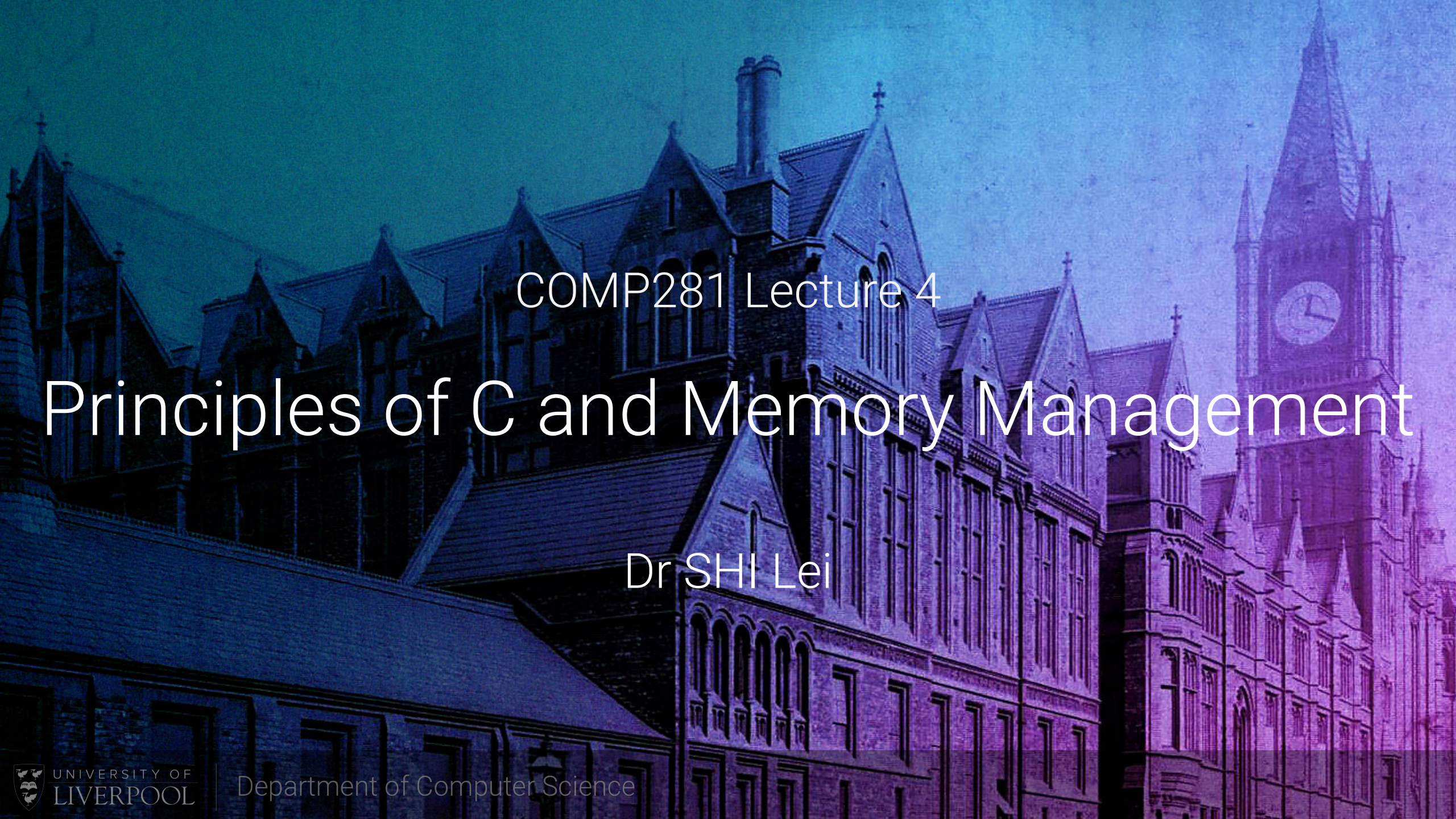


COMP281 Lecture 4

Principles of C and Memory Management

Dr SHI Lei

UNIVERSITY OF LIVERPOOL | Department of Computer Science

Last Lecture

- C Language Basics
