ECE 220: Computer Systems & Programming

Lecture 6: Control Structures & Basic I/O

- MP2 due Tonight by 10:00PM
- Quiz1: 02/05 – 02/07

Thursday, 2/15/2024

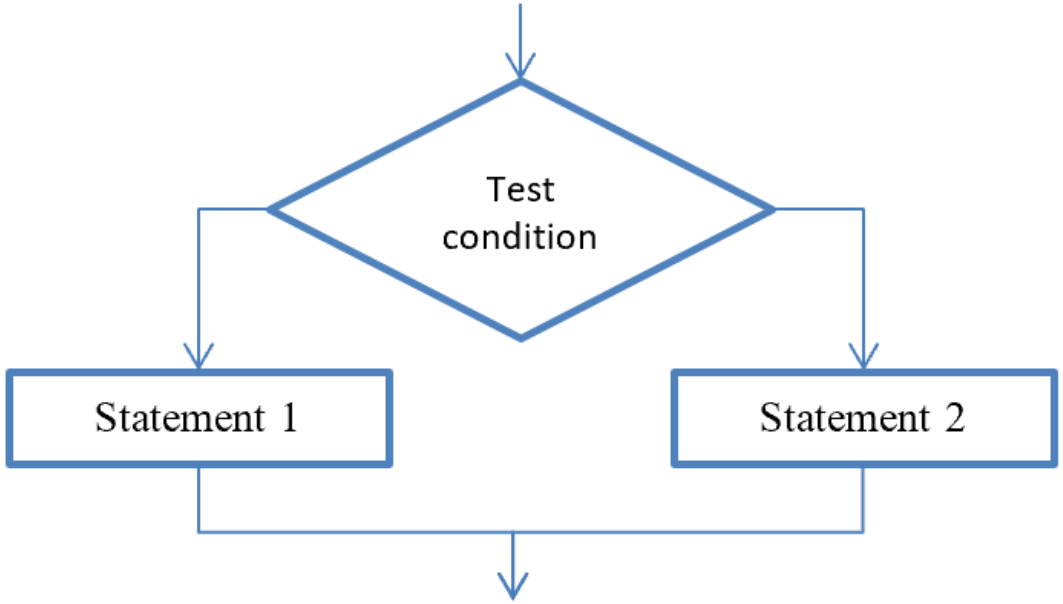
- Regular Exam: 7:00pm - 8:20pm CT
- Conflict Exam: 5:40pm - 7:00pm CT
  - **Conflict Sign-Up is due by 11:59pm CT on Sunday, 2/11/2024.**

Control Structures

- **Conditional**

one or another statement will be executed, but not both, depending on some condition:

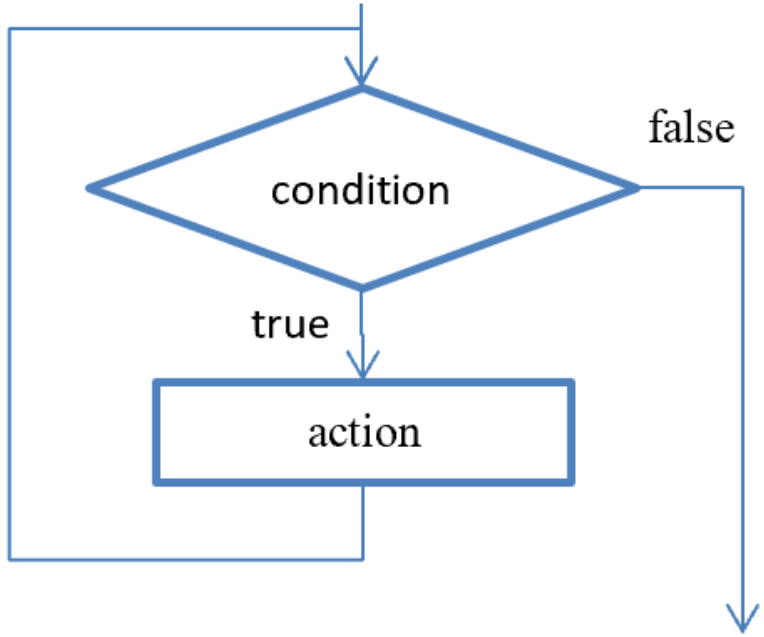
  - `if`
  - `if-else`
  - `switch`



- **Iteration**

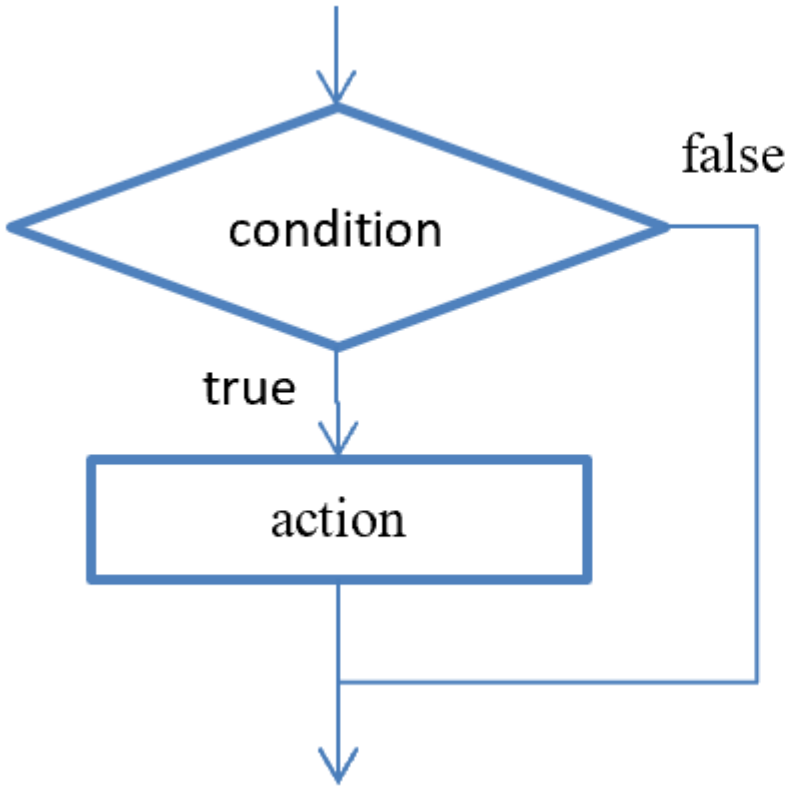
some statements will be executed multiple times until some condition is met:

  - `while`
  - `for`
  - `do-while`



if statement

- `if (condition)`
  `action;`




```
; LC-3 assembly
;
; generate condition code
;
BR(nzp) FALSE
;
; action
;
FALSE
;
```

example

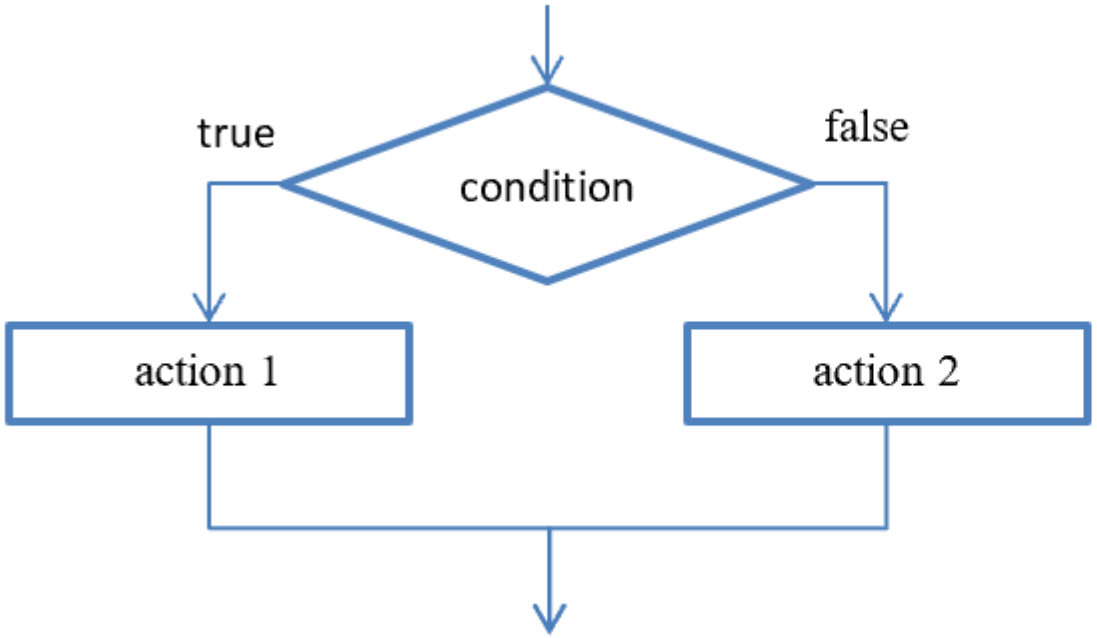
```
if (x < 0)
    x  = -x;                    /* simple statement */

if (x > 5 && x< 25) {
    y = x * x +5;          /* compound statement */
    printf("y=%d\n", y);
}
```

