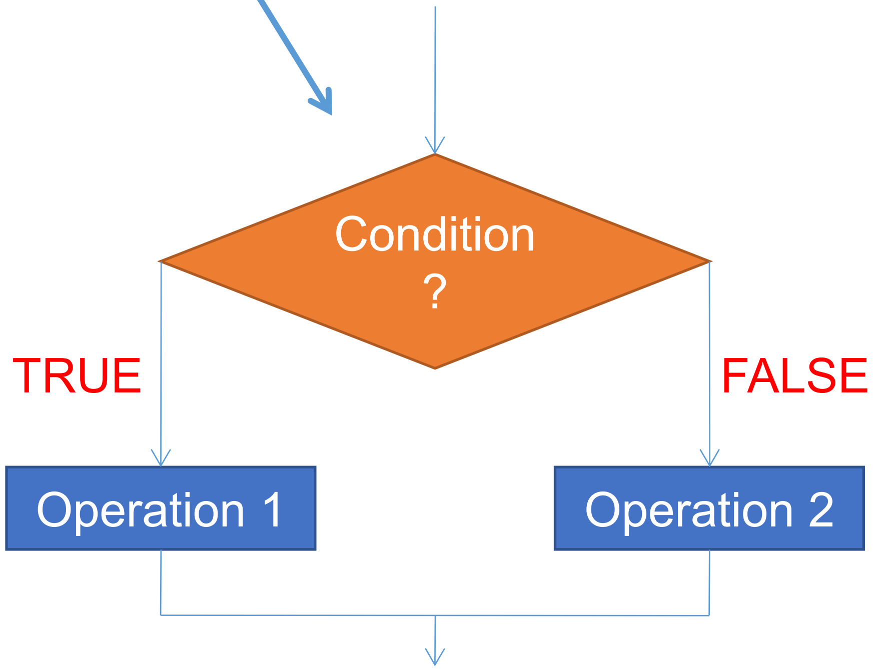
Code Block (A code block is a set of statements enclosed in curly braces)	**Syntax:**	**Example:**

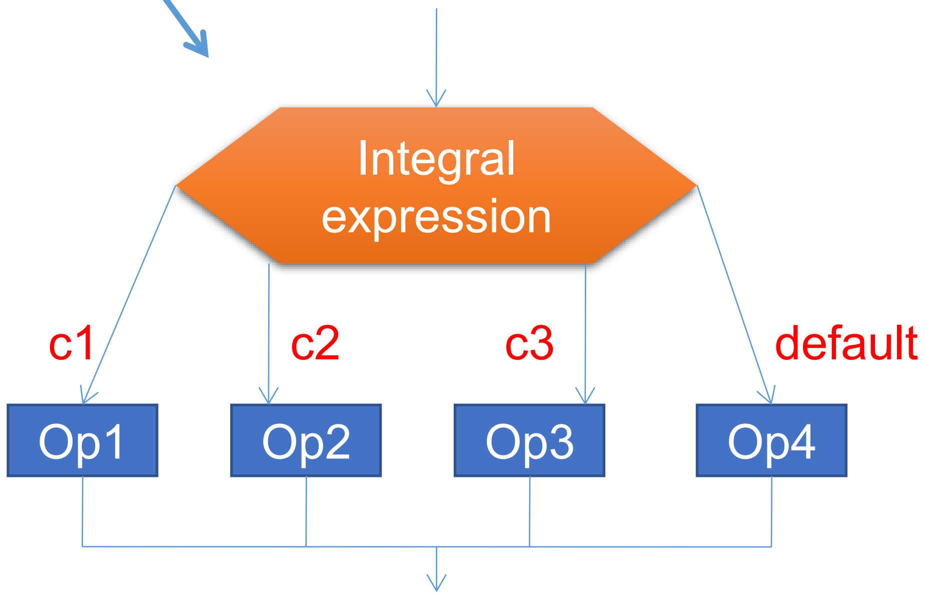
	{
	statement
	...
	statement
	}
		
		`// code block (upgrade)`
		`{`
		`    printf("I like pizza\n");`
		`    printf("I want more pizza\n");`
		`}`

3. Selection Constructs

Select 1/2	Select 1/n

if
if … else
If … else if …. else
? : (operator)
	switch



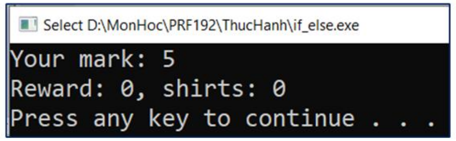


Selection Constructs - if … else

Syntax:

```
if  (condition)
{
      statements
}
```

```c
#include <stdio.h>
#include <stdlib.h>

int main(){
    int mark;
    int reward=0;
    int noOfShirts=0;
    printf("Your mark: ");
    scanf("%d", &mark);

    if(mark>7){
        reward = 500000;
        noOfShirts = 2;
    }
    printf("Reward: %d, shirts: %d\n", reward, noOfShirts);

    system ("pause");
    return 0;
}
```



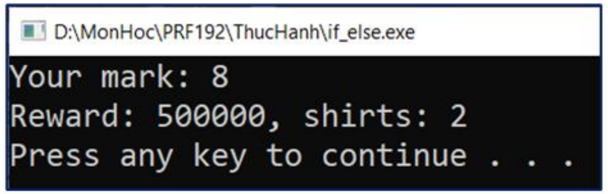
Select D:\MonHoc\PRF192\ThucHanh\if_else.exe

```
Your mark: 5
Reward: 0, shirts: 0
Press any key to continue . . .
```

Syntax:

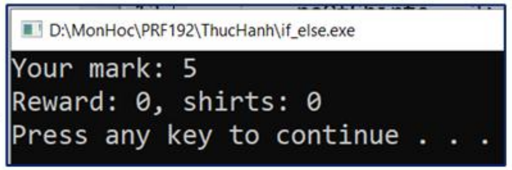
```
if  (condition)
{
      statements
}
else
{
      statements
}
```

```c
#include <stdio.h>
#include <stdlib.h>

int main(){
    int mark;
    int reward=0;
    int noOfShirts=0;
    printf("Your mark: ");
    scanf("%d", &mark);

    if(mark>7){
        reward = 500000;
        noOfShirts = 2;
    }else{
        reward = 0;
        noOfShirts = 0;
    }
    printf("Reward: %d, shirts: %d\n", reward, noOfShirts);

    system ("pause");
    return 0;
}
```



D:\MonHoc\PRF192\ThucHanh\if_else.exe

```
Your mark: 8
Reward: 500000, shirts: 2
Press any key to continue . . .
```



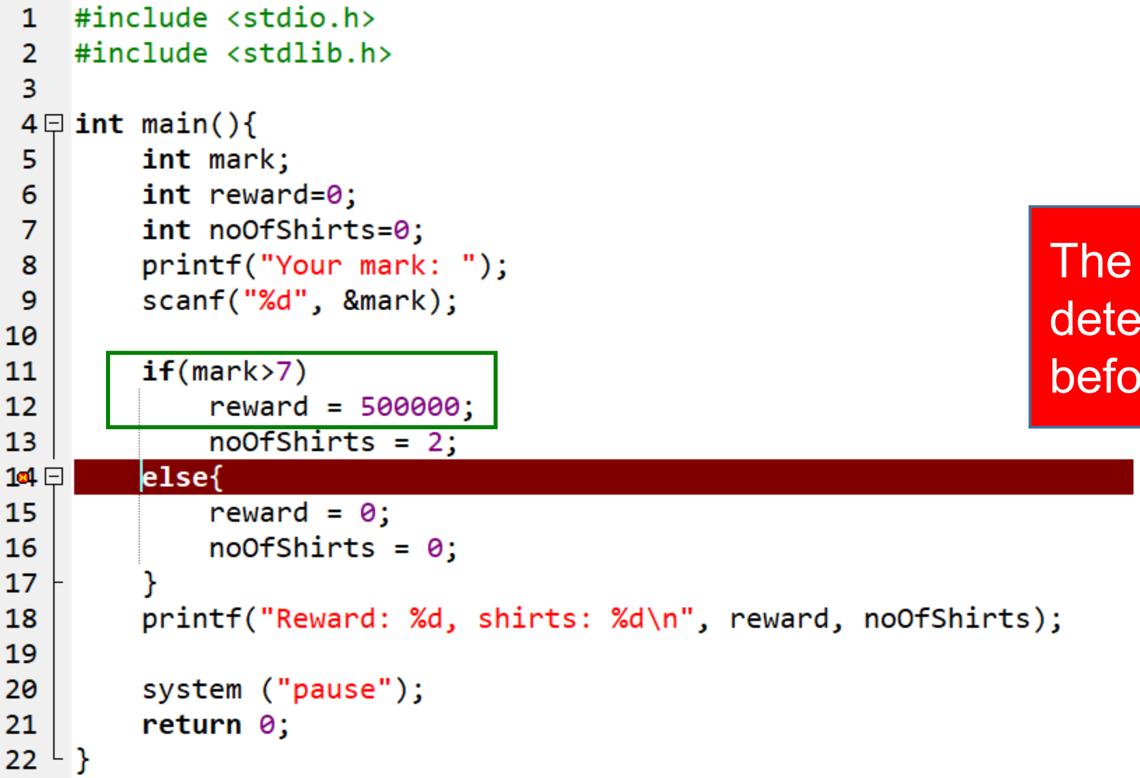
D:\MonHoc\PRF192\ThucHanh\if_else.exe

```
Your mark: 5
Reward: 0, shirts: 0
Press any key to continue . . .
```

Selection Constructs - if … else (cont.)

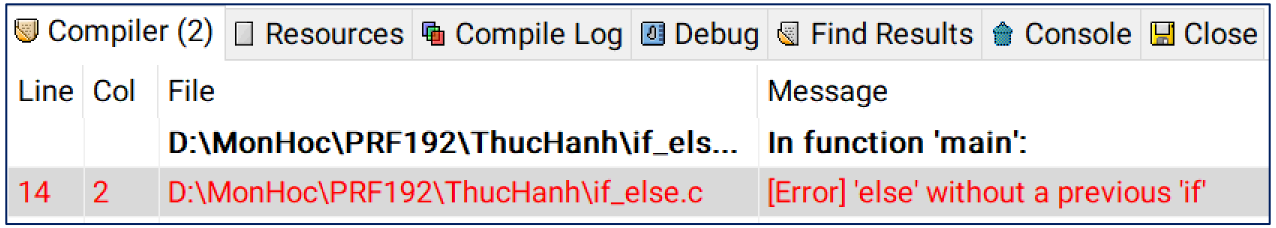
Syntax:

```
if  (condition)
{
     statements
}
else
{
     statements
}
```



```c
#include <stdio.h>
#include <stdlib.h>

int main(){
	int mark;
	int reward=0;
	int noOfShirts=0;
	printf("Your mark: ");
	scanf("%d", &mark);

	if(mark>7)
		reward = 500000;
		noOfShirts = 2;
	else{
		reward = 0;
		noOfShirts = 0;
	}
	printf("Reward: %d, shirts: %d\n", reward, noOfShirts);

	system ("pause");
	return 0;
}
```

The compiler can not determine the if statement before the else statement.



Line	Col	File	Message
		D:\MonHoc\PRF192\ThucHanh\if_els...	**In function 'main':**
14	2	D:\MonHoc\PRF192\ThucHanh\if_else.c	[Error] 'else' without a previous 'if'

Selection Constructs - if … else (cont.)

Syntax:

```
if  (condition 1)
{
        statements
}
else if  (condition2)
        {
                statements
        }
        else
         {
                statements
         }
```

Nested if

Example 1:

- Buying N T-shirts with promotion:
- N<=3:   120000$/item
- From 4th to 6th: 90000$/item
- From 7th to 10th: 85000$/item
- From 11th :   70000$/item
- Describe the expression that will compute the paid value.

```
Begin
N,t → int
Accept N
Compute t
Print out  t
End
```

→

Let N: number of T-shirts bought, t: money must be paid.