  - Basic I/O
  - Operators
  - Decision Making

Today

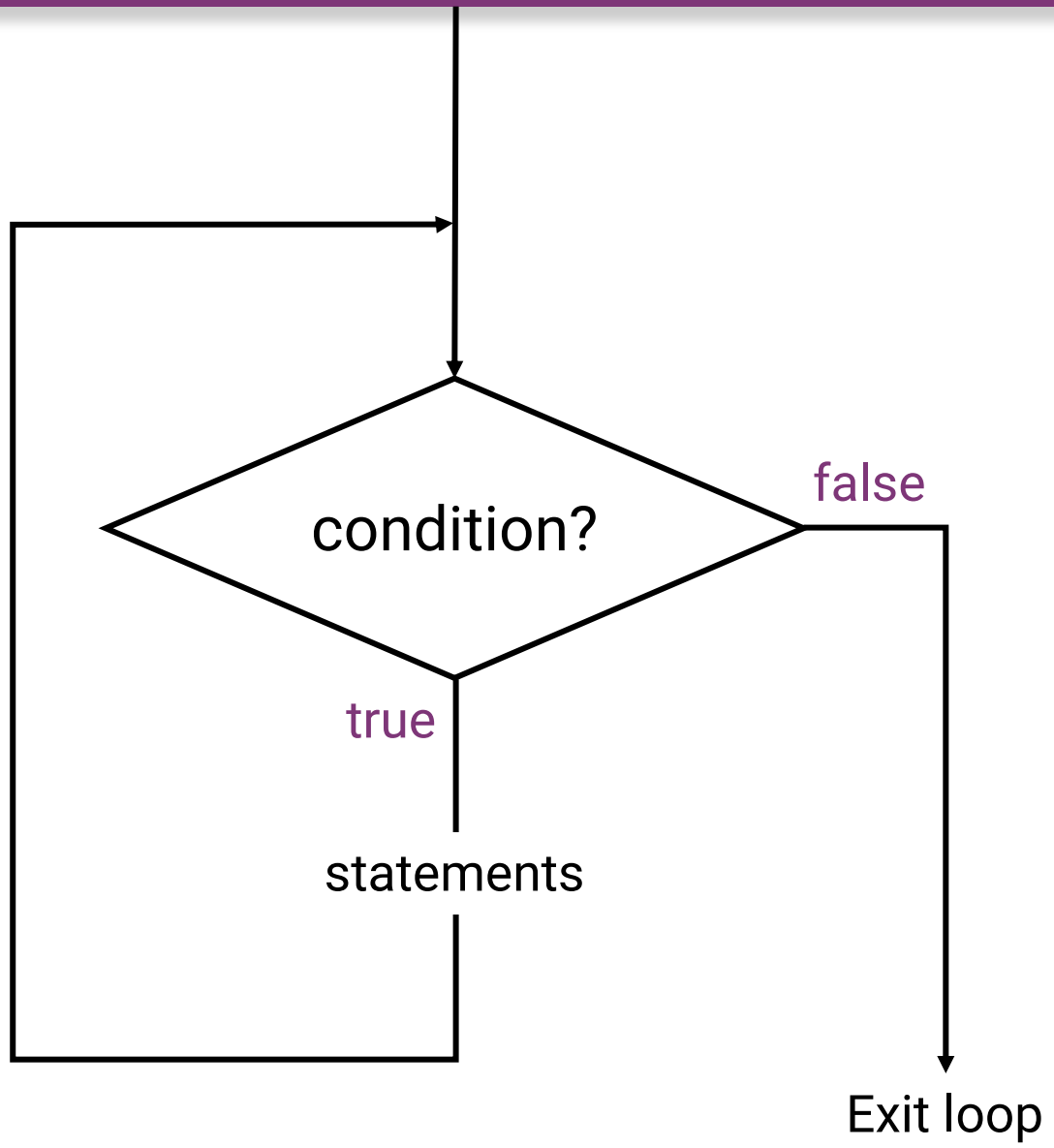
- C Language Basics
  - Basic I/O
  - Operators
  - Decision Making
  - **Loop**
- Online Judge

