Slots 04-05-06
Basic Logics

Module-C-Logic

Logic Constructs
Programming Style
Walkthroughs

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: type var [=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

- 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

Content

1- 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 (1/2, 1/n)
  - Iteration constructs

1.1- 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 <u>one entry point and one exit point</u>.

- The beginning step for developing a program is DESIGN using
  - pseudo-coding or
  - flow charting

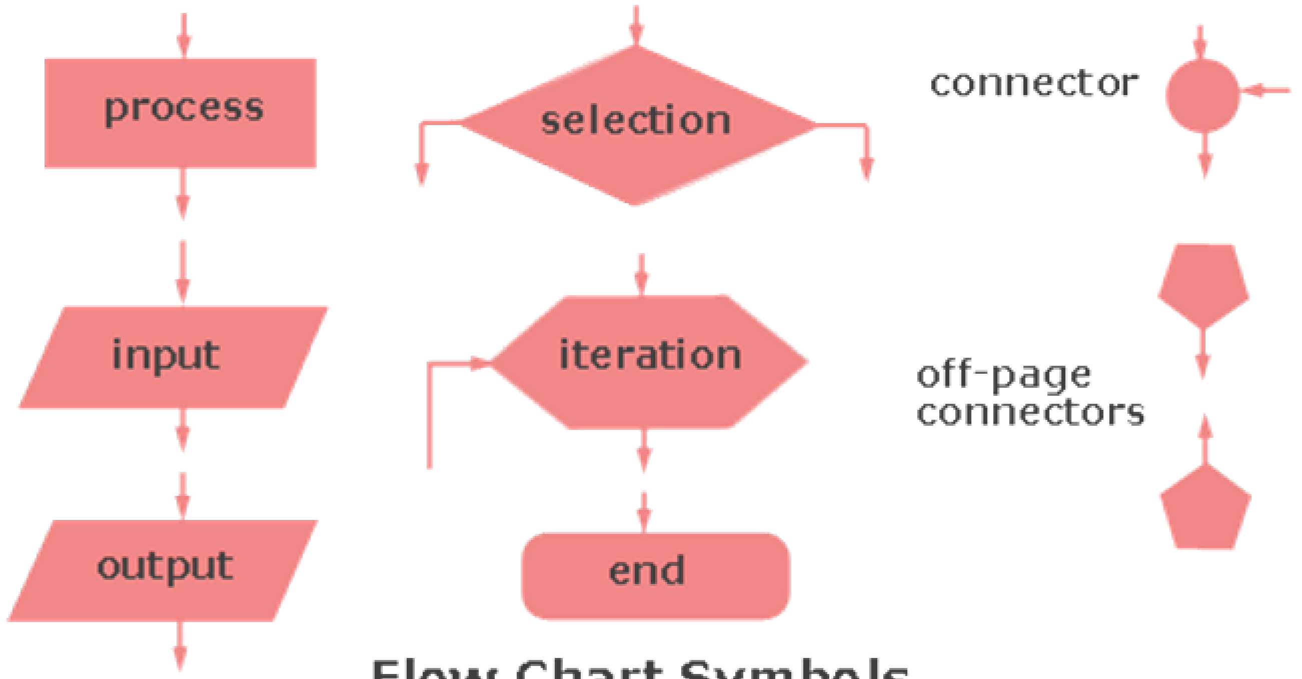
Structured Programming: Pseudo-code

- Example: 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
```

Structured Programming: Flowcharting

- Describe the flow of a program unit symbolically



Flow Chart Symbols

1.2- Sequence Constructs

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

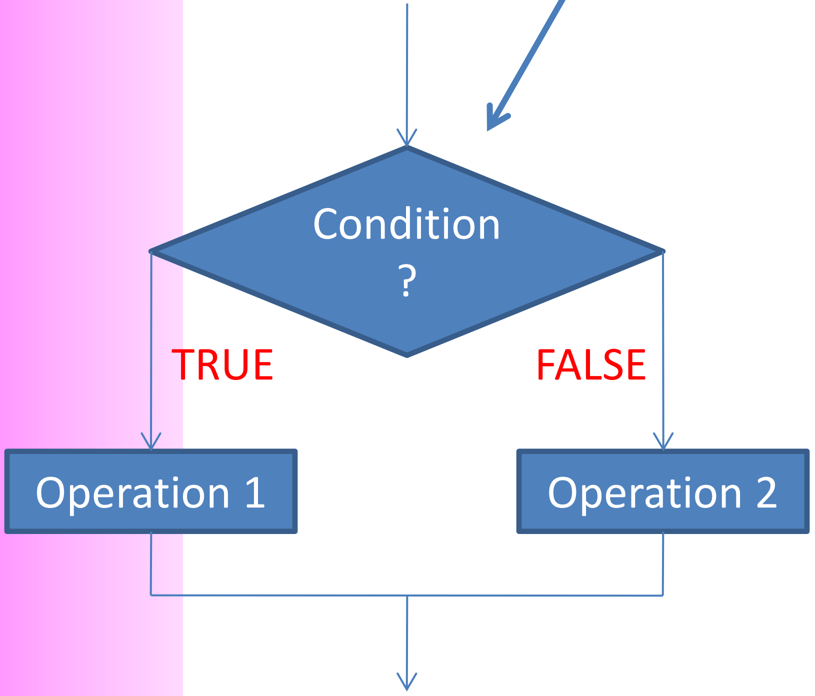
  expression ;
- **Code Blocks**
  - A code block is a set of statements enclosed in curly braces.

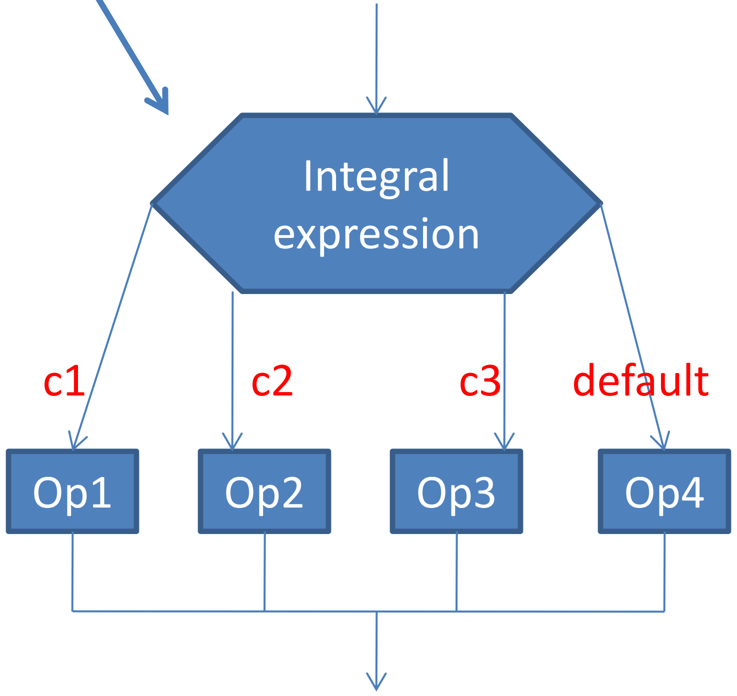
```
{
   statement
   ...
   statement
}
```

1.3- Selection Constructs

Select 1/2	Select 1/n

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



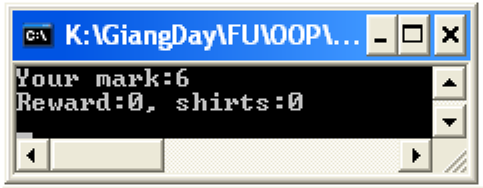


Selection Constructs: if … else

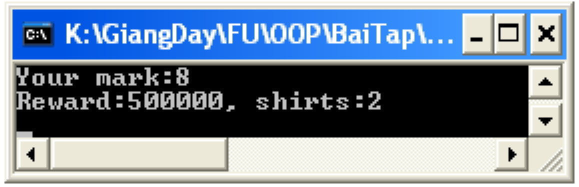
```
if  (condition)
{    statements
}
```

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