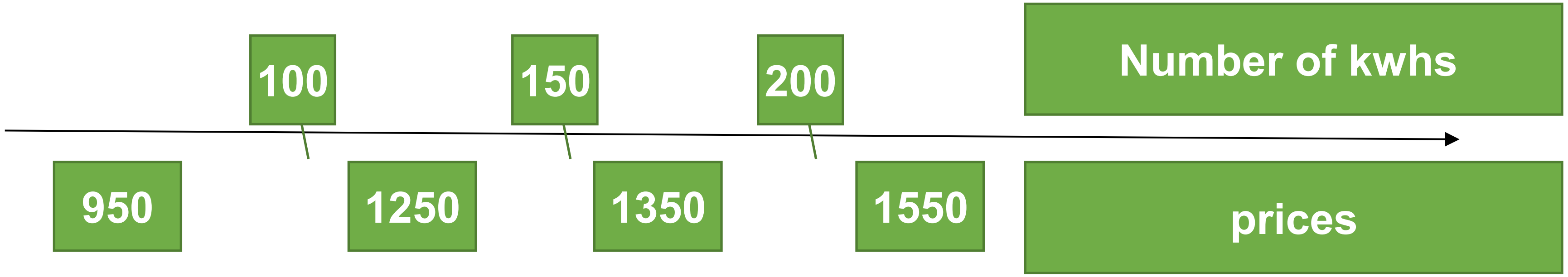
```
if  (N <=3)  t =  N*120000 ;
else if  (N<=6)  t= 3*120000 + (N-3) * 90000;
else if  (N<=10)
        t= 3*120000 + 3*90000 + (N-6)*85000;
else
        t= 3*120000 + 3*90000 + 4*85000 + (N-10)*70000;
```

Selection Constructs - if … else (cont.)

Practice 1:

- Similarly, describe the expression that will compute the paid value when we use electric power.
- Implement it to a program.

```
Begin
N,t → int
Accept N
Compute t
Print out  t
End
```



100

150

200

Number of kwhs

950

1250

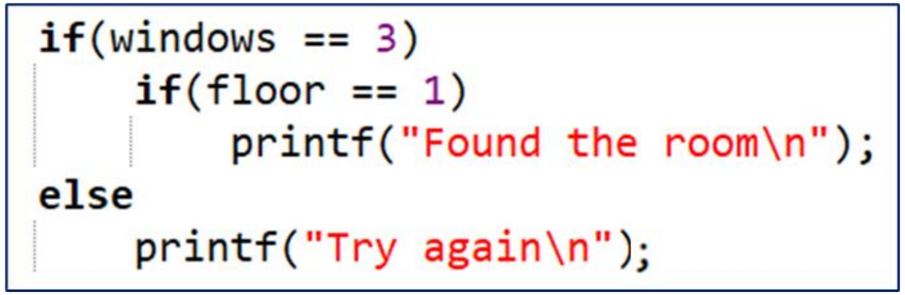
1350

1550

prices

Selection Constructs - Dangling Else

Ambiguity may arise in the case of nested if else constructs

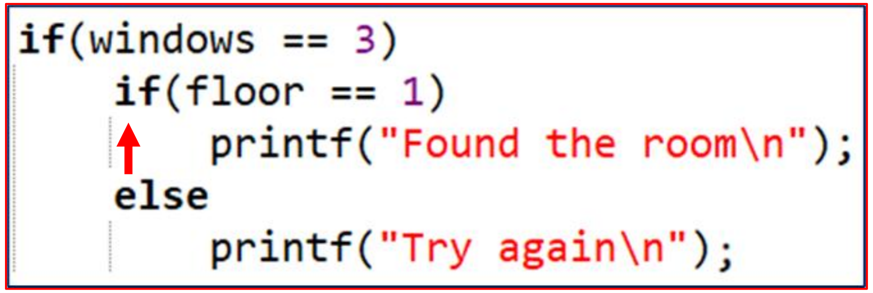


```
if(windows == 3)
    if(floor == 1)
        printf("Found the room\n");
else
    printf("Try again\n");
```

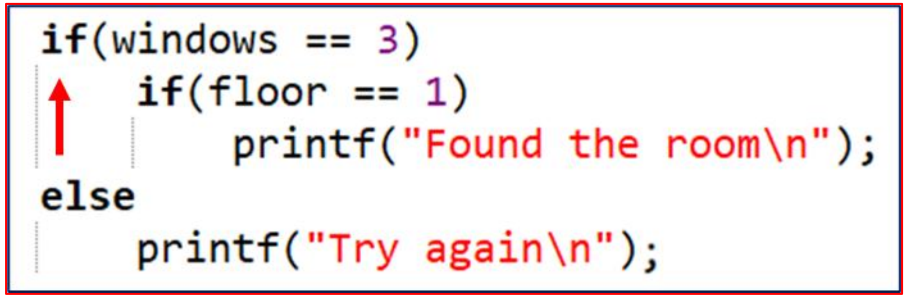
To which if does the else belong?

Which interpretation?

or



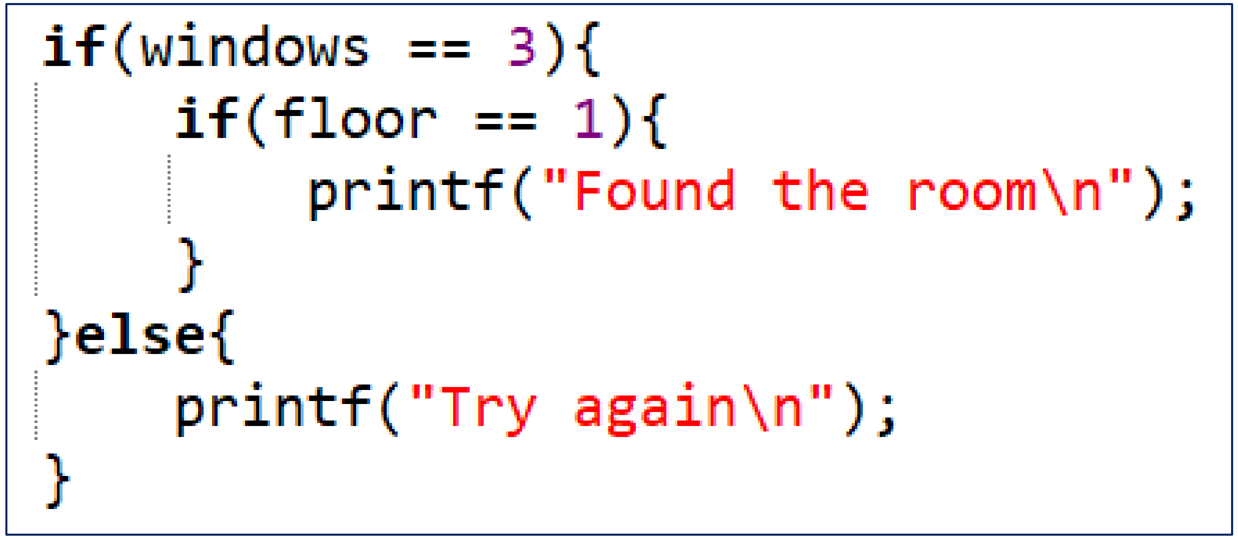
```
if(windows == 3)
    if(floor == 1)
        printf("Found the room\n");
    else
        printf("Try again\n");
```



```
if(windows == 3)
    if(floor == 1)
        printf("Found the room\n");
else
    printf("Try again\n");
```

Selection Constructs - Dangling Else (cont.)

- The rule in C is that an else always belong to the innermost if avariable. Use { } to explicitly determine statements.
- Solution:

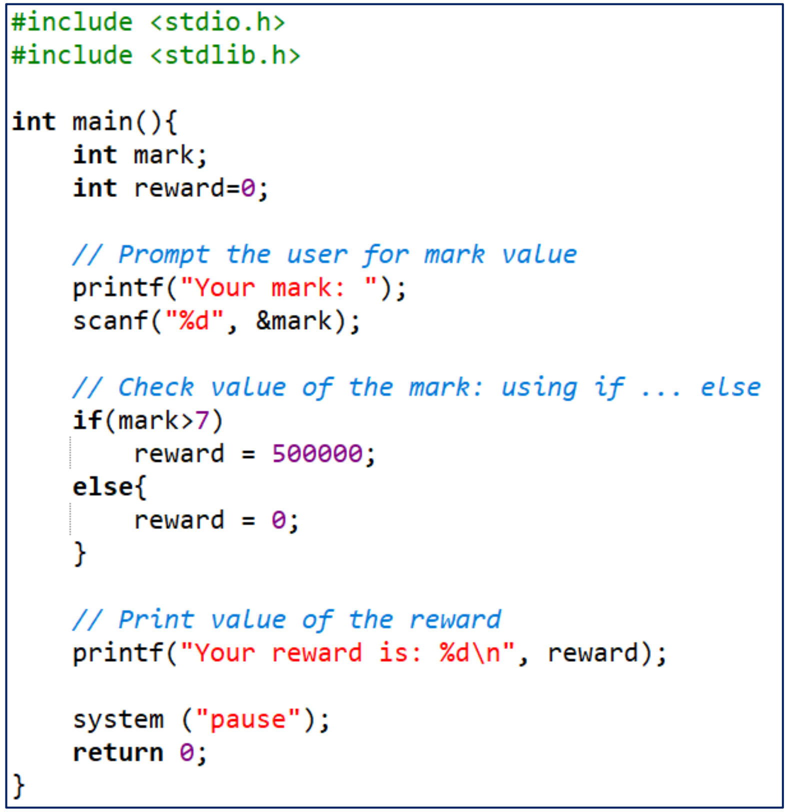


```
if(windows == 3){
    if(floor == 1){
        printf("Found the room\n");
    }
}else{
    printf("Try again\n");
}
```

Selection Constructs - Operator ? :

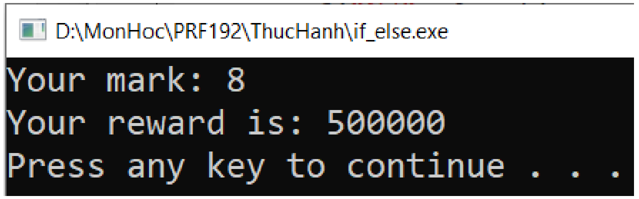
Syntax: (condition) ? True_Value : False_Value

Using if … else



```c
#include <stdio.h>
#include <stdlib.h>

int main(){
    int mark;
    int reward=0;

    // Prompt the user for mark value
    printf("Your mark: ");
    scanf("%d", &mark);

    // Check value of the mark: using if ... else
    if(mark>7)
        reward = 500000;
    else{
        reward = 0;
    }

    // Print value of the reward
    printf("Your reward is: %d\n", reward);

    system ("pause");
    return 0;
}
```