- action statement can be simple, as in first example, or compound, as in second example

if-else statement

```
if (condition)
    action_when_condition_is_true;
else
    action_when_condition_is_false;
```




```
; LC-3 assembly
;
; generate condition code
;
BR(nzp) FALSE
;
; action 1
BRnzp DONE
;
FALSE
; action 2
;
DONE
```

Example

```
if (x < 0)
     x  = -x;
else
     x = x * 2;
```

```
if (x > 5 && x< 25) {
     y = x * x +5;
     printf("y=%d\n", y);
}
else
     printf("x=%f\n", x);
```

common programming errors

- if (x = 2) using assignment operator instead of ==

Associating **if**s with **else**s

- in a cascaded **if-else** statement, an **else** is associated with the closest **if**
  - that is, when not using braces, which is not a good practice

```
if (x != 0)
    if (y > 3)
        z = z / 2;
    else
        z = z + 2;
```

same as

```
if (x != 0) {
    if (y > 3)
        z = z / 2;
    else
        z = z + 2;
}
```

if we really want to associate **else** with the first **if**, then we should use braces:

```
if (x != 0) {
    if (y > 3)
        z = z / 2;
}
else
    z = z + 2;
```

use braces to write clear and readable code!

Floating Number Comparison (Caution)

```
float myFloat = 3.14;

if(myFloat == 3.14)
    printf("My float is PI.\n");
else
    printf("My float is not PI.\n");
```

```
My float is not PI.
```

```
double myDouble = 3.14;

if(myDouble == 3.14)
    printf("My double is PI.\n");
else
    printf("My double is not PI.\n");
```

```
My double is PI.
```

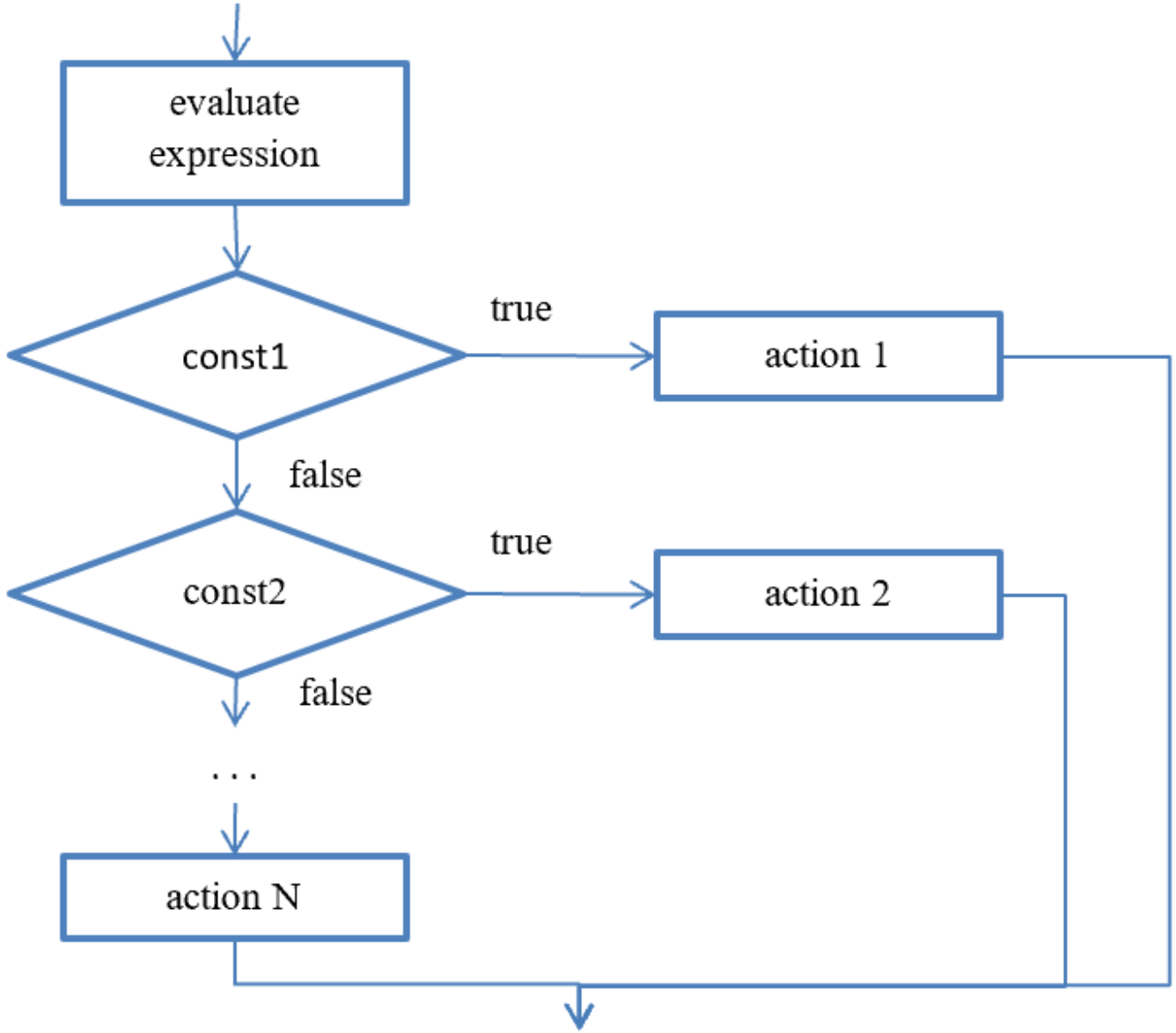
```
printf("%d, %d, %d\n", sizeof(3.14), sizeof(3.14f),  sizeof(myFloat));

8, 4, 4
```