Loops

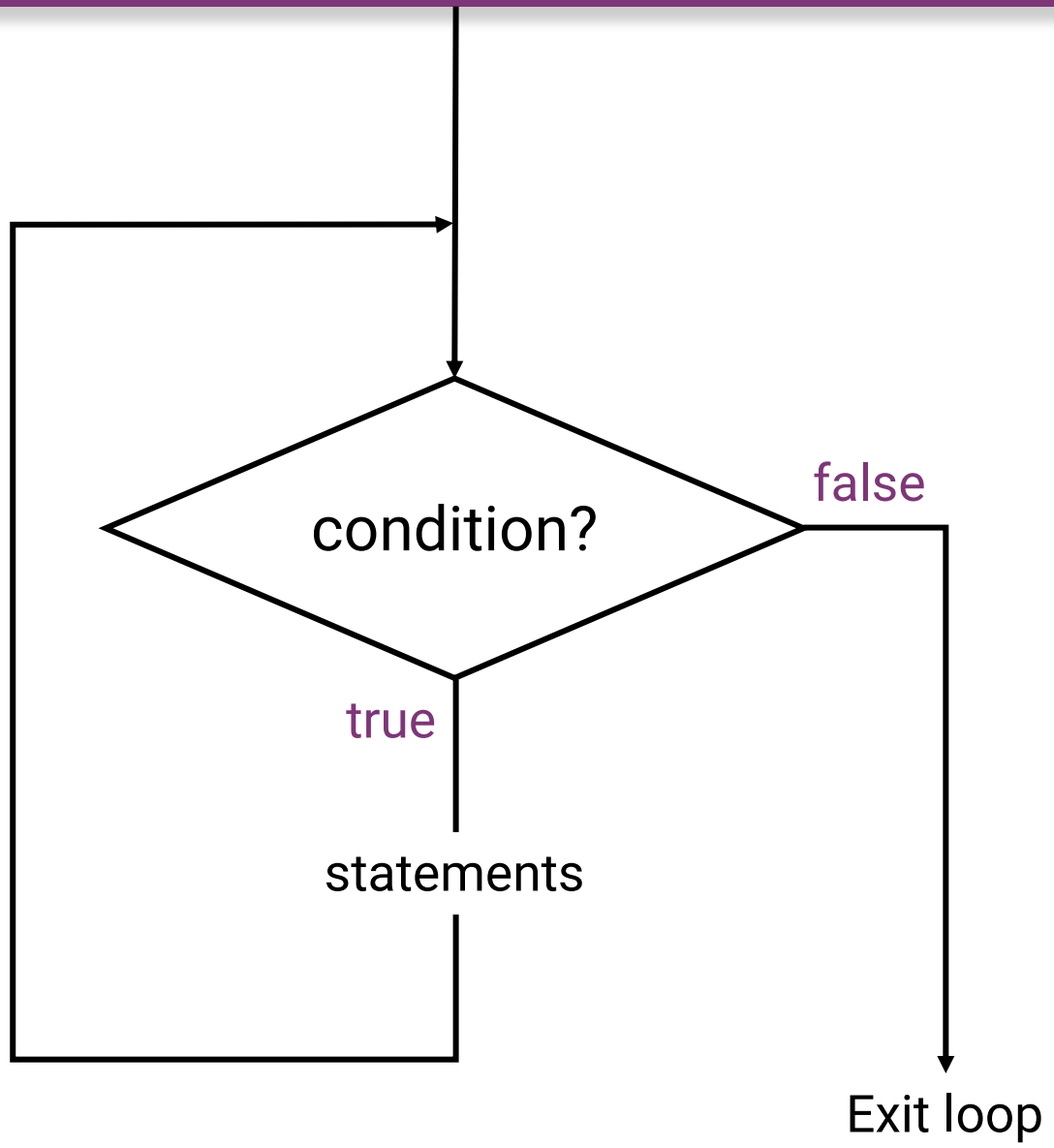
Loops

- `while loop`
- `do… while Loop`
- `for Loop`
- `continue, break, goto`

while Loop



while Loop



The `while` loop evaluates the test expression before every loop, so it can execute zero times if the condition is initially false. It requires the parenthesis like the `if`.

```
while (<expression>)
{
    <statements>
}
```

Example 1.1:

```
#include <stdio.h>
int main(void)
{
    int count = 1;
    while ( count <= 3 )
    {
        printf( "%d\n", count );
        count++;
    }
    return 0;
}
```

Text

Output

??

Example 1.1:

```
#include <stdio.h>
int main(void)
{
    int count = 1;
    while ( count <= 3 )
    {
        printf( "%d\n", count );
        count++;
    }
    return 0;
}
```

Output

```
1
2
3
```

Example 1.2:

```
#include <stdio.h>
int main(void)
{
    int count = 1;
    while ( count <= 3 )
    {
        printf( "%d\n", count );
        count++;
    }
    return 0;
}
```

Output

??

Example 1.2:

```
#include <stdio.h>
int main(void)
{
    int count = 1;
    while ( count <= 3 )
    {
        printf( "%d\n", count );
        count++;
    }
    return 0;
}
```

no exit

The program is an example of **infinite while loop**. Since the value of the variable var is same (there is no ++ or – operator used on this variable, inside the body of loop) the condition var<=3 will be true forever and the loop would never terminate.

Example 1.3:

```
#include <stdio.h>
int main(void)
{
    int count = 6;
    while ( count >= 5 )
    {
        printf( "%d", count );
        count ++;
    }
    return 0;
}
```

Output

??

Example 1.3:

```
#include <stdio.h>
int main(void)
{
    int count = 6;
    while ( count >= 5 )
    {
        printf( "%d", count );
        count ++;
    }
    return 0;
}
```

Infinite loop: count will always have value >=5 so the loop would never end.

Example 1.4:

```
#include <stdio.h>
int main(void)
{
    int count = 5;
    while ( count <= 6 )
    {
        printf( "%d", count );
        count --;
    }
    return 0;
}
```

Output

??

Example 1.4:

```
#include <stdio.h>
int main(void)
{
    int count = 5;
    while ( count <= 6 )
    {
        printf( "%d", count );
        count --;
    }
    return 0;
}
```

Infinite loop: count value will keep decreasing because of -- operator, hence it will always be <= 6.

Use of Logical operators in `while` loop

```
while( num1 <= 10 && num2 <= 10 )
```

```
while( num1 <= 10 || num2 <= 10 )
```

```
while( num1 != num2 && num1 <= num2 )
```

```
while( num1 != 10 || num2 >= num1 )
```

Example 1.5

```
#include <stdio.h>
int main(void)
{
    int i=1, j=1;
    while ( i <= 4 || j <= 3 )
    {
        printf( "%d %d\n", i, j );
        i++;
        j++;
    }
    return 0;
}
```

Output

??