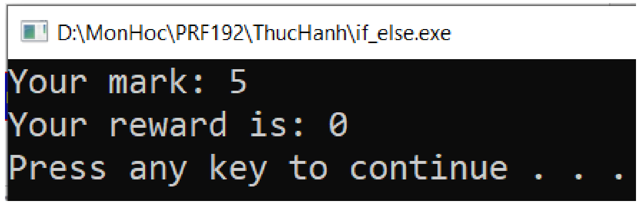
Case 1: mark > 7



D:\MonHoc\PRF192\ThucHanh\if_else.exe

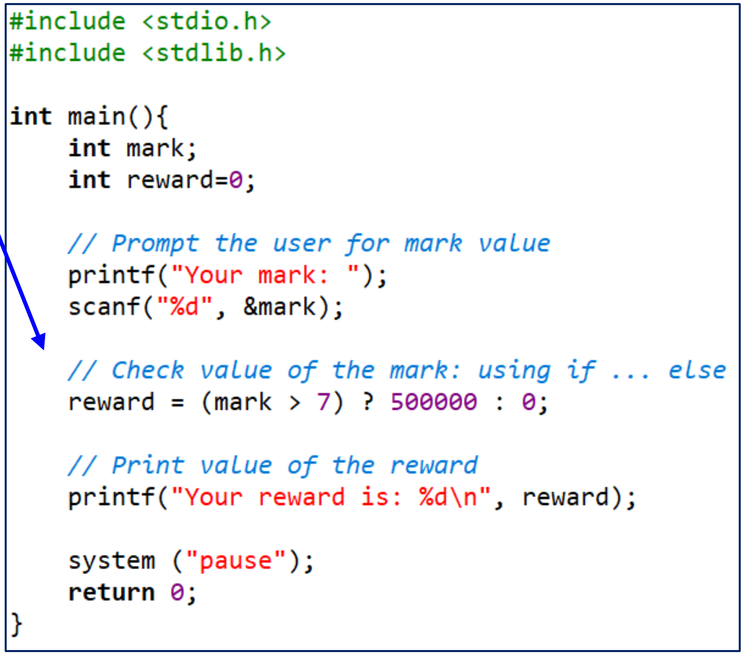
```
Your mark: 8
Your reward is: 500000
Press any key to continue . . .
```

Case 2: mark < 7



D:\MonHoc\PRF192\ThucHanh\if_else.exe

```
Your mark: 5
Your reward is: 0
Press any key to continue . . .
```

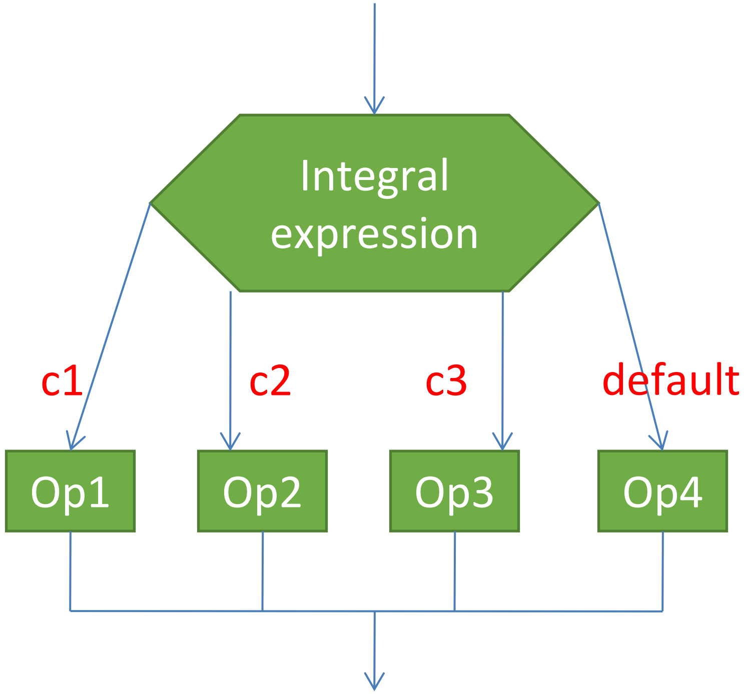


```c
#include <stdio.h>
#include <stdlib.h>

int main(){
    int mark;
    int reward=0;

    // Prompt the user for mark value
    printf("Your mark: ");
    scanf("%d", &mark);

    // Check value of the mark: using if ... else
    reward = (mark > 7) ? 500000 : 0;

    // Print value of the reward
    printf("Your reward is: %d\n", reward);

    system ("pause");
    return 0;
}
```

Selection Constructs - The switch statement

```
switch (variable or expression)
{
    case constant :
        statement(s);
        break;
    case constant :
        statement (s);
        break;
    default:
        statement (s);
}
```

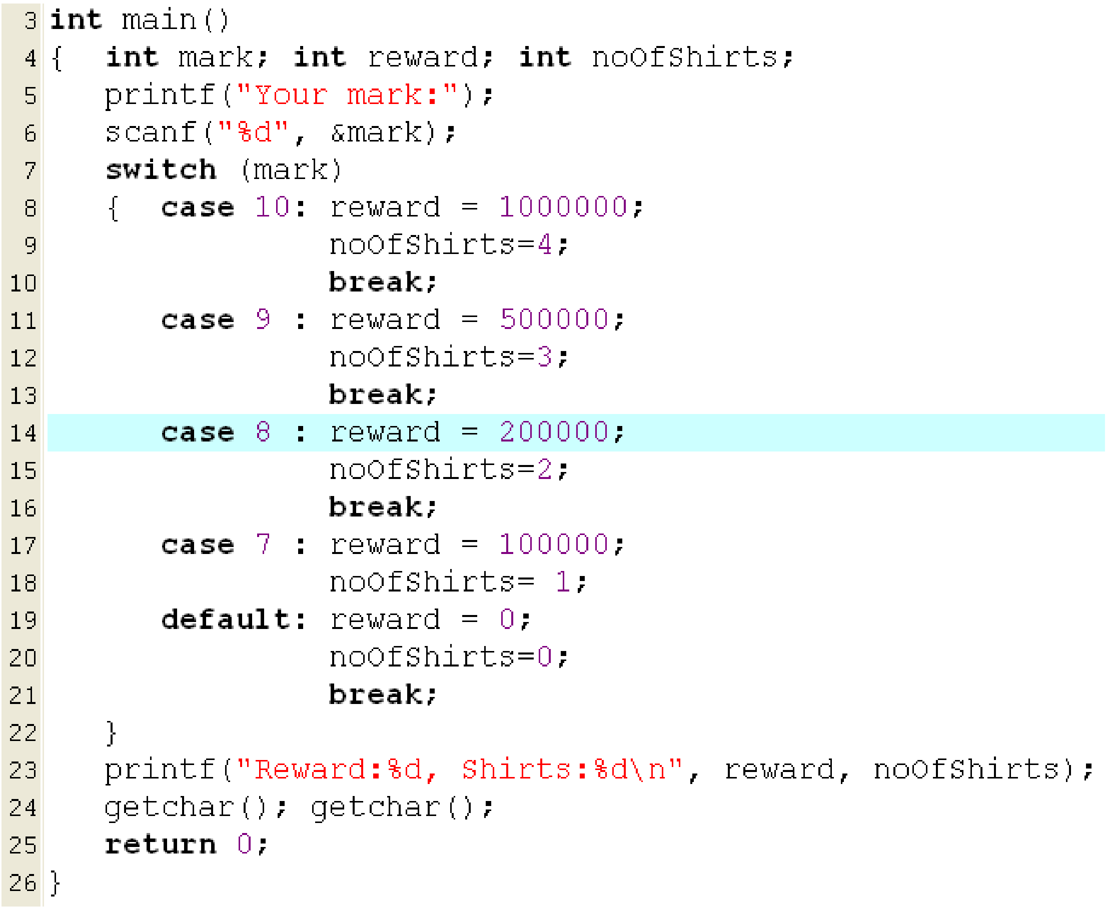
char / int



If the break statement is missed, the next statements are executed until a break is detected or all statements in the body of the switch are executed. Each case is an entry of a selection

Selection Constructs - The switch statement (cont.)

- Example:



```
int main()
{  int mark; int reward; int noOfShirts;
   printf("Your mark:");
   scanf("%d", &mark);
   switch (mark)
   {  case 10: reward = 1000000;
               noOfShirts=4;
               break;
      case 9 : reward = 500000;
               noOfShirts=3;
               break;
      case 8 : reward = 200000;
               noOfShirts=2;
               break;
      case 7 : reward = 100000;
               noOfShirts= 1;
      default: reward = 0;
               noOfShirts=0;
               break;
   }
   printf("Reward:%d, Shirts:%d\n", reward, noOfShirts);
   getchar(); getchar();
   return 0;
}
```

If input is 8, what are outputs?

a) **200000 , 2**
b) **300000, 3**
c) **0, 0**
d) **1000000, 4**
e) **1500000, 10**
f) **None of the others**

If input is 7, what are outputs?

The switch statement (cont.)

Practice 2:

Write a program that allows user inputting a simple expression containing one of four operators +, -, *, / then the result is printed out to the monitor. Input format:

num1 operator num2,

Example: 4*5

Implement it.

Analysis

Nouns: expression num1 op num2

```
          → double num1, num2; char op
          result → double result
Verbs:  Begin
          Accept  num1, op, num2  →  "%lf%c%lf"
          switch (op)
          {   case '+' : result = num1 + num2;
                         print out result;
                         break;
              case '-' : result = num1 - num2;
                         print out result;
                         break;
              case '*' : result = num1 * num2;
                         print out result;
                         break;
              case '/' : if ( num2==0)
                            print out "Divide by 0 "
                         else
                         { result = num1 / num2;
                             print out result;
                         }
                         break;
              default: print out "Op is not supported"
           }
          End
```

4. Iteration (loop) Constructs

- **Loop/Iteration**: some statements are executed repetitively
- **Structure of a loop**:
  - Initial block.
  - Condition.
  - Task in each execution.
- **Types of iteration**: fixed loops, variable loops
- The iteration constructs are:

  while

  do while

  for

Iteration Constructs (cont.)

Identify a loop:

- Calculate S= 1 + 2 + 3 + 4 + 5 + 6 + 7 + … + 100

  ➔ Some addition are performed ➔ Loop

- Sum of some numbers they are inputted until user enters 0.

  ➔ Accept and add numbers ➔ Loop

Iteration Constructs (cont.)

Identify a loop for an expression:

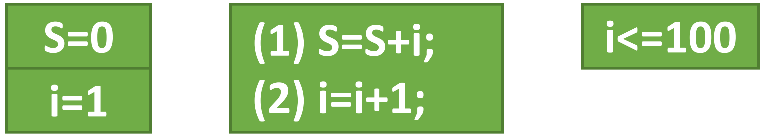
Left side → Initial block

Right side → Condition

An operation and preparations for the next iteration: Tasks in each iteration.

S=1+2+3+4+… + 100

→ S=0+1+2+3+4+… + 100



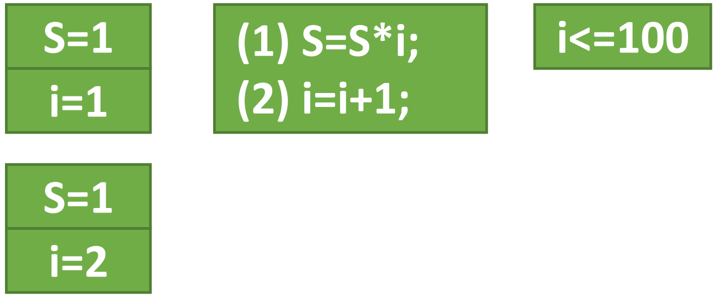
S=0
i=1

(1) S=S+i;
(2) i=i+1;

i<=100

S=1*2*3*4*… * 100

→S=1*1*2*3*4*… * 100



S=1
i=1

(1) S=S*i;
(2) i=i+1;

i<=100

S=1
i=2

Iteration Constructs (cont.)

$$S=\begin{cases} 0 \ , n<=0 \\ 1+3+5+\ldots+n, \text{ n is odd} \\ 2+4+6+\ldots+n, \text{ n is even} \end{cases}$$