```c
#include <stdio.h>
int main()
{   int mark;
    int reward;
    int noOfShirts;
    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);
    getchar(); getchar();
    return 0;
}
```

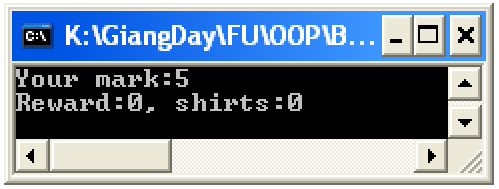


```
Your mark:6
Reward:0, shirts:0
```



```
Your mark:8
Reward:500000, shirts:2
```

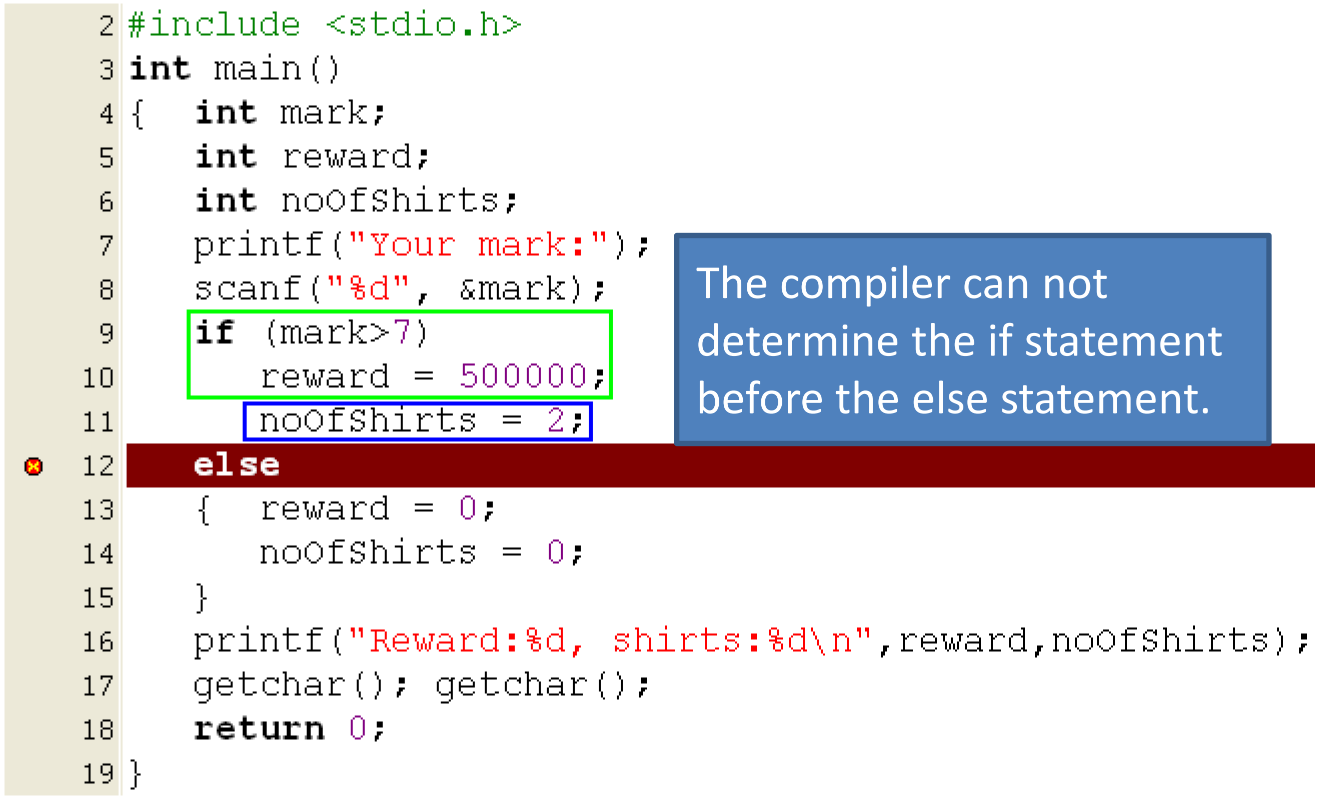
```c
#include <stdio.h>
int main()
{   int mark;
    int reward=0;
    int noOfShirts=0;
    printf("Your mark:");
    scanf("%d", &mark);
    if (mark>7)
    {   reward = 500000;
        noOfShirts = 2;
    }
    printf("Reward:%d, shirts:%d\n",reward,noOfShirts);
    getchar(); getchar();
    return 0;
}
```