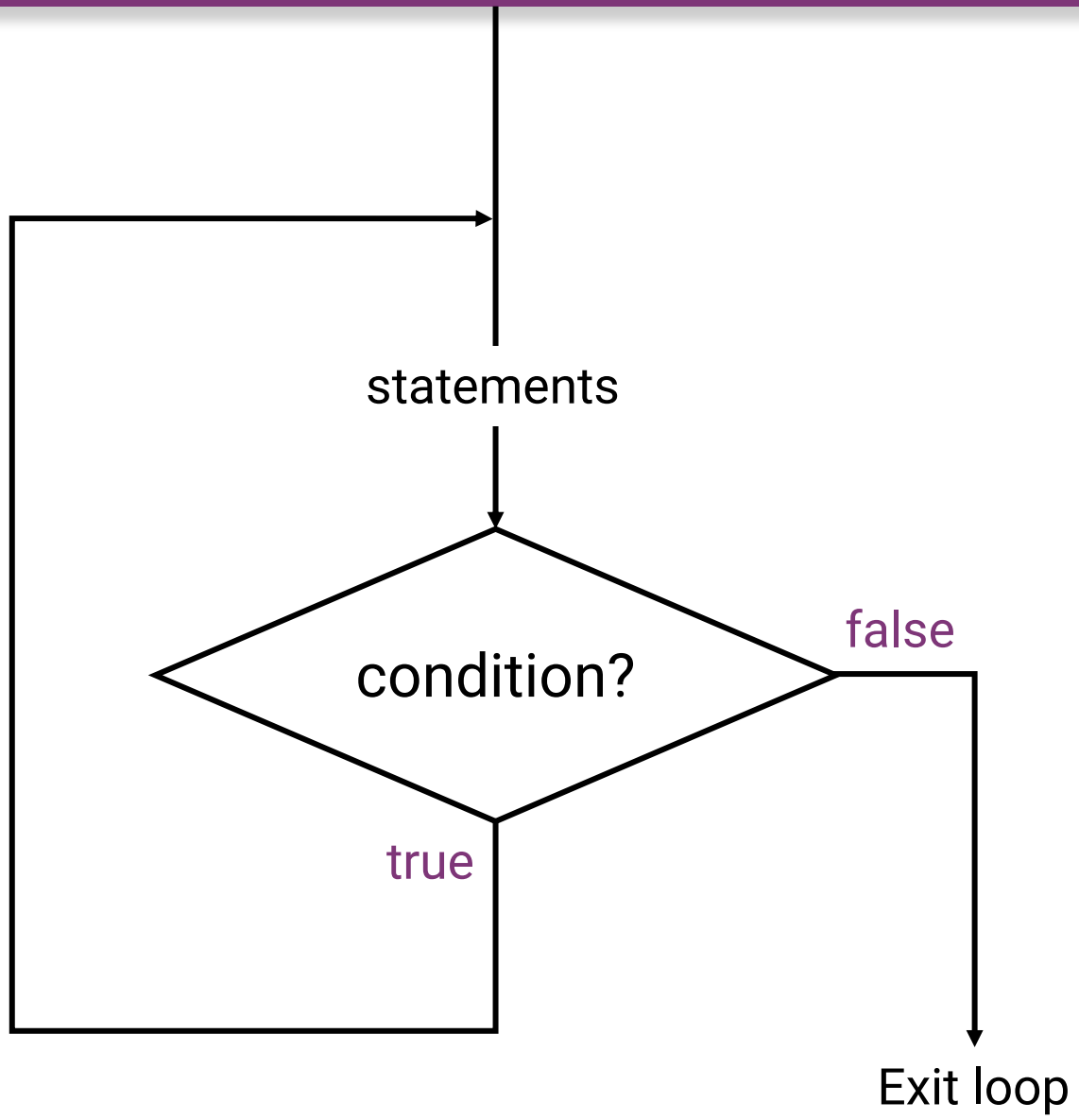
Example 1.5

```
#include <stdio.h>
int main(void)
{
    int i=1, j=1;
    while ( i <= 4 || j <= 3 )
    {
        printf( "%d %d\n", i, j );
        i++;
        j++;
    }
    return 0;
}
```