Initial value:
S = 0
i = (n%2==1)? 1: 2;
Condition i <=n
Tasks: S += i;
i+=2;

$$S=\begin{cases} 0 \ , n<=0 \\ n+(n-2)+(n-4)+\ldots+0 \end{cases}$$

Initial value:
S = 0;
i = n;
Condition i>0
Tasks: S += i;
i -=2;

$$S= 1 + 1/1^0 + 1/2^1 + 1/3^2 + \ldots + 1/n^{n-1}$$

Initial value:
S = 1.0;
i = 1;
Condition i <=n
Tasks: S += 1.0/ pow (i, i-1);
i = i +1;

math.h

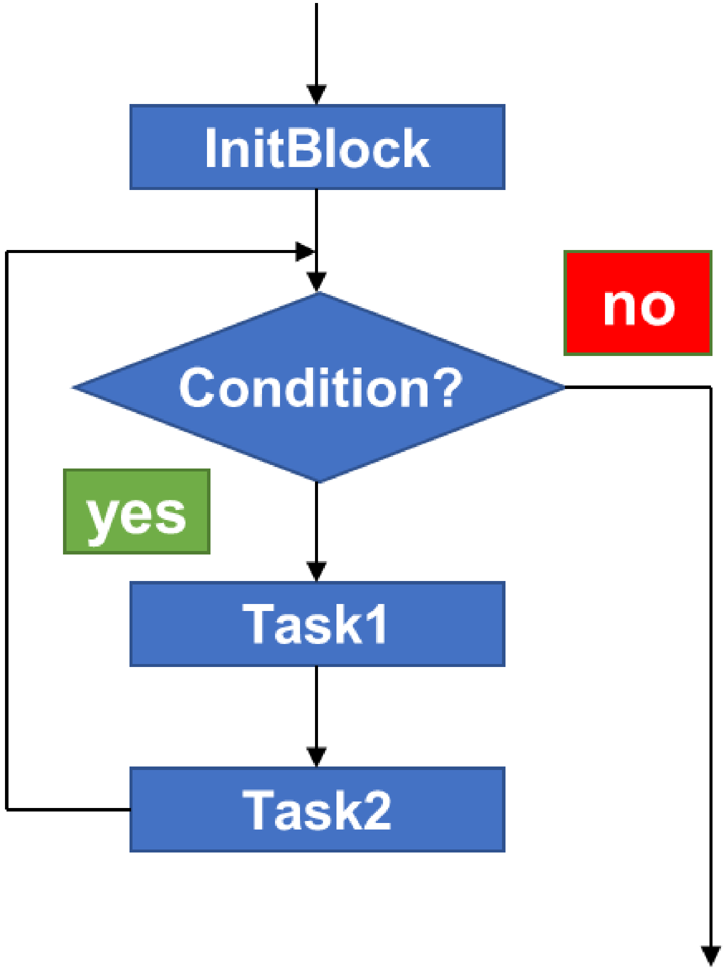
Iteration - for statement

- for (InitBlock; Condition ; Task2) Task1;
- for (Init1, Init2; Condition ; Task1, Task2);
- InitBlock;

```
for ( ; Condition ; Task2) Task1;
```

- InitBlock;

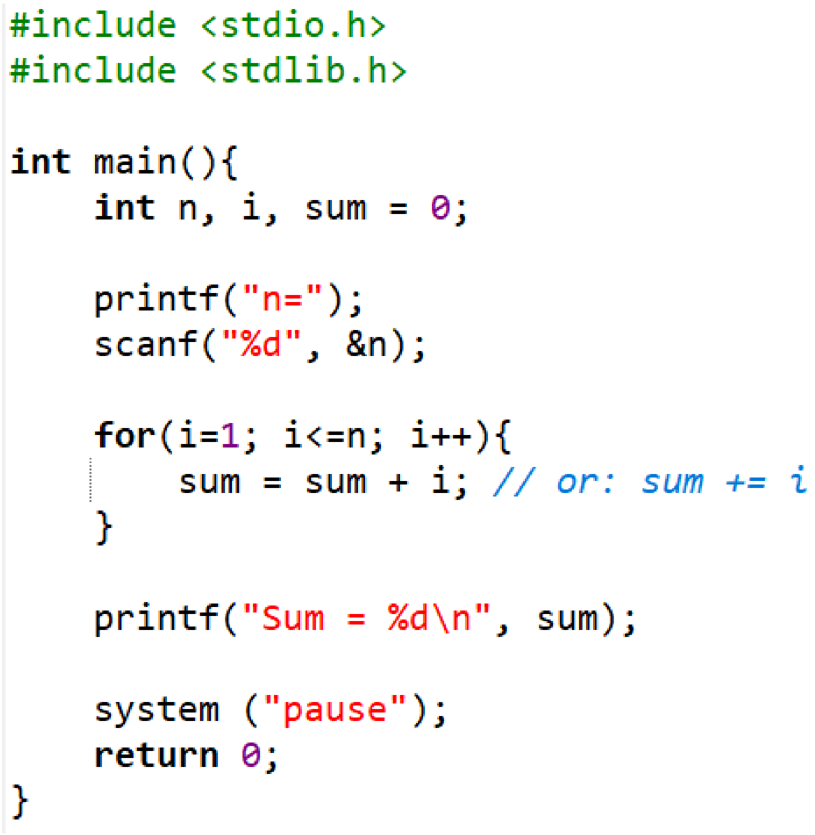
```
for ( ; Condition ;)
{
  Task1;
  Task2;
}
```



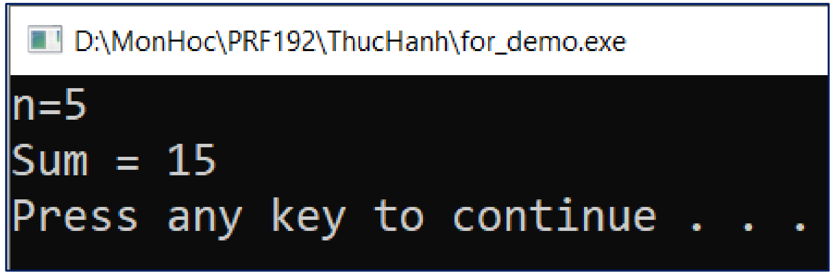
Iteration - for statement (cont.)

- **Example**: Write a program that will calculate 1+2+3+…+n.

```
Accepted variable: int n
Sum 1 .. N → int sum
Algorithm
Accept n
Loop:
   Initialize i=1, sum=0
   Condition i<=n
   Tasks:  sum += i; i++;
Print out sum
```



```c
#include <stdio.h>
#include <stdlib.h>

int main(){
    int n, i, sum = 0;

    printf("n=");
    scanf("%d", &n);

    for(i=1; i<=n; i++){
        sum = sum + i; // or: sum += i
    }

    printf("Sum = %d\n", sum);

    system ("pause");
    return 0;
}
```



```
n=5
Sum = 15
Press any key to continue . . .
```

Requirement: Students practice other structures of for and give feedback.

Iteration - for statement

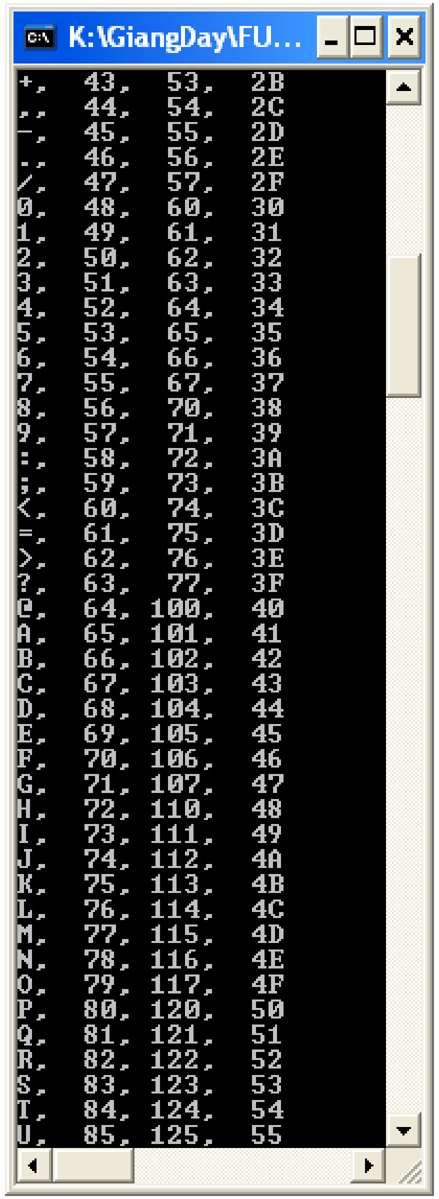
Practice 3:

- Write a program that will print out the ASCII table.

Analysis:

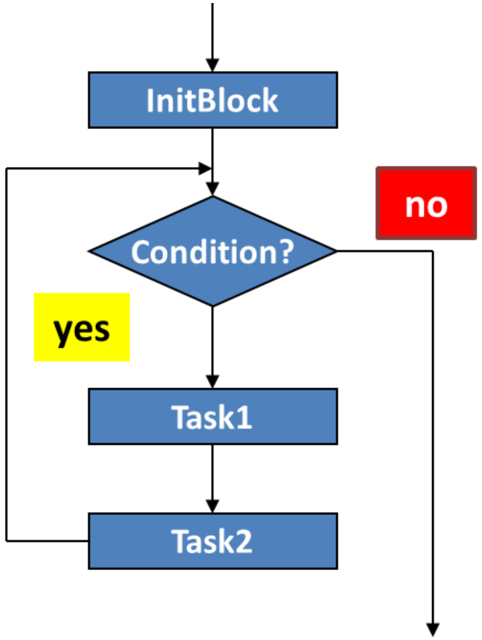
ASCII code : 0 → 255
Initialize: int code =0
Condition: code <256
Task:
  Print the code using 4 format: %c, %d, %o, %X
  code = code +1

```
for (code= 0; code <256; code++)
  printf ("%c, %3d, %3o, %3X\n", code, code, code, code);
```



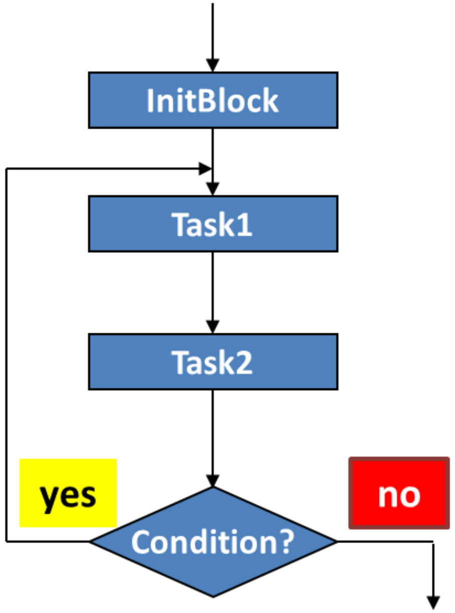
```
+,  43,  53,  2B
,,  44,  54,  2C
-,  45,  55,  2D
.,  46,  56,  2E
/,  47,  57,  2F
0,  48,  60,  30
1,  49,  61,  31
2,  50,  62,  32
3,  51,  63,  33
4,  52,  64,  34
5,  53,  65,  35
6,  54,  66,  36
7,  55,  67,  37
8,  56,  70,  38
9,  57,  71,  39
:,  58,  72,  3A
;,  59,  73,  3B
<,  60,  74,  3C
=,  61,  75,  3D
>,  62,  76,  3E
?,  63,  77,  3F
@,  64, 100,  40
A,  65, 101,  41
B,  66, 102,  42
C,  67, 103,  43
D,  68, 104,  44
E,  69, 105,  45
F,  70, 106,  46
G,  71, 107,  47
H,  72, 110,  48
I,  73, 111,  49
J,  74, 112,  4A
K,  75, 113,  4B
L,  76, 114,  4C
M,  77, 115,  4D
N,  78, 116,  4E
O,  79, 117,  4F
P,  80, 120,  50
Q,  81, 121,  51
R,  82, 122,  52
S,  83, 123,  53
T,  84, 124,  54
U,  85, 125,  55
```

Iteration - while/ do … while statements



```
/* Initializations  */
while (condition)
{   statements;
}
```

The condition will be tested before tasks are performed.



```
/* Initializations */
do
{   statements;
}
while (condition) ;
```

The condition will be tested after tasks are performed.