```
Your mark:5
Reward:0, shirts:0
```

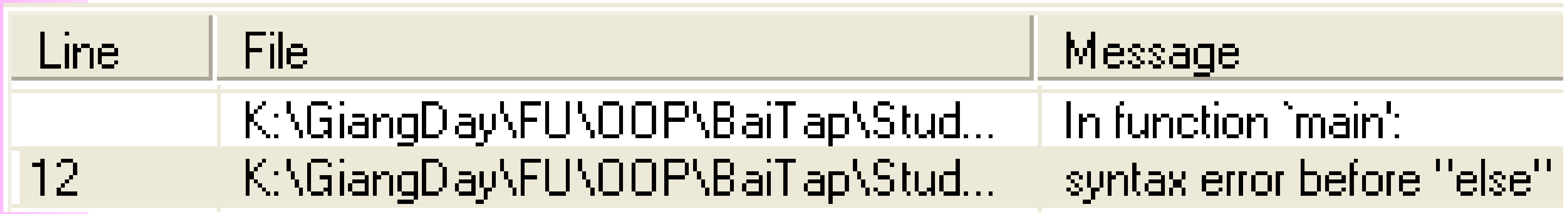
Selection Constructs: if … else

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



```
 2 #include <stdio.h>
 3 int main()
 4 {   int mark;
 5     int reward;
 6     int noOfShirts;
 7     printf("Your mark:");
 8     scanf("%d", &mark);
 9     if  (mark>7)
10        reward = 500000;
11        noOfShirts = 2;
12     else
13     {   reward = 0;
14         noOfShirts = 0;
15     }
16     printf("Reward:%d, shirts:%d\n",reward,noOfShirts);
17     getchar(); getchar();
18     return 0;
19 }
```

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



Line	File	Message
	K:\GiangDay\FU\OOP\BaiTap\Stud...	In function `main':
12	K:\GiangDay\FU\OOP\BaiTap\Stud...	syntax error before "else"

Selection Constructs: if … else

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

Nested if

- **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.

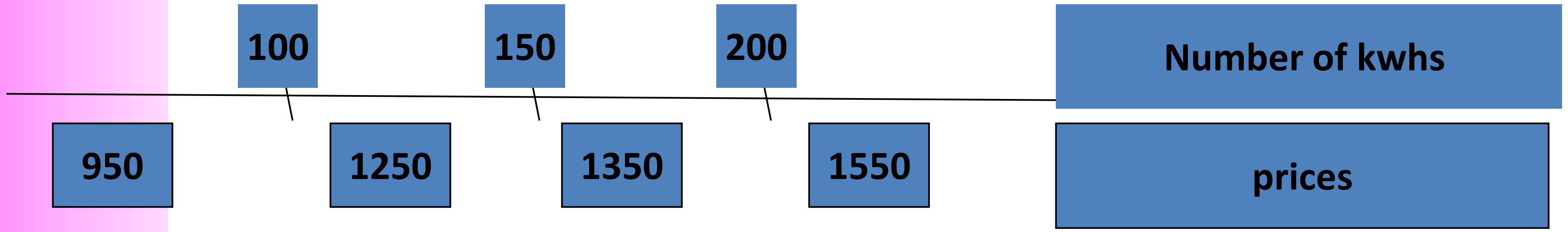
```
if  (N <=3)  t =  N*120000 ;
else if  (N<=6)  t= 3*120000 + (N-3) * 90000;
else if  (N<=10)
      t= 3*120000 + 3*90000 + (N-6)*85000;
else
      t= 3*120000 + 3*90000 + 4*85000 + (N-10)*70000;
```

Selection Constructs: if … else

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

- 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
```



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?

-- either --

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

-- or --

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

?

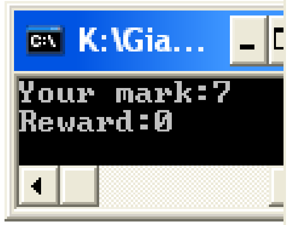
?

The rule in C is that <u>an else always belongs to the innermost if available</u>. Use { } to explicitly determine statements

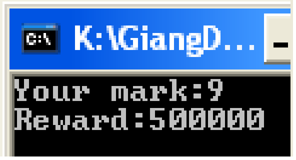
```
if ( windows == 3 )   {
     if ( floor ==1 )
          printf("Found the room\n");
} else
     printf("Try again\n);
```

Selection Constructs: Operator ? :

```c
#include <stdio.h>
int main()
{   int mark;
    int reward;
    printf("Your mark:");
    scanf("%d", &mark);
    reward = mark>8? 500000:0;
    printf("Reward:%d\n",reward);
    getchar(); getchar();
    return 0;
}
```

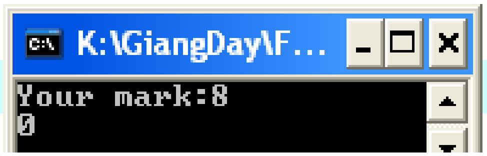


```
Your mark:7
Reward:0
```

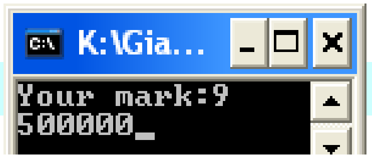


```
Your mark:9
Reward:500000
```

```c
#include <stdio.h>
int main()
{   int mark;
    printf("Your mark:");
    scanf("%d", &mark);
    printf(mark >8? "500000" : "0");
    getchar(); getchar();
    return 0;
}
```