switch statement

- consider example shown in the left column; it also can be implemented as shown on the right:

Using cascaded **if-else** statements	Using **switch** statement
if (expression == const1) *action1;* **else if** (expression == const2) *action2;* **else if** (expression == const3) *action3;* ... **else** *actionN;*	**switch** (expression) { **case** const1: *action1;* **break;** **case** const2: *action2;* **break;** **case** const3: *action3;* **break;** ... **default:** *actionN;* }



this only works when we consider some discrete values to which `expression` is evaluated, `const1, const2, …`

Break Example

```
a = 5;
switch(a){
    case 5:
        printf("E");
        break;
    case 2:
        printf("C");
        break;
    default:
        printf("G");
        break;
}
```

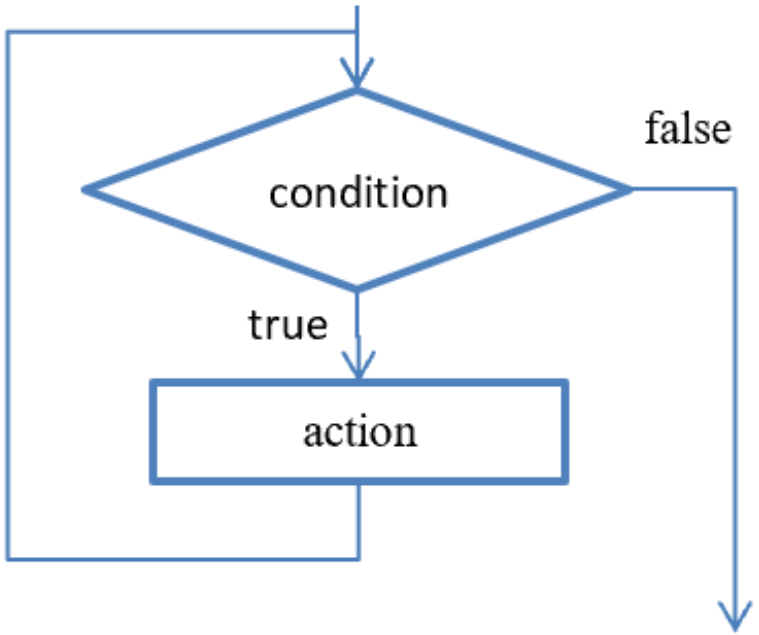
E

```
a = 5;
switch(a){
    case 5:
        printf("E");
    case 2:
        printf("C");
    default:
        printf("G");
}
```

ECG

Iterative constructs

Iterative construct means that some statements will be executed multiple times until some condition is met:



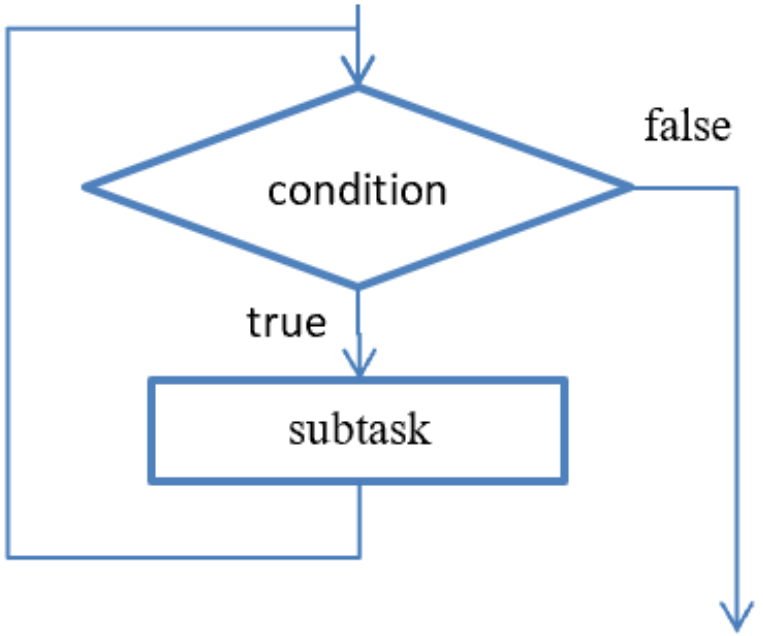

```
; LC-3 implementation
LOOP
; generate condition code
BR(nzp) FALSE
;
; action
;
BRnzp LOOP
;
FALSE
;
```

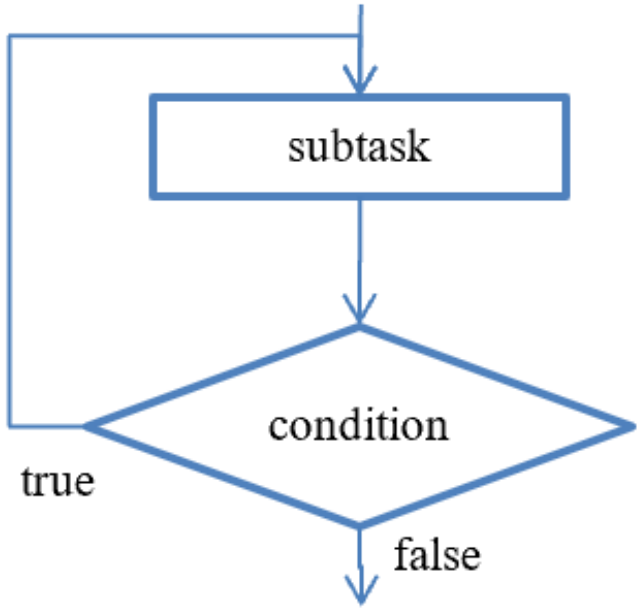
Such construct implements a loop structure in which *action* is executed multiple times, as long as some *condition* is true

- *action* is also called *loop body*

while and do-while statements

- ```
  while (condition) {
      subtask;
  }
  ```
- ```
  do {
      subtask
  } while (condition);
  ```
- For while loop, loop body may or may not be executed even once
- For do-while loop, loop body will be executed at least once