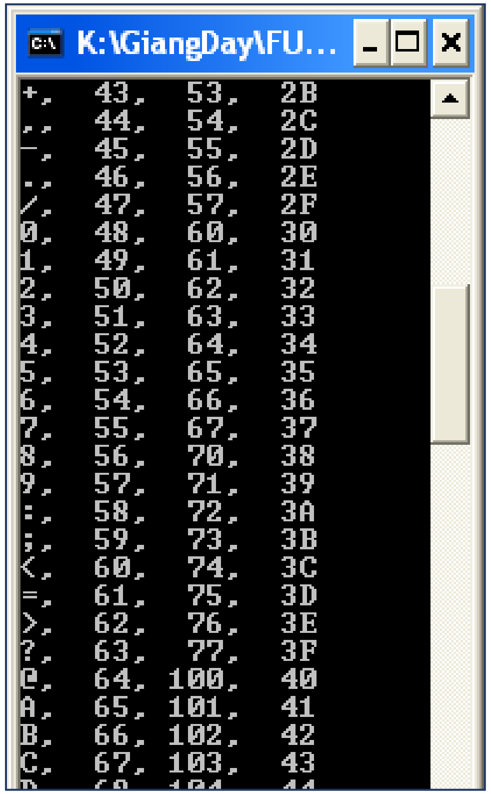
Iteration - while/ do … while statements

Practice 4:

- Write a program that will print out the ASCII table.

Analysis:

ASCII code : 0 → 255
Initialize: int code =0
Condition: code <256
Task:
  Print the code using 4 format: %c, %d, %o, %X
  code = code +1



```
+,  43,  53,  2B
,,  44,  54,  2C
-,  45,  55,  2D
.,  46,  56,  2E
/,  47,  57,  2F
0,  48,  60,  30
1,  49,  61,  31
2,  50,  62,  32
3,  51,  63,  33
4,  52,  64,  34
5,  53,  65,  35
6,  54,  66,  36
7,  55,  67,  37
8,  56,  70,  38
9,  57,  71,  39
:,  58,  72,  3A
;,  59,  73,  3B
<,  60,  74,  3C
=,  61,  75,  3D
>,  62,  76,  3E
?,  63,  77,  3F
@,  64, 100,  40
A,  65, 101,  41
B,  66, 102,  42
C,  67, 103,  43
```

```
int code=0;
while (code<256)
{ printf ("%c, %3d, %3o, %3X\n", code, code, code, code);
  code++;
}
```

```
int code=0;
do
{ printf ("%c, %3d, %3o, %3X\n", code, code, code, code);
  code++;
}
while (code <256);
```

Iteration - Exercise 1

- Write a program that will print out the sum of integers inputted by user. The input will terminate if user enters the value 0.

```
Nouns:  inputted integer → int x,
            sum of integers → int sum
Tasks (algorithm)
Begin
      sum =0
      do
      {  accept x;
         sum += x;
      }
      while (x!=0);
      Print out sum
End
```

Implement it by yourself.

Iteration - Exercise 2

Implement it by yourself.

- Write a program that permits user entering some characters until the ENTER key (code 10) is pressed. The program will print out the number of digits, number of letters, number of other keys were pressed. Accept a character: c=getchar();

Nouns:

character inputted → char c, Number of digits → int noDigits
Number of letters → noLetters, Number of other keys → noOthers
#define ENTER 10

Algorithm

```
Begin
noDigits = noLetters = noOthers =  c= 0
printf("Enter a string:");
While (c!=ENTER)
{  accept c;
   if ( c>='0' && c <='9') noDigits++;
   else if ( (c>='a' && c <='z') || (c>='A' && c <='Z') ) noLetters++;
   else noOthers++;
}
Print out noDigits, noLetters, noOthers
End
```

The while statement is intentionally used. So, c=0 is assigned and the condition c!=ENTER is evaluated to TRUE

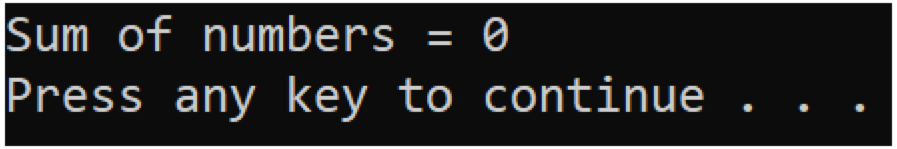
Input form:
abc1234fGH+-*/?(ENTER)

Iteration - break/ bypass a loop

Example: break

```
#include <stdio.h>
#include <stdlib.h>

int main(){
    int i, sumNumbers = 0;

    // Using continue keyword in a loop
    for(i=0; i<5; i++){
        if(i%2==1){
            break;
        } else{
            sumNumbers += i;
        }
    }

    // Print value of sumNumbers
    printf("Sum of numbers = %d\n", sumNumbers);
    system("pause");
    return 0;
}
```

0 1 2 3 4 5

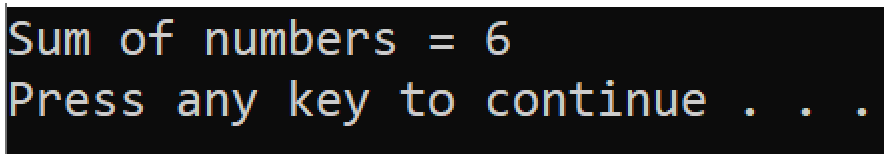


```
Sum of numbers = 0
Press any key to continue . . .
```

Example: continue

```
#include <stdio.h>
#include <stdlib.h>

int main(){
    int i, sumNumbers = 0;

    // Using continue keyword in a loop
    for(i=0; i<5; i++){
        if(i%2==1){
            continue;
        }
        sumNumbers += i;
    }

    // Print value of sumNumbers
    printf("Sum of numbers = %d\n", sumNumbers);
    system("pause");
    return 0;
}
```

0 1 2 3 4 5



```
Sum of numbers = 6
Press any key to continue . . .
```

Iteration Constructs: Flags

- The one entry, one exit principle is fundamental to structured programming.
- C includes three keywords that allow jumps across statements: **goto**, **continue**, and **break**. Using any of these keywords, except for **break** in a **switch** construct, violates the one entry, one exit principle of structured programming.

Iteration Constructs: Flags (cont.)

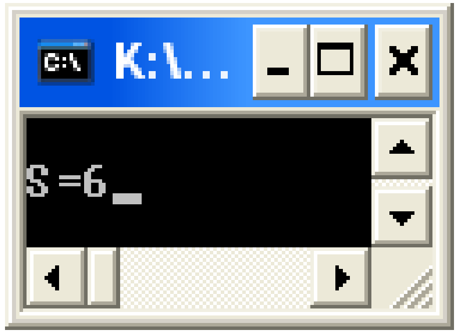
- To improve readability, programmers advocated:
    - The use of whitespace to identify the logical structure of the code
    - The abolition of all goto statements
    - The abolition of all continue statements
    - The abolition of all break statements, except with switch
- A technique for avoiding jumps is called flagging. A **flag** is a variable that keeps track of a true or false state.

Iteration Constructs: Flags (cont.)

Use *if* instead of *continue*

- Example:

```
#include <stdio.h>
int main()
{   int S=0;
    int i;
    /* Remove continue through using if */
    for (i=0;i<5; i++)
      {   if (i%2==1) continue;
          S+=i;
      }
    printf ("\nS=%d", S);
    getchar();
    return 0;
}
```

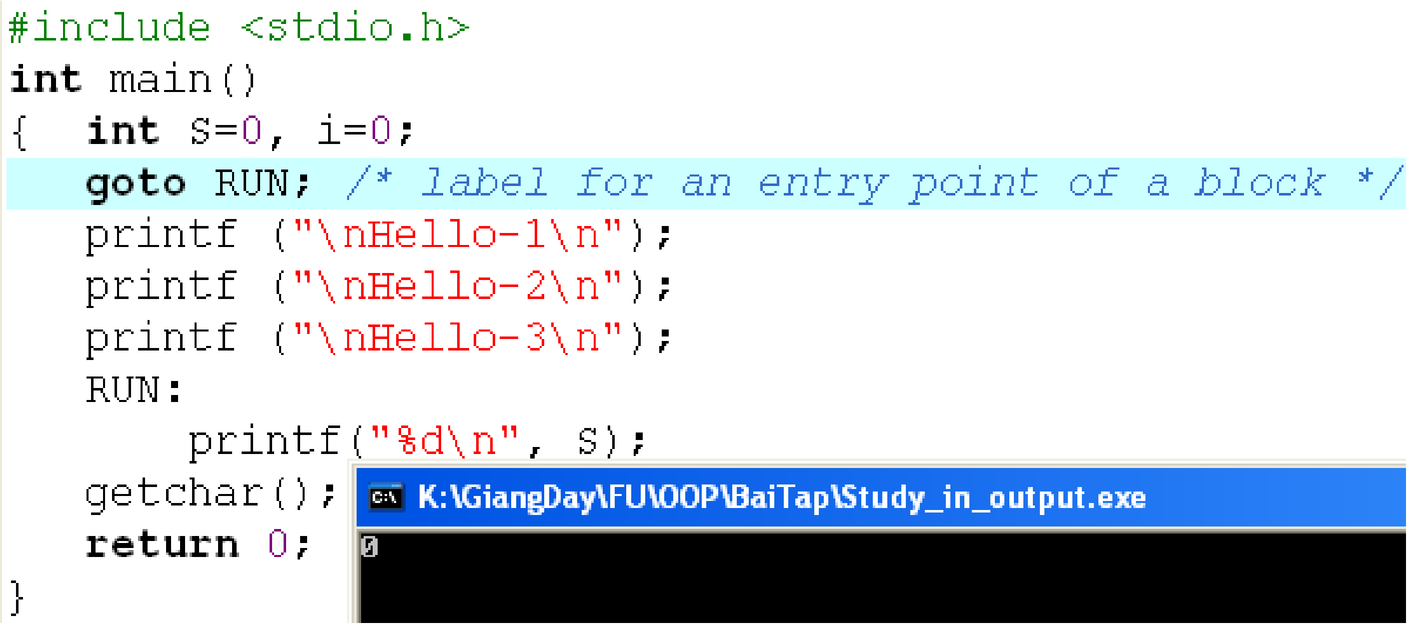


```
S=6_
```

```
#include <stdio.h>
int main()
{   int S=0;
    int i;
    /* Remove continue through using if */
    for (i=0;i<5; i++)
      {   if (i%2==0) S+=i;
      }
    printf ("\nS=%d", S);
    getchar();
    return 0;
}
```

Iteration Constructs: Flags (cont)