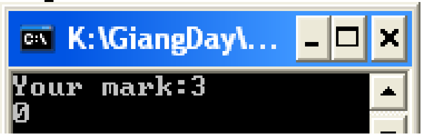


```
Your mark:8
0
```



```
Your mark:9
500000_
```

```c
#include <stdio.h>
int main()
{   int mark;
    printf("Your mark:");
    scanf("%d", &mark);
    (mark >8)? printf("500000") : printf("0");
    getchar(); getchar();
    return 0;
}
```



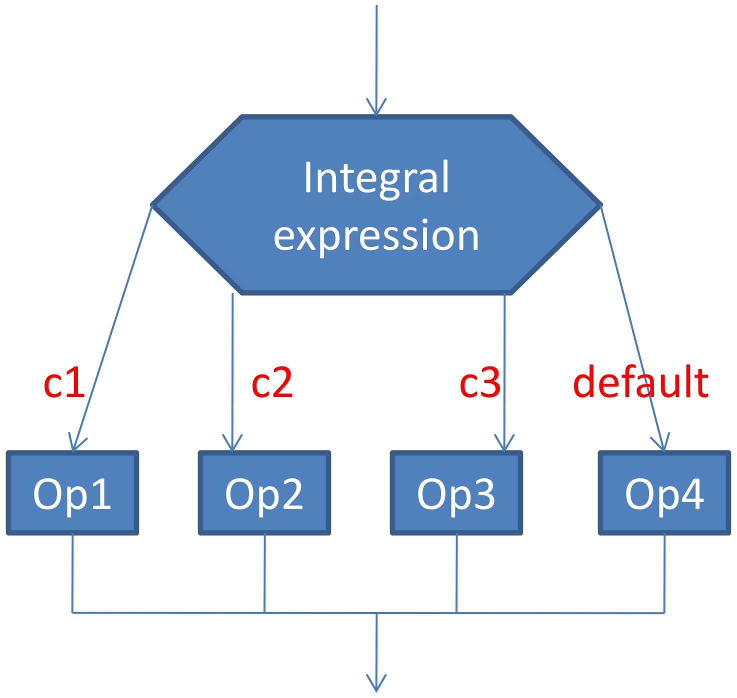
```
Your mark:3
0
```

(condition) ? True_Value : False_Value

Selection Constructs: The switch statement

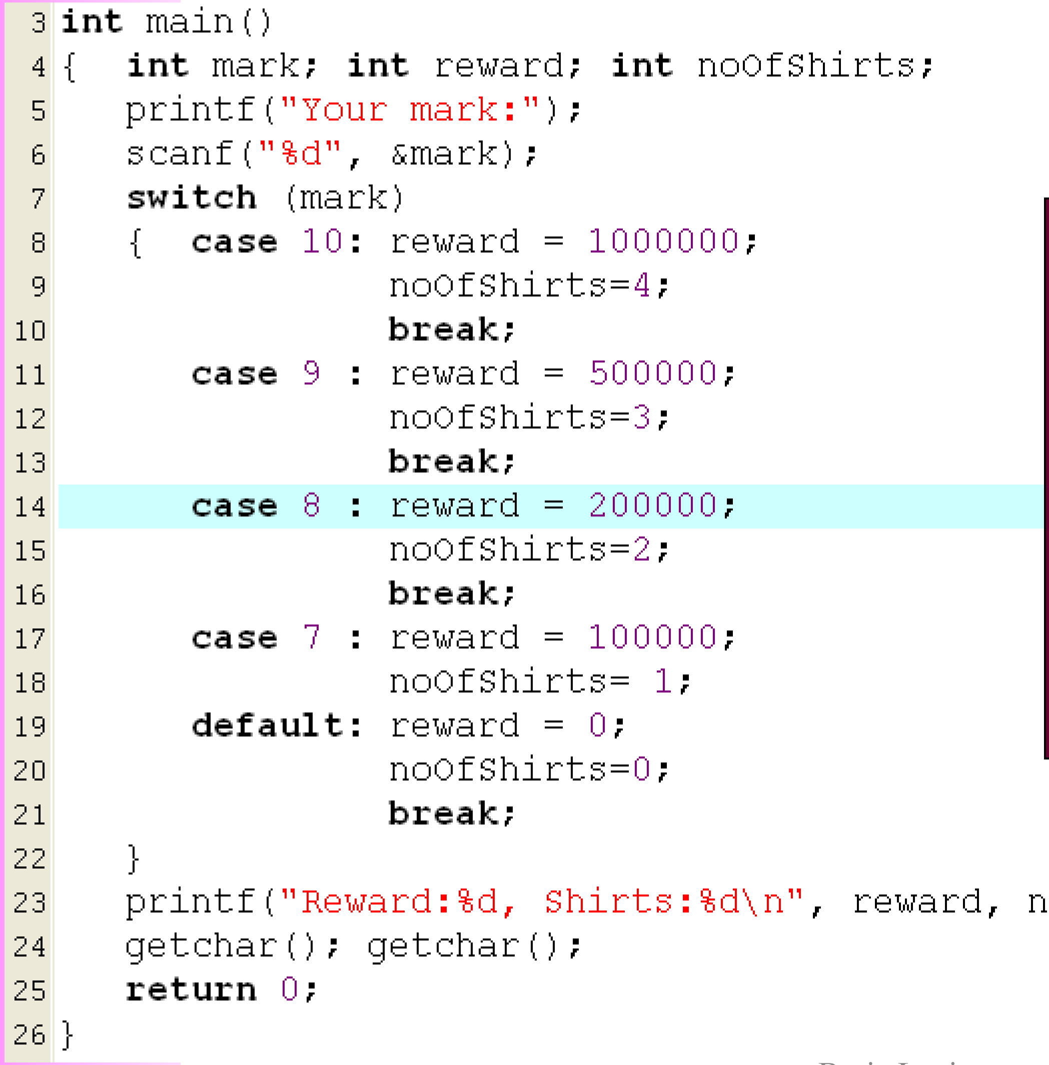
```
switch (variable or expression)
{
    case constant :
        statement(s);
        break;
    case constant :
        statement (s);
        break;
    default:
        statement (s);
}
```