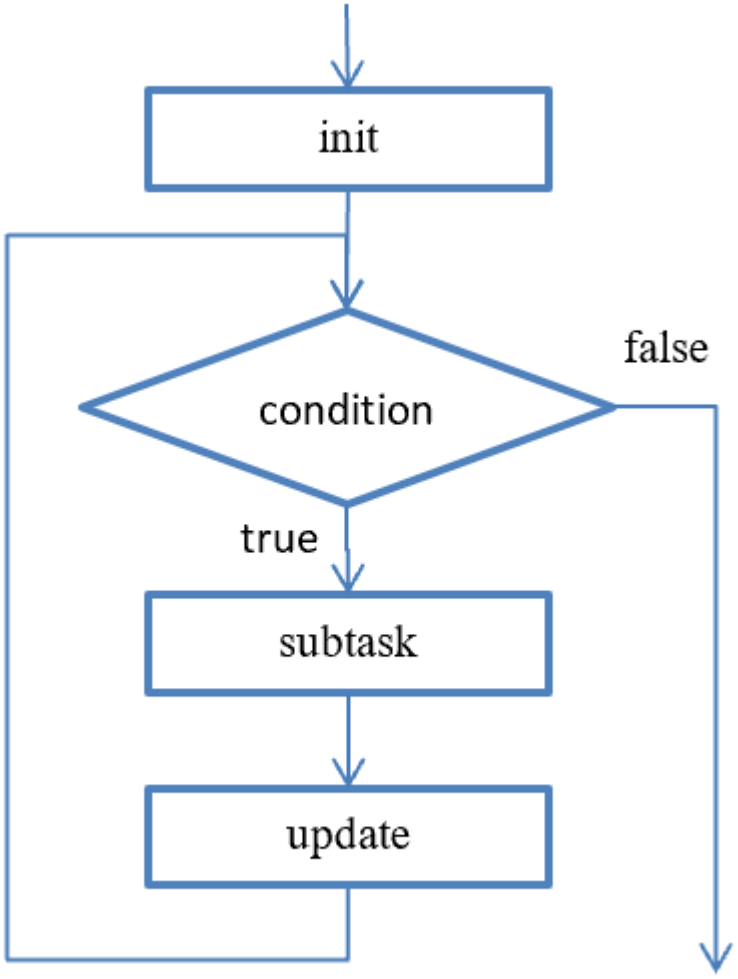


Example

while	do-while
`x = 0;` `while (x < 10) {` `  printf("x=%d\n", x);` `  x = x + 1;` `}`	`x = 0;` `do {` `  printf("x=%d\n", x);` `  x = x + 1;}` `while (x < 10);`

for statement

- **for** (init; test; update) {
  subtask;
  }



Example

While	for
x = 0; while (x < 10) { printf("x=%d\n", x); x = x + 1; }	for (x = 0; x < 10; x++) printf("x=%d\n", x);

break and continue

- `break`

  used only in switch or iteration statement

  break will cause the loop to be terminated

- `continue`
  - used only in iteration statement
  - end the current iteration and start the next

```
for (i = 1; i < 10; i++){
    if(i == 5)
        break;
    printf("%d ",i);
}
```

```
1 2 3 4
```

```
for (i = 1; i < 10; i++){
    if(i == 5)
        continue;
    printf("%d ",i);
}
```

```
1 2 3 4 6 7 8 9
```

Example:

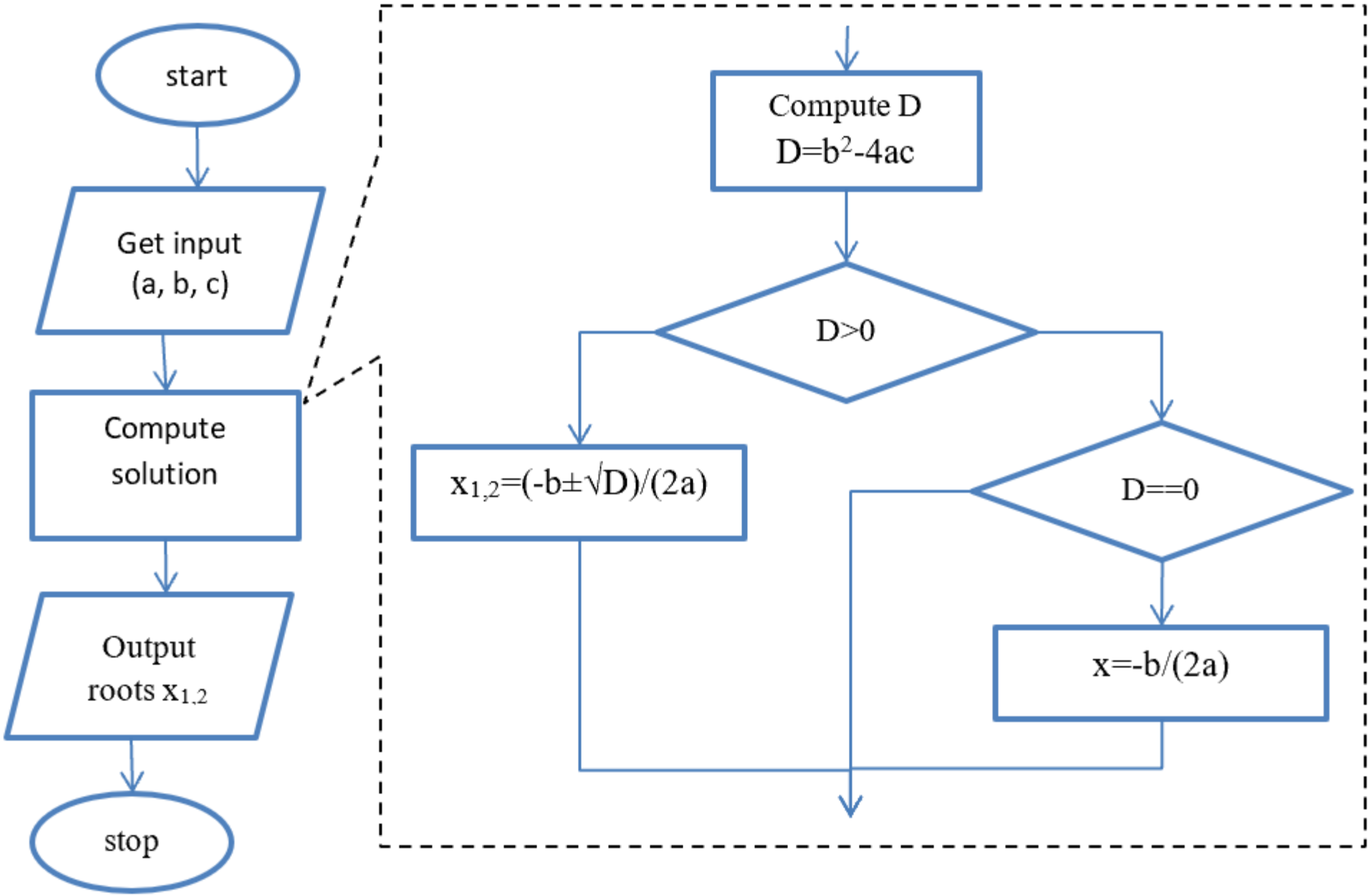
Computing solution of a quadratic equation $ax^2+bx+c=0$