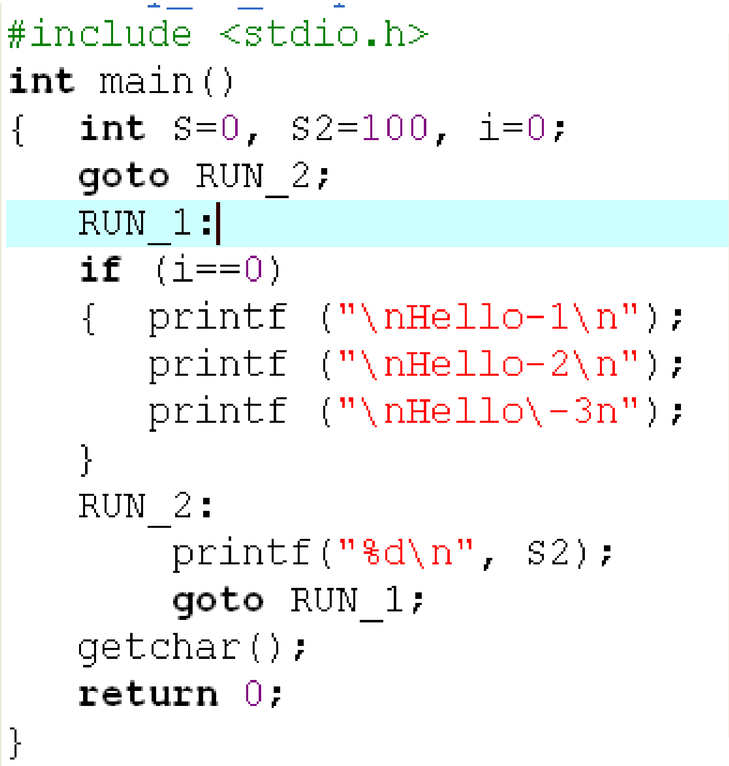
- Example: using **goto**



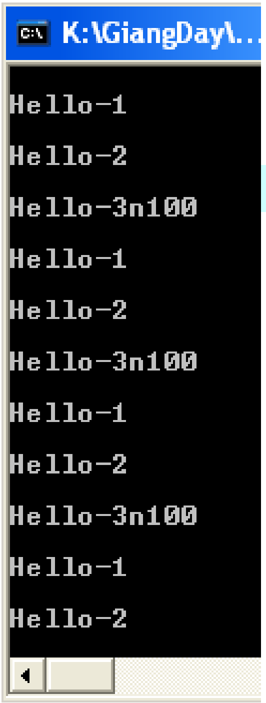
```
#include <stdio.h>
int main()
{   int S=0, i=0;
    goto RUN; /* label for an entry point of a block */
    printf ("\nHello-1\n");
    printf ("\nHello-2\n");
    printf ("\nHello-3\n");
    RUN:
        printf("%d\n", S);
    getchar();
    return 0;
}
```

K:\GiangDay\FU\OOP\BaiTap\Study_in_output.exe

```
0
```



```
#include <stdio.h>
int main()
{   int S=0, S2=100, i=0;
    goto RUN_2;
    RUN_1:
    if (i==0)
    {   printf ("\nHello-1\n");
        printf ("\nHello-2\n");
        printf ("\nHello\-3n");
    }
    RUN_2:
        printf("%d\n", S2);
        goto RUN_1;
    getchar();
    return 0;
}
```



K:\GiangDay\...

```
Hello-1
Hello-2
Hello-3n100
Hello-1
Hello-2
Hello-3n100
Hello-1
Hello-2
Hello-3n100
Hello-1
Hello-2
```

Loop infinitively

Iteration Construct: Flags (cont.)

Use ***a flag*** instead of ***break***

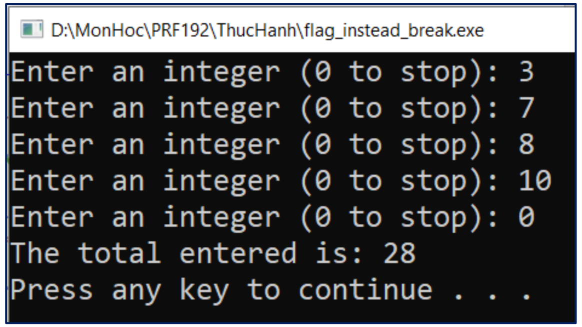
No flag is used.

```c
#include <stdio.h>
#include <stdlib.h>

int main(){
    int i, total = 0;
    int value;

    for(i=0; i<10; i++){
        printf("Enter an integer (0 to stop): ");
        scanf("%d", &value);
        if(value == 0)
            break;   /* POOR PROGRAMMING */
        else
            total += value;
    }

    printf("The total entered is: %d\n", total);

    system("pause");
    return 0;
}
```

A flag is used.

```c
#include <stdio.h>
#include <stdlib.h>

int main(){
    int i, total = 0;
    int value;
    int keepReading = 1;

    for(i=0; i<10 && keepReading==1; i++){
        printf("Enter an integer (0 to stop): ");
        scanf("%d", &value);
        if(value == 0)
            keepReading = 0;
        else
            total += value;
    }

    printf("The total entered is: %d\n", total);

    system("pause");
    return 0;
}
```

Same output:



D:\MonHoc\PRF192\ThucHanh\flag_instead_break.exe

```
Enter an integer (0 to stop): 3
Enter an integer (0 to stop): 7
Enter an integer (0 to stop): 8
Enter an integer (0 to stop): 10
Enter an integer (0 to stop): 0
The total entered is: 28
Press any key to continue . . .
```

Programming Styles

Habits in programming

- A well-written program is a pleasure to read. Other programmers can understand it without significant effort. The coding style is consistent and clear throughout.
- Develop your own style guide or adopt the style outlined here or elsewhere, but adopt some style.
- **Recommendations on**
  - Naming
  - Indentation
  - Comments
  - General Guidelines

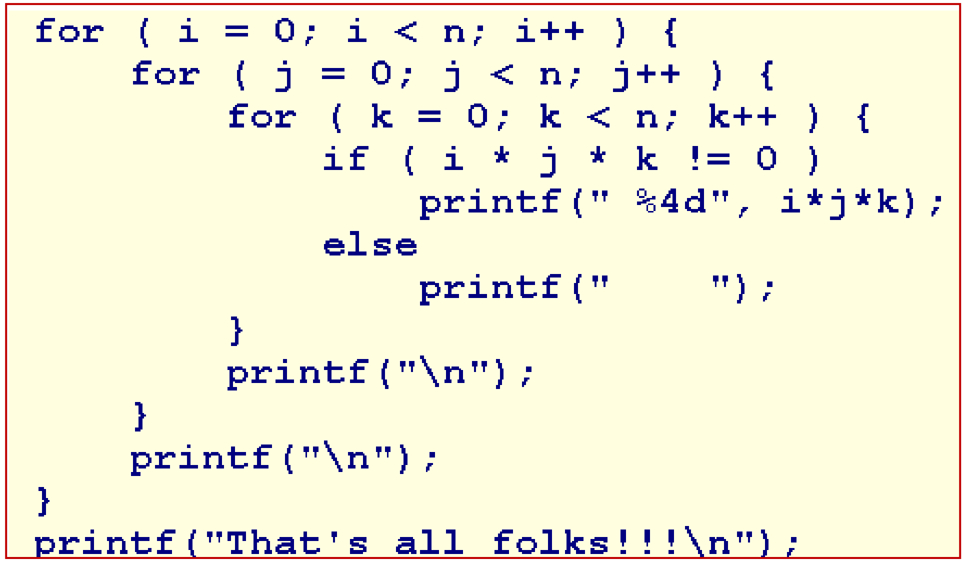
Programming Stytes: Naming

- Adopt names that are self-descriptive so that comments clarifying their meaning are unnecessary
- Use names that describe identifiers completely, avoiding cryptic names
- Prefer nouns for variable names
- Keep variable names short - studentName rather than theNameOfAStudent → (Camel case rule)
- Keep the names of indices very short - treat them as mathematical notation

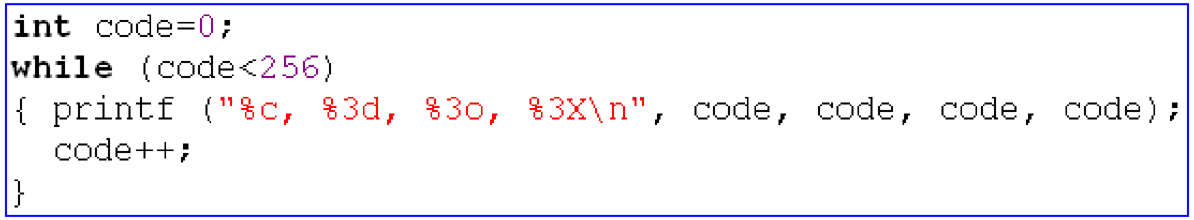
Programming Styles: Indentation

- Indent the body of any construct that is embedded within another construct.

- For example:



```
for ( i = 0; i < n; i++ ) {
    for ( j = 0; j < n; j++ ) {
        for ( k = 0; k < n; k++ ) {
            if ( i * j * k != 0 )
                printf(" %4d", i*j*k);
            else
                printf("    ");
        }
        printf("\n");
    }
    printf("\n");
}
printf("That's all folks!!!\n");
```



```
int code=0;
while (code<256)
{ printf ("%c, %3d, %3o, %3X\n", code, code, code, code);
  code++;
}
```

Note: Use in-line opening braces or start opening braces on a newline but don't mix the two styles.

Programming Styles: Comment

- Use comments to declare what is done, rather than describe how it is done.
- Comments introduce what follows.
- Keep them brief and avoid decoration.

Programming Styles: Guidelines

- Limit line length to 80 characters - both comments and code
- Avoid global variables
- Select data types for variables wisely and carefully
- Initialize a variable when declaring it only if the initial value is part of the semantic of the variable.
- If the initial value is part of an algorithm, use a separate assignment statement.
- Avoid **goto**, **continue**, **break** except in switch.
- Avoid using the character encodings for a particular machine.
- Use a single space or no spaces either side of an operator.

Programming Styles: Guidelines

- Use in-line opening braces or start opening braces on a newline but don't mix the two styles.
- Initialize iteration variables in the context of the iteration
- Avoid assignments nested inside logical expressions
- Avoid iterations with empty bodies - reserve the body for the algorithm
- Limit the initialization and iteration clauses of a **for** statement to the iteration variables
- Distribute and nest complexity

Programming Styles: Guidelines

- Avoid fancy algorithms that may be efficient but are difficult to read
- Add additional comments where code has been fine tuned for efficient execution
- Add an extra pair of parentheses where an assignment is also used as a condition
- Remove unreferenced variables
- Remove all commented code and debugging statements from release and production code

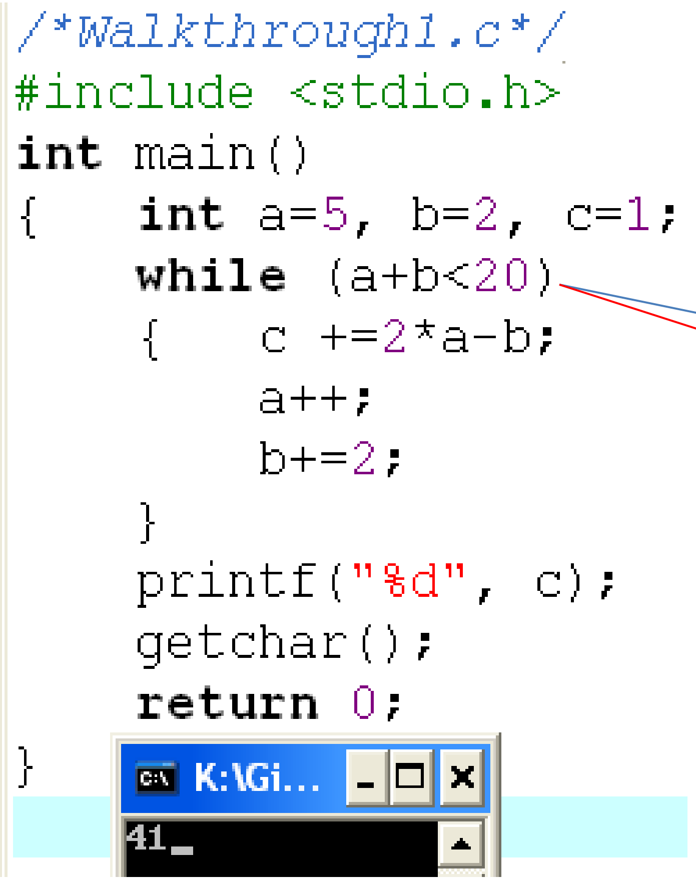
Walkthroughs

Walkthroughs

- Understand code is a skill of programmer.
- To understand code, we should know how the code execute.
- To know how the code execute, we should perform each instruction.
- **A walkthrough is**
  - A record of the changes that occur in the values of program variables as a program executes and
  - A listing of the output, if any, produced by the program.
- **Ways to perform a walkthrough**
  - Memory Map → You knew that
  - Walkthrough Tables → A simpler way
- **Debug a program**
  - What is debugging?
  - Why use debugging?