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



```c
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?

Selection Constructs: switch…

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
```

1.4- 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…

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…

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…

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

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;

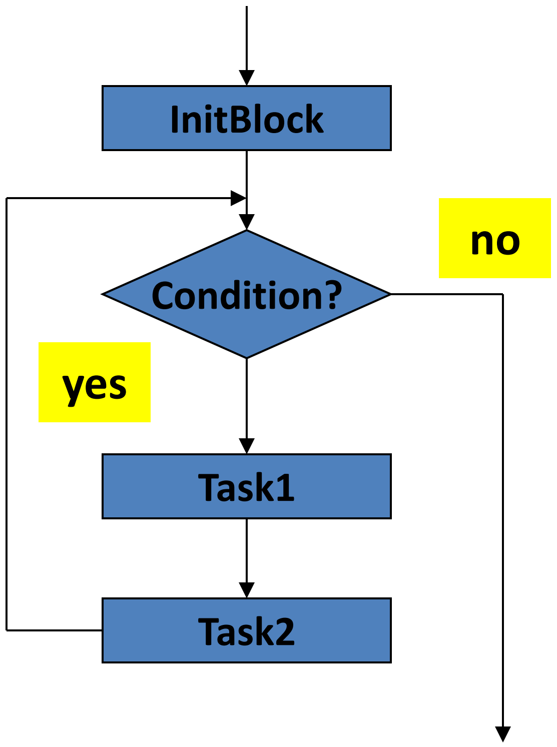
math.h

Iteration: for statement

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

  for (; Condition ; Task2) Task1;
- InitBlock;

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



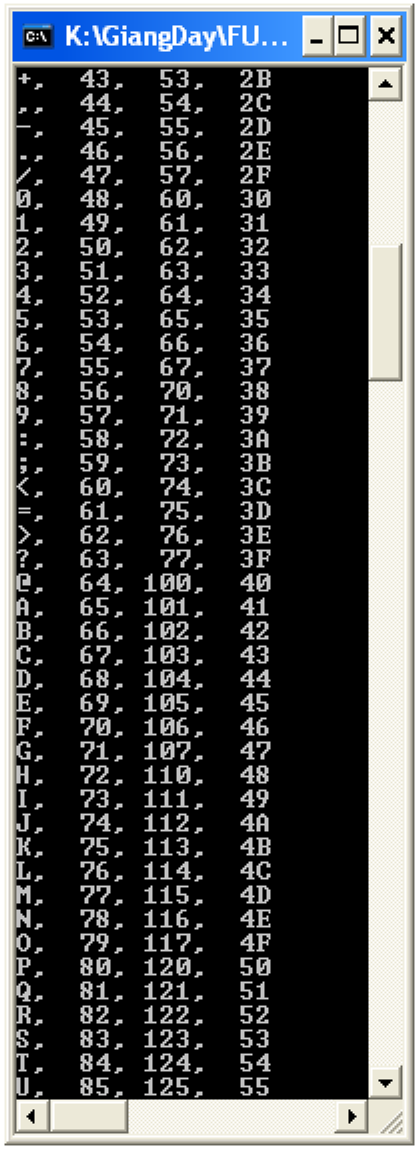
The condition will be tested before tasks are performed.

Iteration…

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

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
```

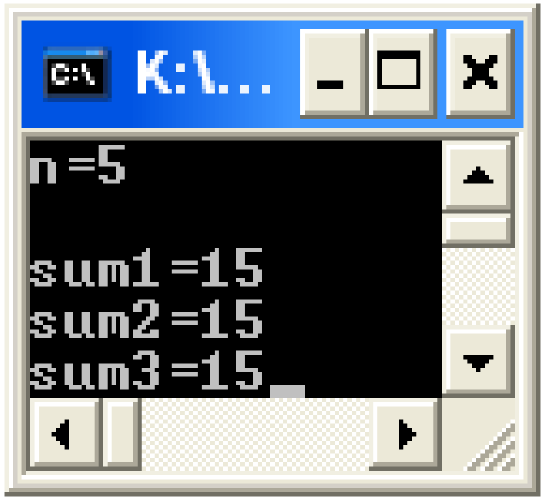
Iteration…

- 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
```

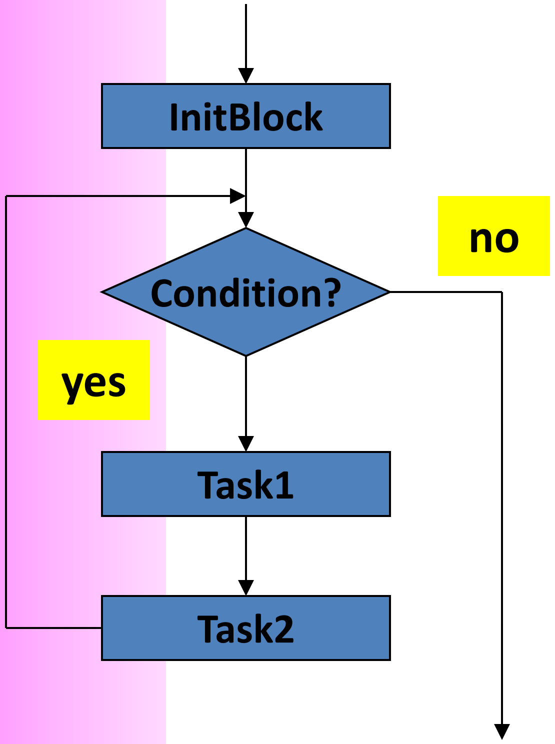
```c
#include <stdio.h>
int main()
{   int n, i;
    printf ("n=");
    scanf ("%d", &n);
    /* Form 1 */
    int sum1=0;
    for (i=1; i<=n; i++) sum1 += i;
    printf ("\nsum1=%d", sum1);
    /* Form 2 */
    int sum2;
    for (sum2=0, i=1; i<=n; sum2+=i, i++ );
    printf ("\nsum2=%d", sum2);
    /* Form 3 */
    int sum3=0;
    i=1;
    for (; i<=n;)
    { sum3+=i;
      i++;
    }
    printf ("\nsum3=%d", sum3);
    getchar();getchar();
    return 0;
}
```