Algorithm:

- $D = b^2 - 4ac$
- If D equals 0, there is one real root: $x = -b/(2a)$
- If D is positive, there are two roots: $x_{1,2} = (-b \pm \sqrt{D})/(2a)$
- If D is negative, no real roots exist

Problem decomposition into separate steps using a flowchart

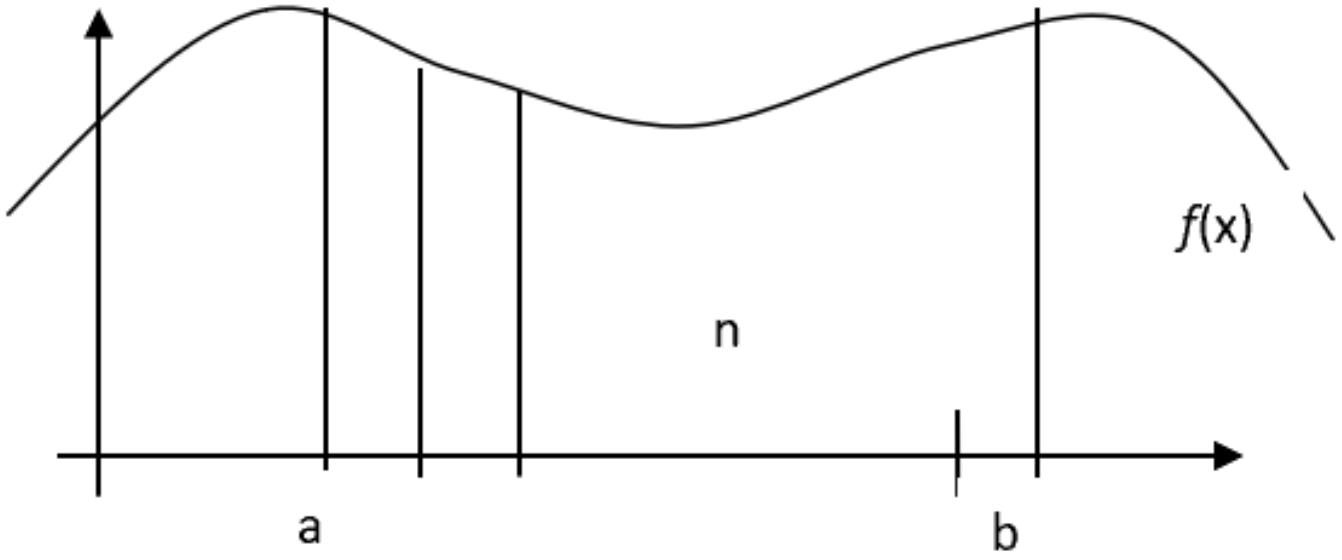
- Get input
- Compute solution according to the above algorithm
- Print output


start
Get input
(a, b, c)
Compute
solution
Output
roots x1,2
stop
Compute D
D=b2-4ac
D>0
x1,2=(-b±√D)/(2a)
D==0
x=-b/(2a)