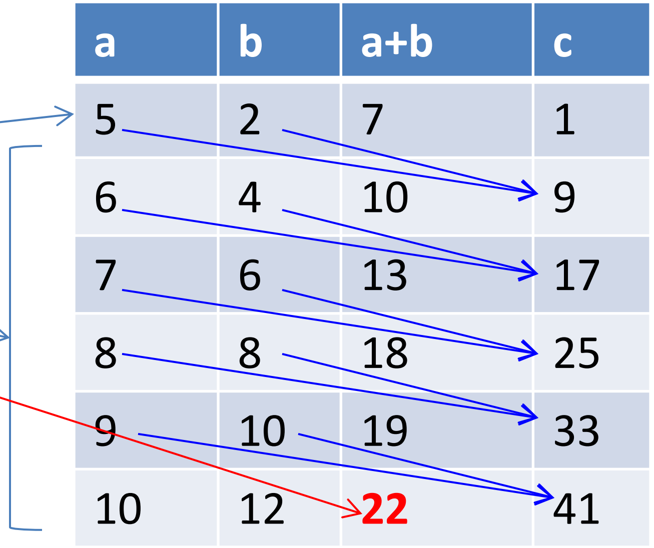
Walkthroughs (cont.)

- Example 1:



```
/*Walkthrough1.c*/
#include <stdio.h>
int main()
{    int a=5, b=2, c=1;
     while (a+b<20)
     {    c +=2*a-b;
          a++;
          b+=2;
     }
     printf("%d", c);
     getchar();
     return 0;
}
```

```
41_
```



a	b	a+b	c
5	2	7	1
6	4	10	9
7	6	13	17
8	8	18	25
9	10	19	33
10	12	**22**	41

Output
41

Walkthroughs (cont.)

- Example 2:

```
int n, i, S=0;
scanf("%d", &n);
for (i=1; i<=n; i+=3)
    if (i%2!=0 && i%3!=0) S+=i;
printf("%d", S);
```

What is the output if the input is 15?

n	15					
i	1	4	7	10	13	16
S	0+1 → 1		1+7 → 8		8+13→21	

S=21

Walkthroughs - Exercise

```
int m,n, i, S=0;
scanf(“%d%d”, &n, &m);
for (i=m; i<=n; i++) S+=i;
printf(“%d”, S);
```

What is the output if the input are 8 12?

Test the program:

```
int m,n, i, S=0;
scanf(“%da%d”, &n, &m);
for (i=m; i<=n; i++) S+=i;
printf(“%d”, S);
```

What is the output if the input are 8a12?

Modify	Input
“%d %d”	12 8
“%d%d”	12 8
“%d%d”	12 8
“%d-%d”	12-8

Debug a program

What is debugging?

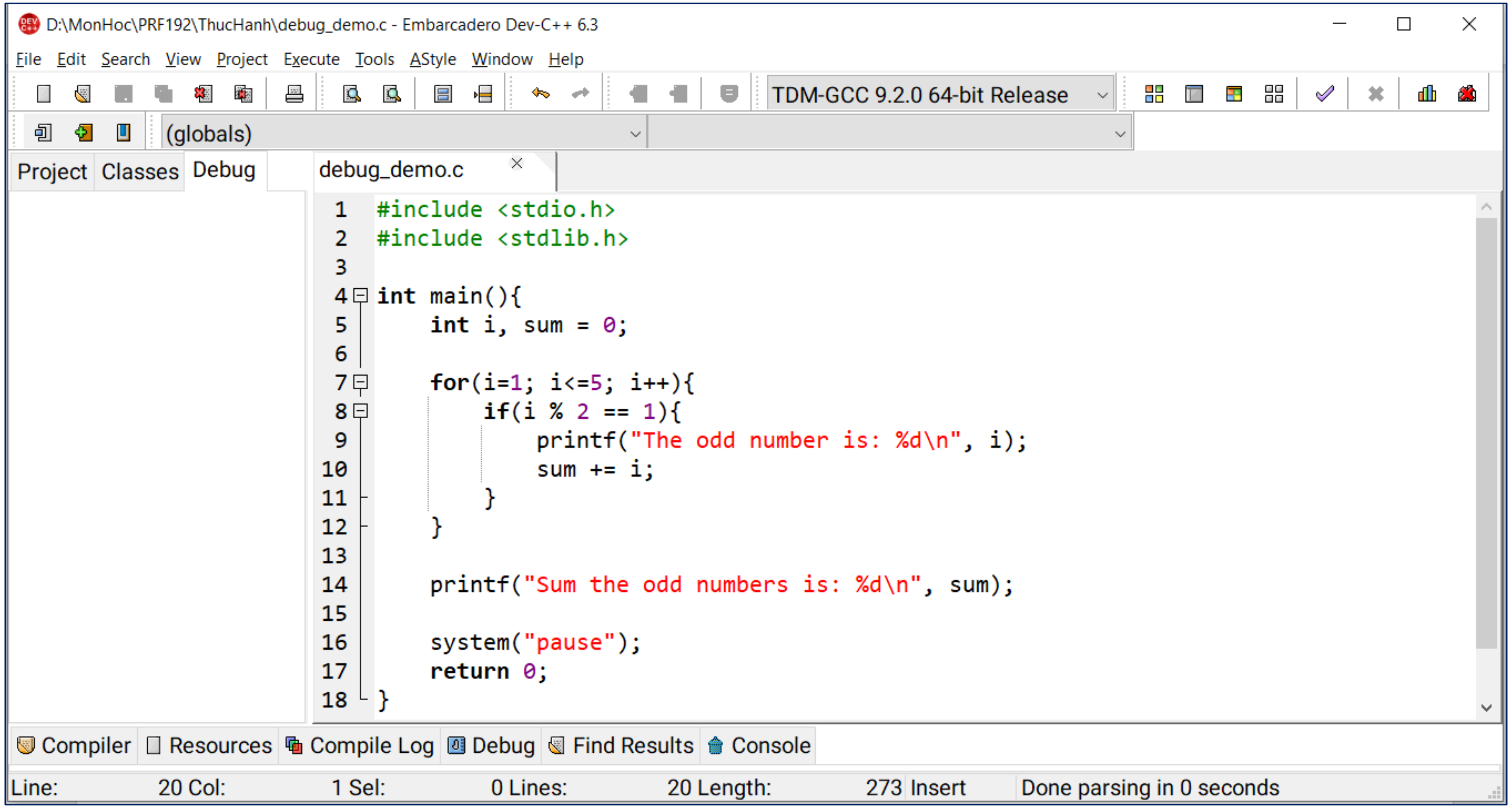
- Debugging is the process of **identifying**, **analyzing**, and **fixing errors** (or bugs) in the source code of a program.

Why use debugging?

- To identify and fix errors
- To save time and effort
- To understand program behavior
- To improve software quality
- To prevent future issues

How to debug a simple program? (using Dev-C++)

- **Step 1**: Write Your Code



D:\MonHoc\PRF192\ThucHanh\debug_demo.c - Embarcadero Dev-C++ 6.3
File Edit Search View Project Execute Tools AStyle Window Help
TDM-GCC 9.2.0 64-bit Release
(globals)
Project Classes Debug
debug_demo.c
Compiler Resources Compile Log Debug Find Results Console

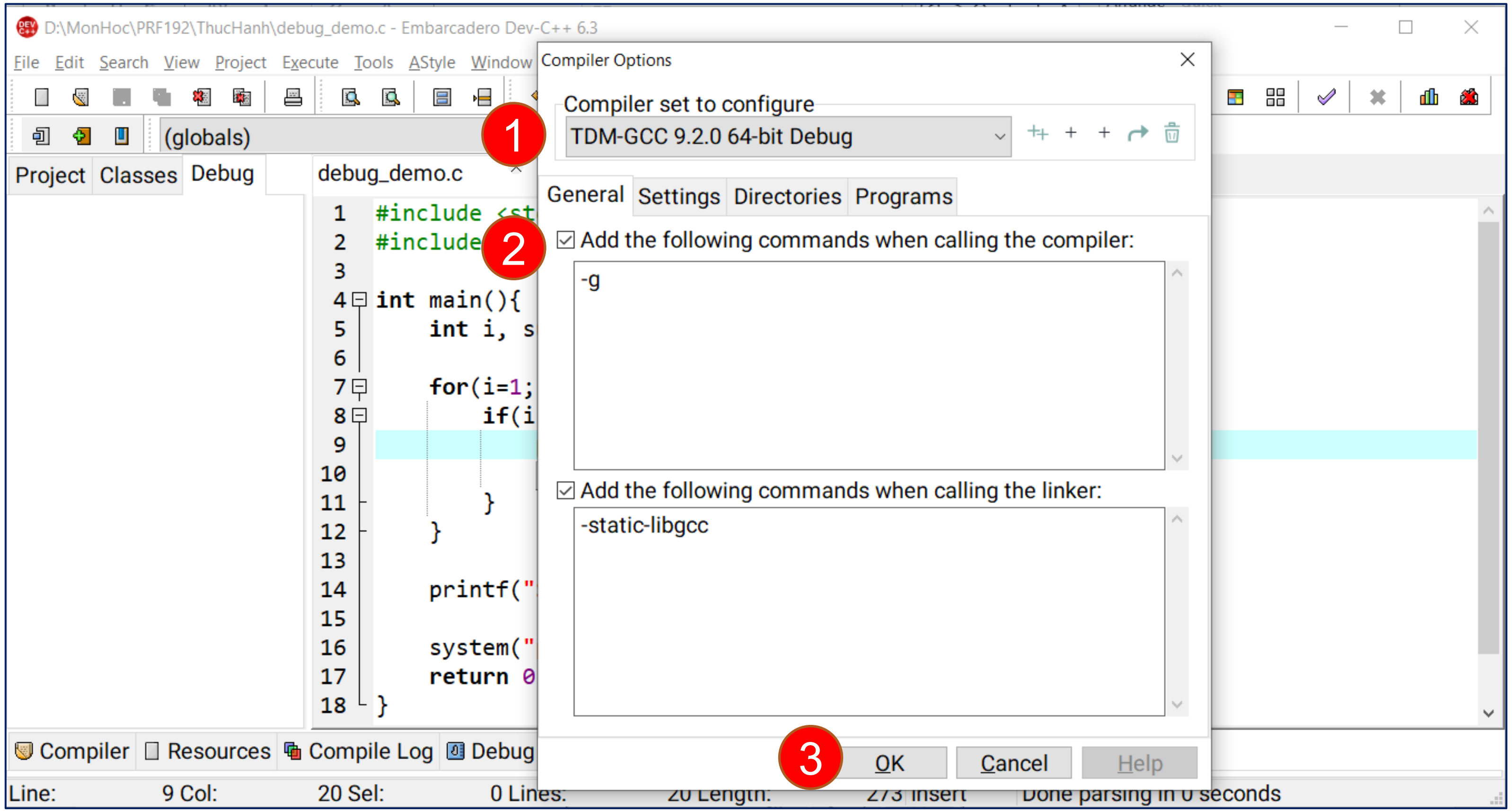
```c
#include <stdio.h>
#include <stdlib.h>

int main(){
    int i, sum = 0;

    for(i=1; i<=5; i++){
        if(i % 2 == 1){
            printf("The odd number is: %d\n", i);
            sum += i;
        }
    }

    printf("Sum the odd numbers is: %d\n", sum);

    system("pause");
    return 0;
}
```

Line: 20 Col: 1 Sel: 0 Lines: 20 Length: 273 Insert Done parsing in 0 seconds

How to debug a simple program? (cont.)