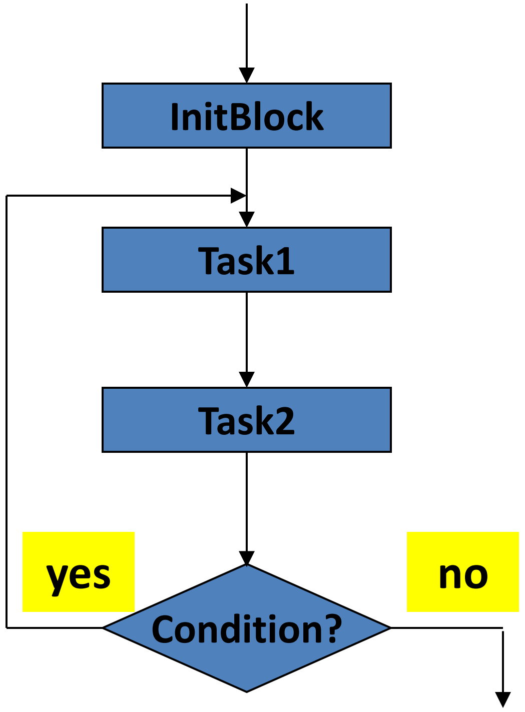


```
n=5

sum1=15
sum2=15
sum3=15
```

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.

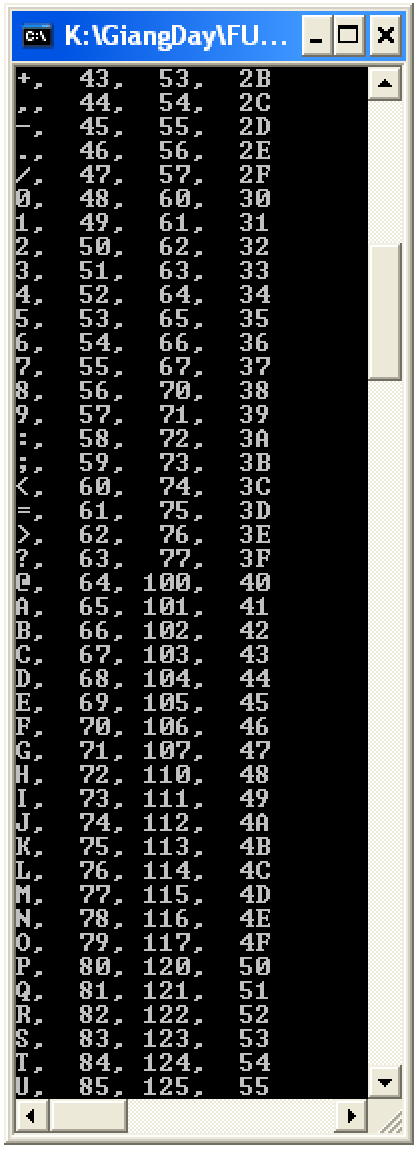
Iteration: while/do…while statements

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

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

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

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



```
+,  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…

- 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…

- 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

Implement it by yourself.

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

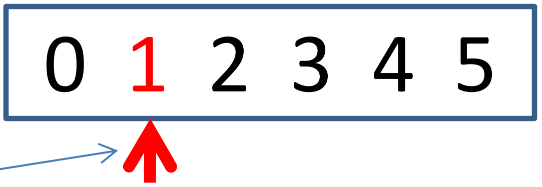
Iteration…

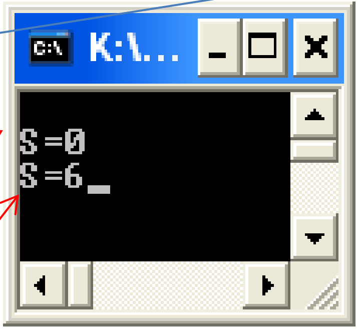
- When a loop is needed, what loop statement should be used?

→ All of them can be used to satisfy your requirement.

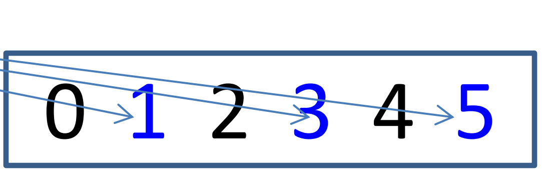
Iteration: Break/ Bypass a loop

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





```
S=0
S=6_
```



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.

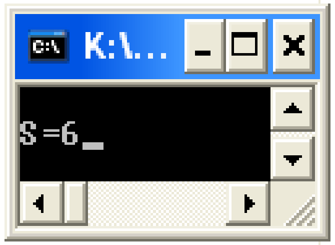
Iteration Constructs: Flags…

- 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…

Use *if* instead of ***continue***

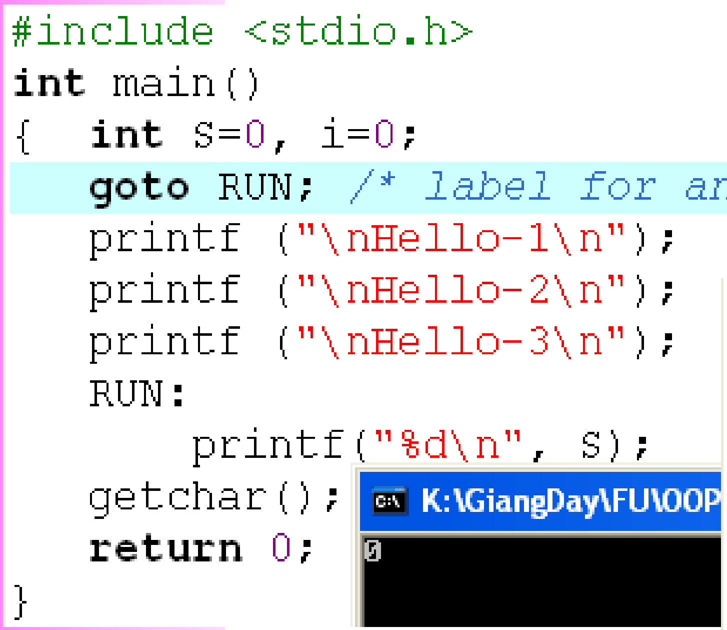
```
#include <stdio.h>
int main()
{   int S=0;
    int i;
    /* Remove continue through using if */
    for (i=0;i<5; i++)
      {   if (i%2==1) continue;
          S+=i;
      }
    printf ("\nS=%d", S);
    getchar();
    return 0;
}
```



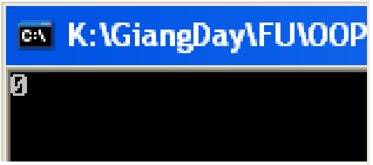
```
S=6_
```