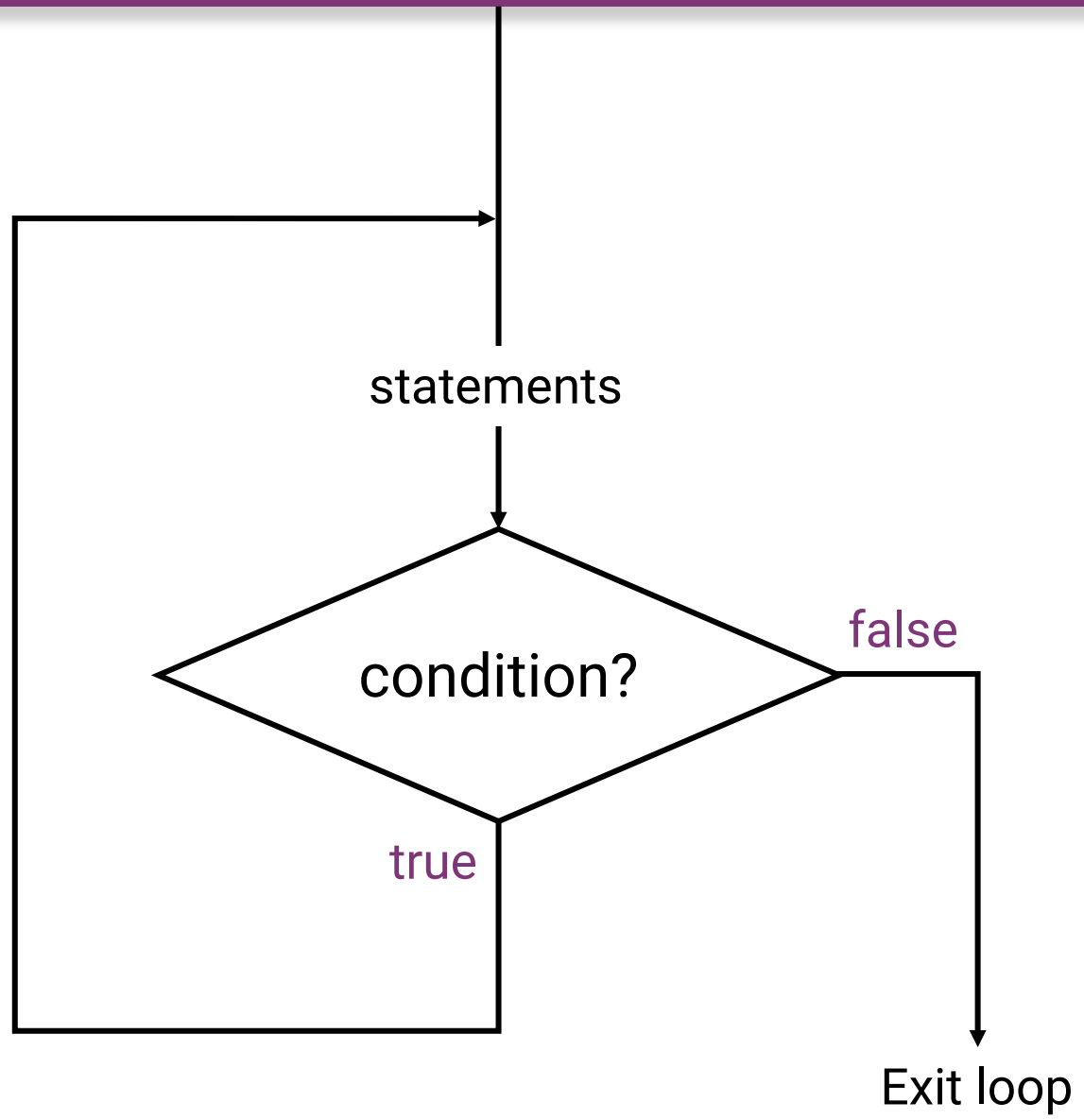
Output

```
1 1
2 2
3 3
4 4
```

do… while Loop



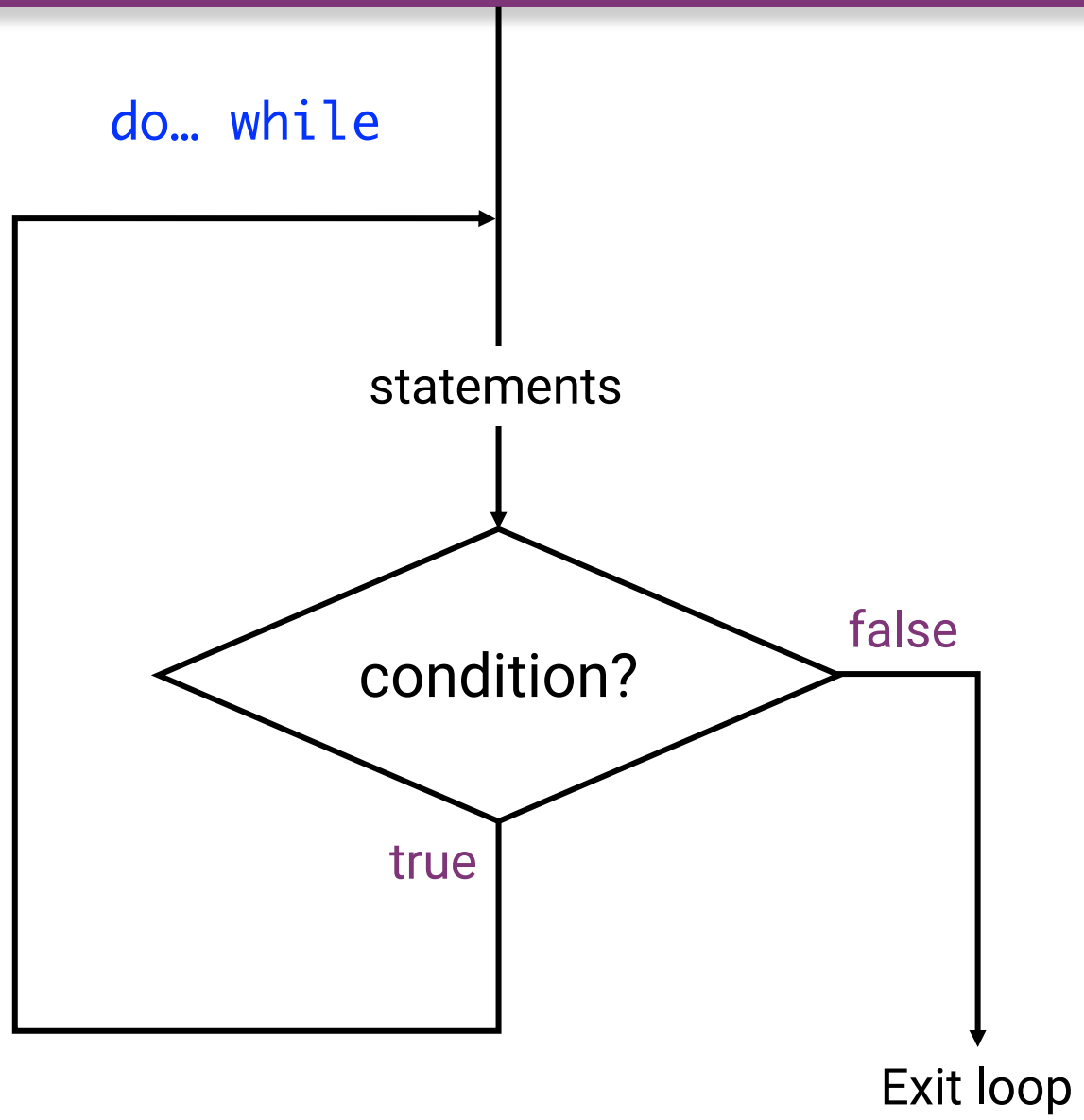
do… while Loop

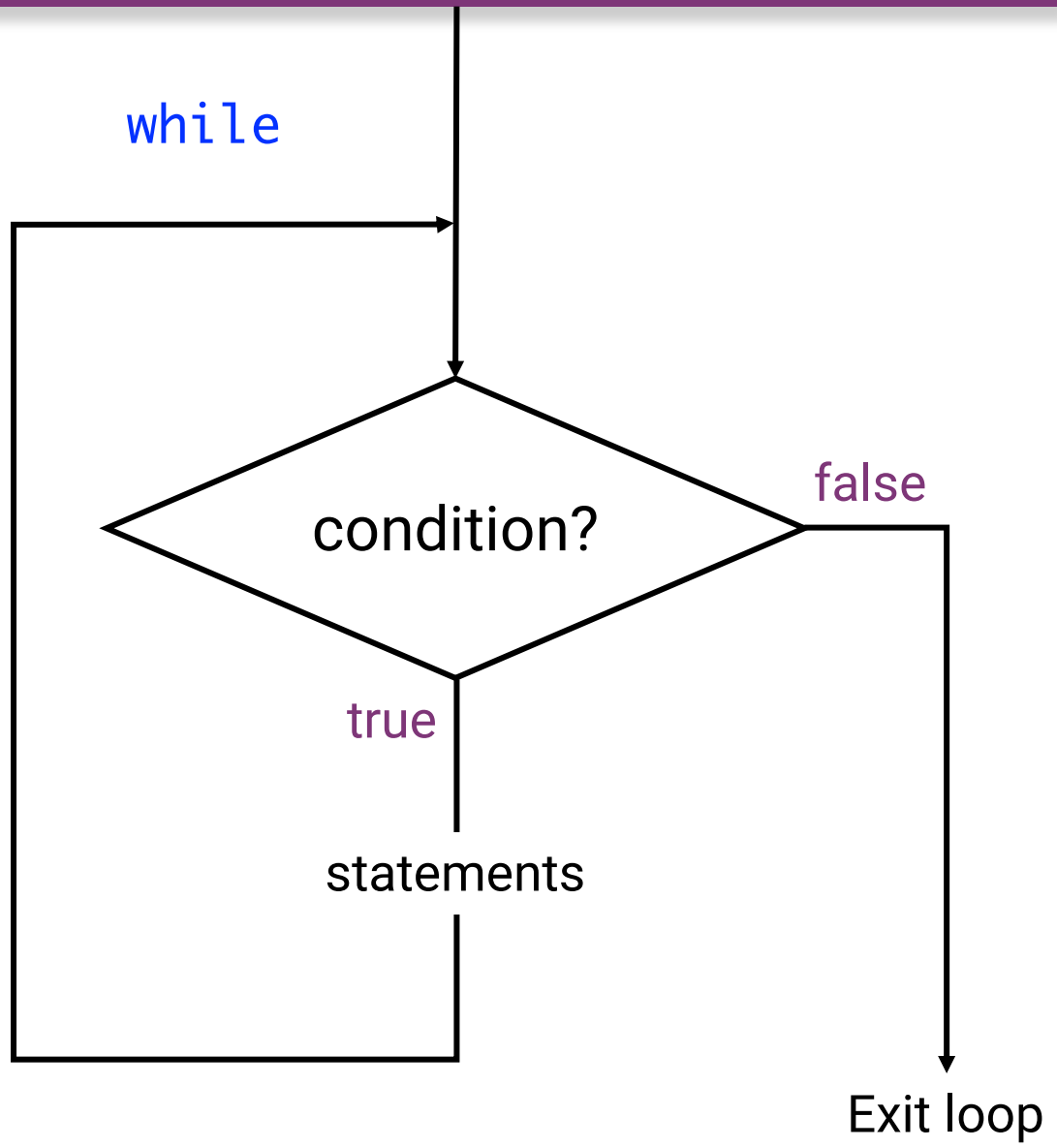


Like a `while`, but with the test condition at the bottom of the loop. The loop body will always execute at least once. The `do-while` is an unpopular area of the language, most everyone tries to use the straight `while` if at all possible.

```
do
{
    <statements>
} while (<expression>)
```

do… while vs while





Example 2:

```c
#include <stdio.h>
int main(void)
{
    int i = 0;
    do
    {
        printf( "%d\n", i);
        i++;
    } while( i<=3 );
    return 0;
}
```

Output

??

Example 2:

```
#include <stdio.h>
int main(void)
{
    int i = 0;
    do
    {
        printf( "%d\n", i);
        i++;
    } while( i<=3 );
    return 0;
}
```