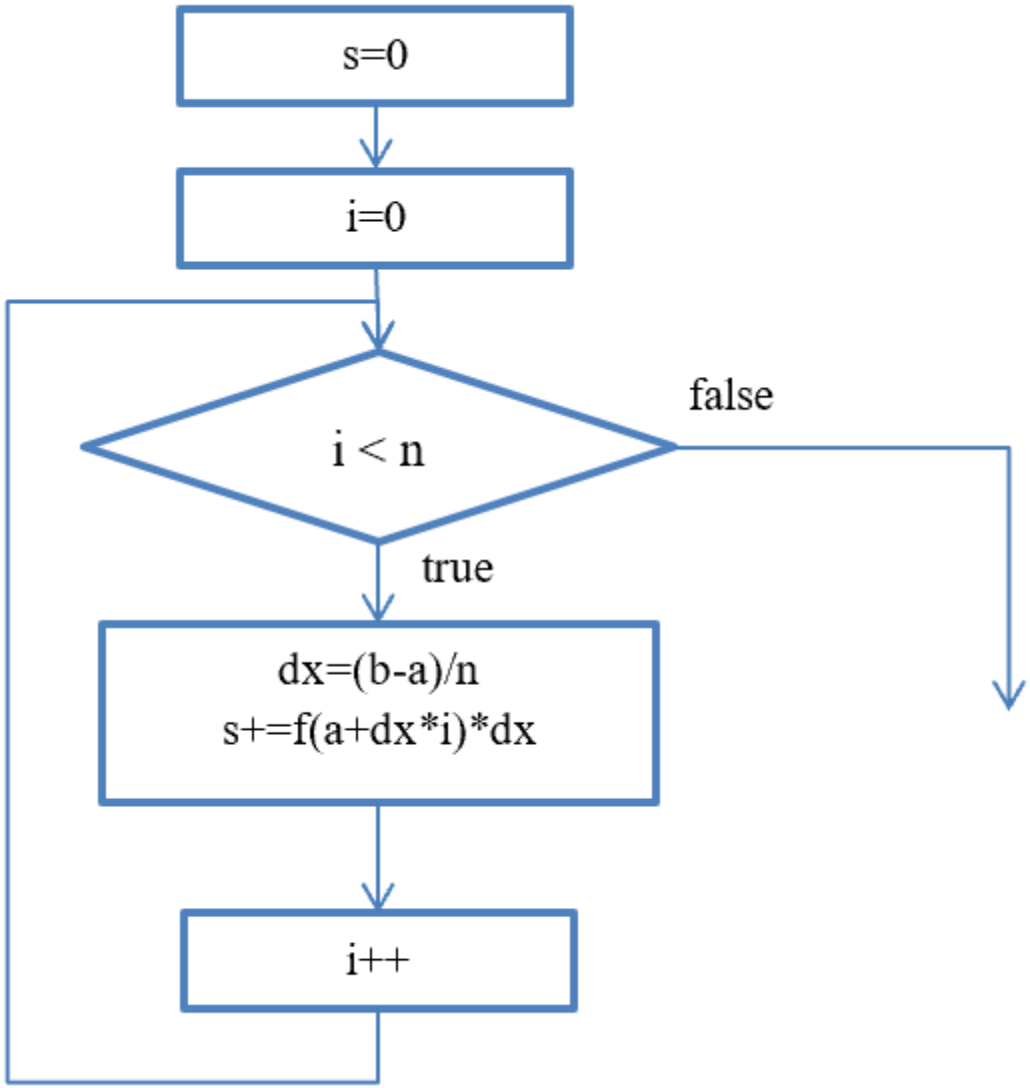
Riemann integral

Problem statement: write a program to compute integral of a function f(x) on an interval [a,b].

Algorithm: use integral definition as an area under a function *f*(x) on an interval [a,b]



$$\int_a^b f(x)dx = \lim_{n\to\infty} \sum_{i=0}^{n-1} f\left(a + \frac{b-a}{n} i\right) \frac{b-a}{n}$$


s=0
i=0
i < n
false
true
dx=(b-a)/n
s+=f(a+dx*i)*dx
i++

```c
/* compute integral of f(x) = x*x+2x+3 on [a,b] */
#include <stdio.h>

int main()
{
    int n = 100;        /* hardcoded number of Reimann sum terms */
    float a = -1.0f;    /* hardcoded [a,b] */
    float b = 1.0f;
    float s = 0.0f;             /* computed integral value */
    int i;                      /* loop counter */
    float x, y;                 /* x and y=f(x) */
    float dx = (b - a) / n;     /* width of rectangles */

    for (i = 0; i < n; i++)
    {
        x = a + dx * i;
        y = x * x + 2 * x + 3;
        s += y * dx;
    }

    printf("%f\n", s);

    return 0;
}
```