```
#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…

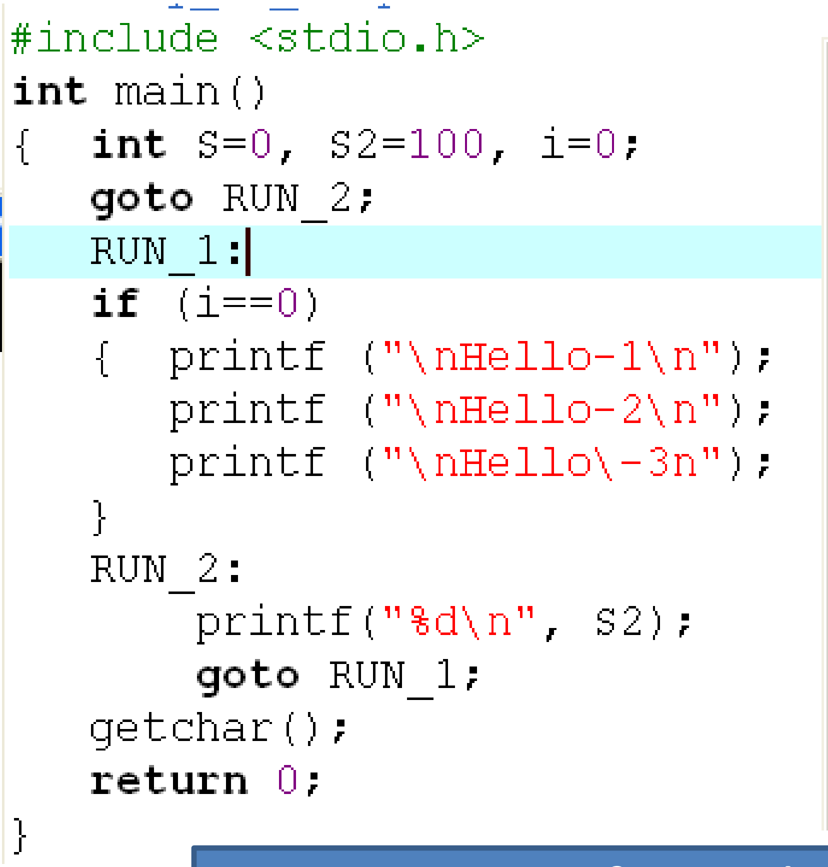


```
#include <stdio.h>
int main()
{   int S=0, i=0;
    goto RUN; /* label for an entry point of a block */
    printf ("\nHello-1\n");
    printf ("\nHello-2\n");
    printf ("\nHello-3\n");
    RUN:
        printf("%d\n", S);
    getchar();
    return 0;
}
```

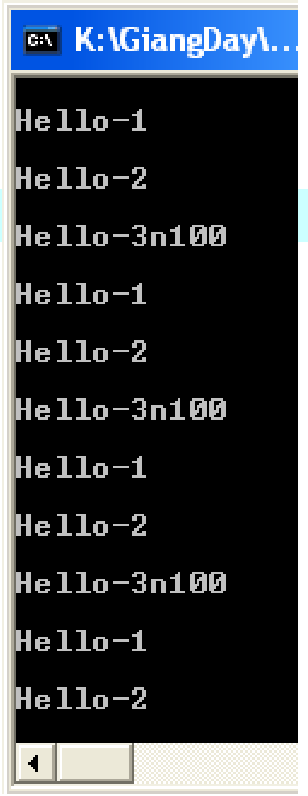


K:\GiangDay\FU\OOP

```
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 Constructs: Flags…

```c
total = 0;
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);
```

No flag is used.

```c
total = 0;
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);
```

A flag is used.

2- 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.

<u>Recommendations on</u>

– Naming
– Indentation
– Comments
– Magic Values
– General Guidelines

Programming Styles: 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
- 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");
```

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

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

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: Magic Values

- These may be mathematical constants, tax rates, default values or names.
- To improve readability, assign symbolic names to these magic values and refer to the symbolic names throughout the code.
- Use the directive

#define SYMBOLIC_NAME value

```
#define PI 3.14159

main () {
     double radius:
     printf("Enter radius : ");
     scanf("%lf", &radius);
     printf("The area of your circle is : %lf\n",
              PI * radius * radius);
}
```

Programming Styles: Magic Values…

- Compiler Idiosyncracies
  - Use **#define** to manage idiosyncracies (đặc điểm) across platforms.
  - For example, the Borland 5.5 compiler does not recognize the **long long** data type. Instead, it recognizes an **_int64** data type. To improve portability, **#define** the data type using a symbolic name such as **LONG_LONG** and embed that name throughout our code.