- **Step 2**: Enable Debugging

Go to **Tools** → **Compiler Options** → Change the configuration as follows:


D:\MonHoc\PRF192\ThucHanh\debug_demo.c - Embarcadero Dev-C++ 6.3
File Edit Search View Project Execute Tools AStyle Window
(globals)
Project Classes Debug
debug_demo.c
Compiler Resources Compile Log Debug
Line: 9 Col: 20 Sel: 0 Lines: 20 Length: 273 Insert Done parsing in 0 seconds
Compiler Options
General Settings Directories Programs
OK Cancel Help
1
2
3

```
#include <st
#include
int main(){
    int i, s
    for(i=1;
        if(i
        }
    }
    printf("
    system("
    return 0
}
```

Compiler set to configure: TDM-GCC 9.2.0 64-bit Debug

☑ Add the following commands when calling the compiler:

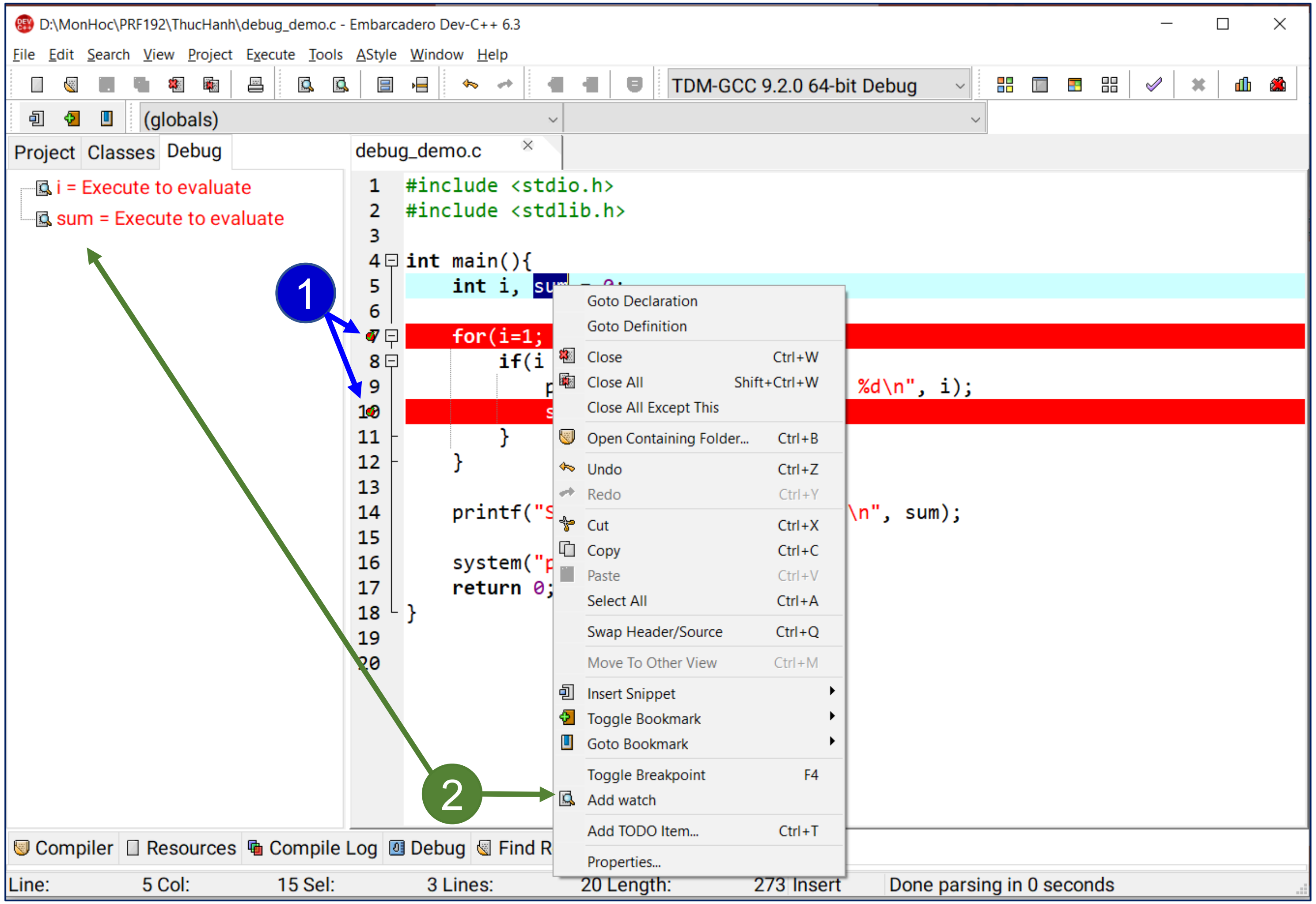
-g

☑ Add the following commands when calling the linker:

-static-libgcc

How to debug a simple program? (cont.)

- **Step 3:**
  - **Set Breakpoints**: To set a **breakpoint**, click on the margin (left of the line numbers) next to the line of code where you want the program to pause → A red dot will appear, indicating the breakpoint.
  - **Add watch:** To watch the changes of each variable → Right click on the variable → **Add watch**



i = Execute to evaluate
sum = Execute to evaluate

```
#include <stdio.h>
#include <stdlib.h>

int main(){
    int i, su

    for(i=1;
        if(i
            p                      %d\n", i);
            s
        }
    }

    printf("S                      \n", sum);

    system("p
    return 0;
}
```

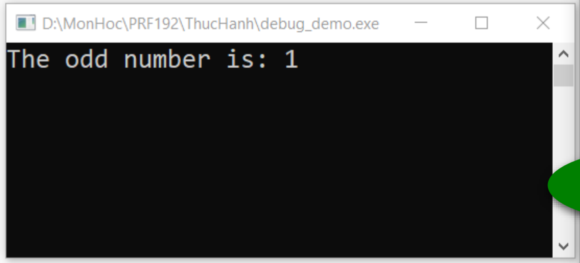
Line: 5 Col: 15 Sel: 3 Lines: 20 Length: 273 Insert Done parsing in 0 seconds

How to debug a simple program? (cont.)

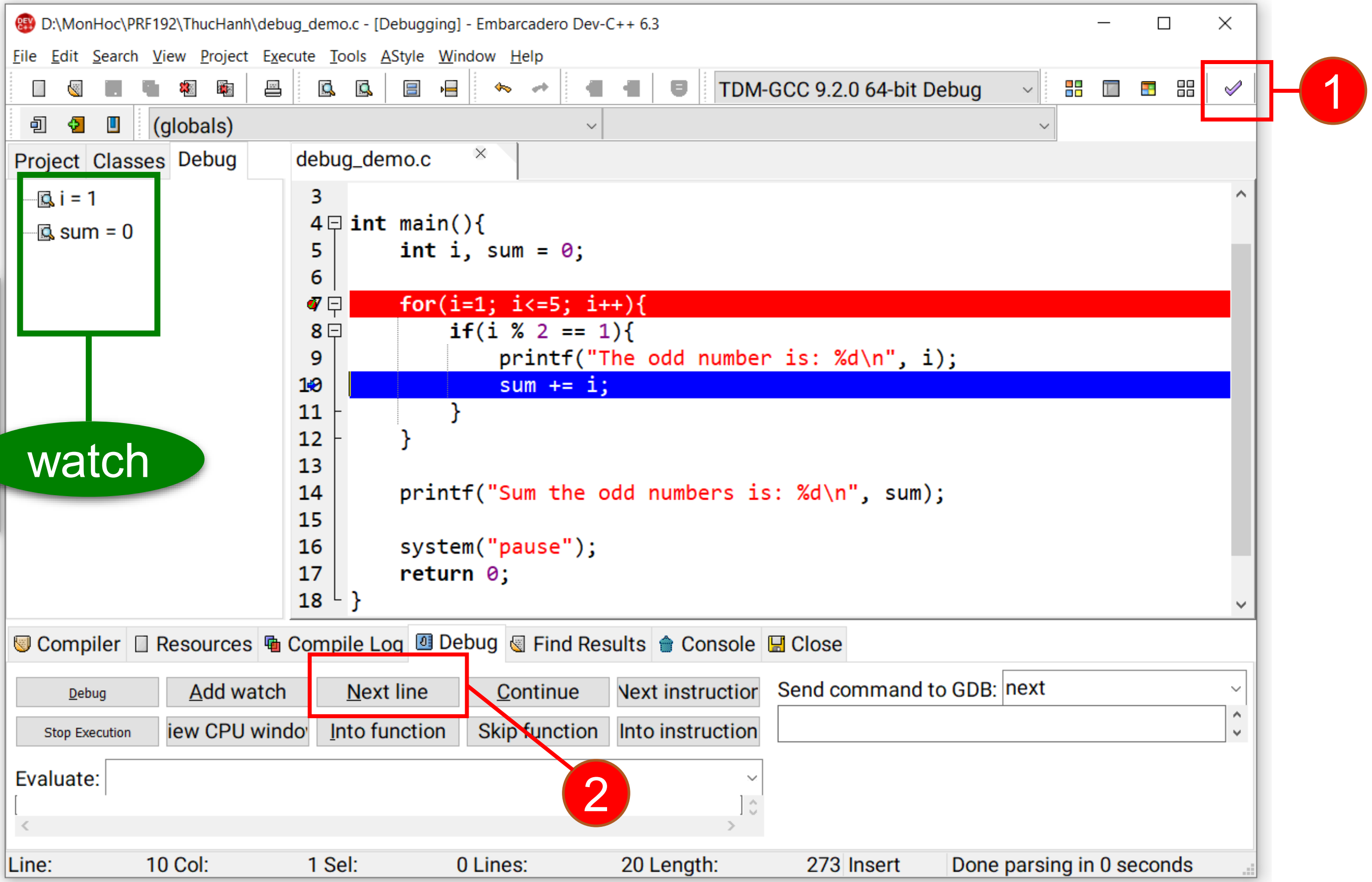
- **Step 4**: Run the Debugger

Go to **Execute** → **Debug** (or Debug icon ✓ on Toolbar) → Next Line (or **F7**)

Output:



```
The odd number is: 1
```


D:\MonHoc\PRF192\ThucHanh\debug_demo.c - [Debugging] - Embarcadero Dev-C++ 6.3
File Edit Search View Project Execute Tools AStyle Window Help
TDM-GCC 9.2.0 64-bit Debug
(globals)
Project Classes Debug
debug_demo.c
watch
1
Compiler Resources Compile Log Debug Find Results Console Close
Debug
Add watch
Next line
Continue
Next instruction
Stop Execution
iew CPU windo
Into function
Skip function
Into instruction
Send command to GDB:
Evaluate:
2

```
i = 1
sum = 0
```

```

int main(){
    int i, sum = 0;

    for(i=1; i<=5; i++){
        if(i % 2 == 1){
            printf("The odd number is: %d\n", i);
            sum += i;
        }
    }

    printf("Sum the odd numbers is: %d\n", sum);

    system("pause");
    return 0;
}
```

Send command to GDB: next

Line: 10 Col: 1 Sel: 0 Lines: 20 Length: 273 Insert Done parsing in 0 seconds

Exercise

- Explain the types of errors in the program?
  - Syntax Errors
  - Logical Errors
  - Runtime Errors
  - Semantic Errors
- Write a simple program with the types of errors that are likely to occur. Also, use the Dev-C++ debugger to debug and revise the program.

Summary (cont.)

- **Logic constructs** = Statements can be used in a program.
  - 3 basic constructs:
    - Sequence,
    - selection constructs (if, if…else, ?:, switch),
    - Iteration constructs (for/ while/ do … while)
- **Walkthrough**: Code are executed by yourself, Tasks in a walkthrough: a record of the changes that occur in the values of program variables and listing of the output, if any, produced by the program. Debug & Fix code in the program.