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 <u>but don't mix the two styles</u>.
- 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

3- Walkthroughs

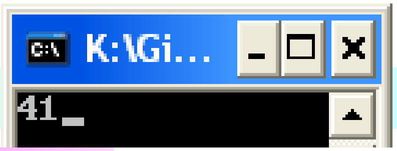
- Understand code is a skill of programmer.
- To understand code, we should know how the code execute.
- To know how the code execute, we should perform each instruction.
- A walkthrough is
  - <u>a record of the changes</u> that occur in the values of <u>program variables</u> as a program executes and
  - a listing of the output, if any, produced by the program.

<u>Ways to perform a walkthrough</u>

- Memory Map → You knew that
- Walkthrough Tables → A simpler way

Walkthroughs: Demo.

```
/*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: Demo.

```
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	$0+1 \rightarrow 1$		$1+7 \rightarrow 8$		$8+13 \rightarrow 21$	

S=21

Walkthroughs: Demo.

```
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

Walkthroughs: Demo.

Study the following code:

```
int n=15;
int S=0;
i=1;
while (i<2*n)
{   S+= i;
    i*=4;
}
```

Give your opinion.

a) S=21

b) S= 85

c) A syntax error

Extra Demo: Print star characters

i

```
     N=6
i= 1 *          Print out 1*, "\n"
   2 **         Print out 2*, "\n"
   3 ***        Print out 3*, "\n"
   4 ****       Print out 4*, "\n"
   5 *****      Print out 5*, "\n"
   6 ******     Print out 6*, "\n"
```

```
Accept N;
for ( i=1; i<=N; i++)
{   for (j=1; j<=i; j++) printf("*");
    printf ("\n");
}
```

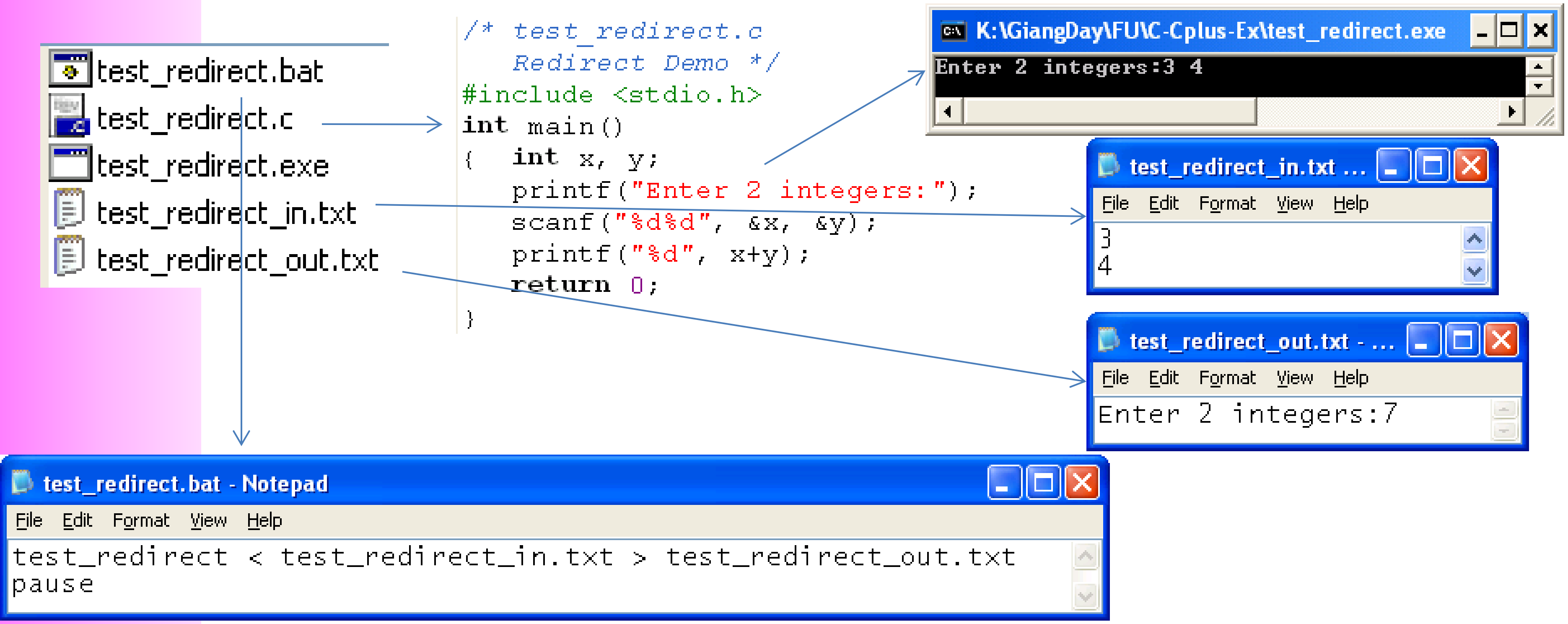
Extra Demo: Multiplication Table

```
N=5
5x 1= 5
5x 2=10
5x 3=15
5x 4=20
5x 5=25
5x 6=30
5x 7=37
5x 8=40
5x 9=45
5x10=50
```

```
Accept n;
for ( i=1; i<=10; i++)
Print out “%dx%2d=%2d\n”, n, i, n*i
```

4- Redirect a Program

- A characteristic is supported by the operating system and it is used in the ACM International contest.



test_redirect.bat
test_redirect.c
test_redirect.exe
test_redirect_in.txt
test_redirect_out.txt

```c
/* test_redirect.c
   Redirect Demo */
#include <stdio.h>
int main()
{  int x, y;
   printf("Enter 2 integers:");
   scanf("%d%d", &x, &y);
   printf("%d", x+y);
   return 0;
}
```

K:\GiangDay\FU\C-Cplus-Ex\test_redirect.exe

```
Enter 2 integers:3 4
```

test_redirect_in.txt

```
3
4
```

test_redirect_out.txt

```
Enter 2 integers:7
```

test_redirect.bat - Notepad

```
test_redirect < test_redirect_in.txt > test_redirect_out.txt
pause
```

➔ So, pay attention to input/output formats.

Summary

- **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.

Thank You