Output

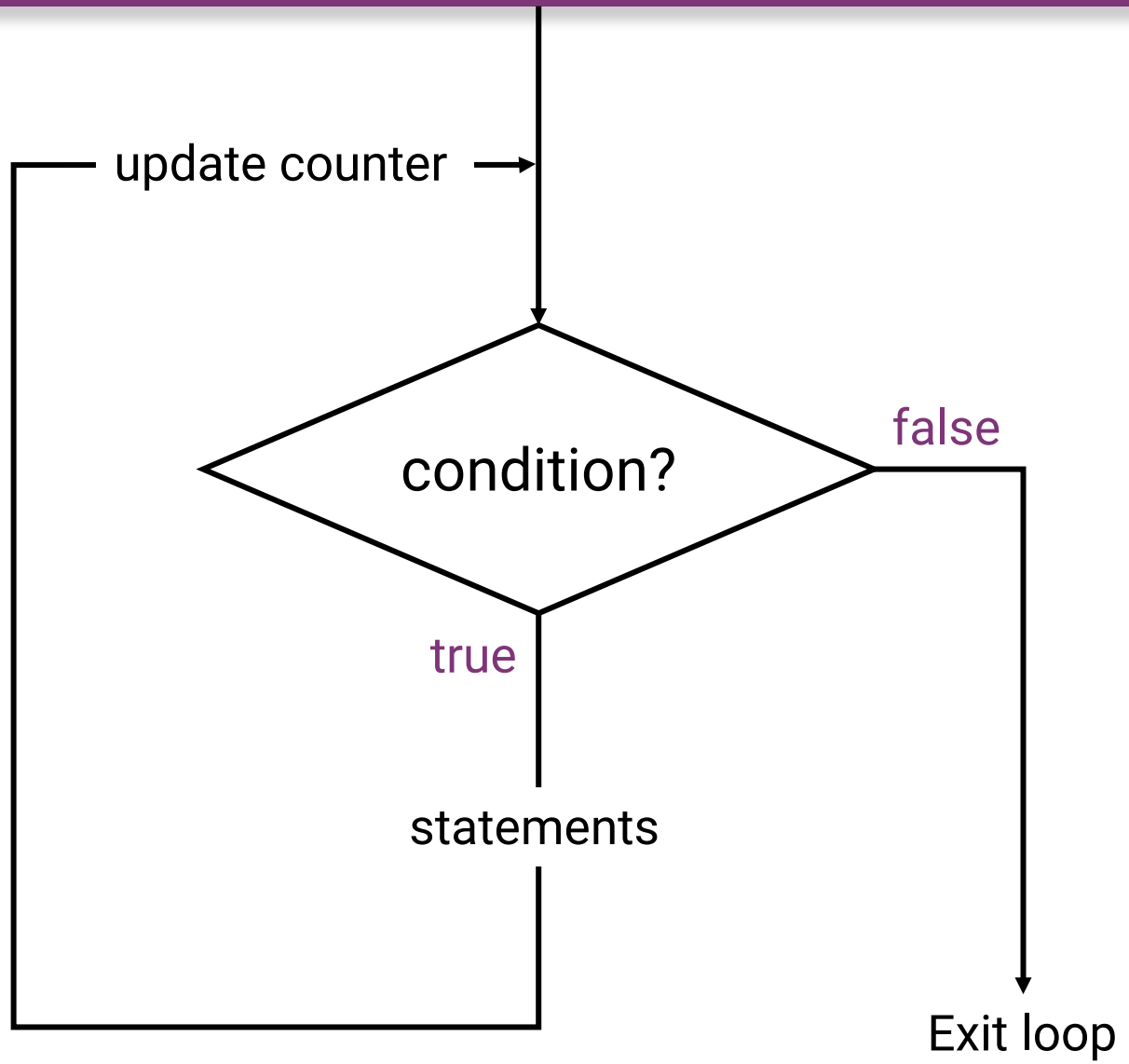
0

1

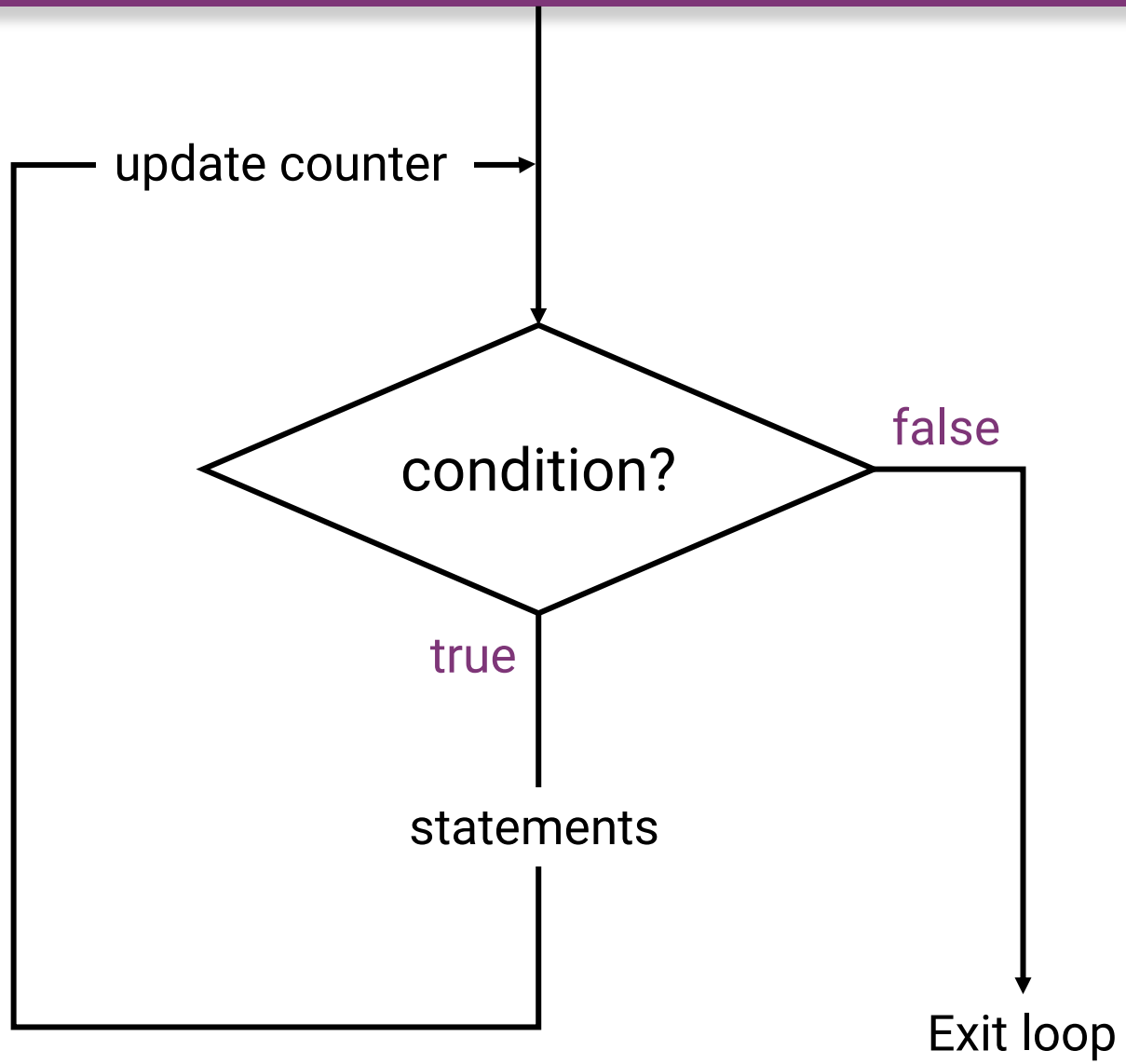
2

3

for Loop



`for` Loop



The `for` loop is the most general looping construct.

The loop header contains three parts: an ***initialisation***, a ***continuation condition***, and an ***action***.

```
for (<initialisation>;<continuation>;<action>)
{
    <statements>
}
```

The ***initialisation*** is executed once before the body of the loop is entered. The loop continues to run as long as the ***continuation condition*** remains true (like a `while`).

After every execution of the loop, the ***action*** is executed.

Example 3.1: favorite

```
#include <stdio.h>
int main(void)
{
    int i;
    for ( i=1; i<=3; i++ )
    {
        printf( "%d\n", i );
    }
    return 0;
}
```

Output

??

Example 3.1:

```
#include <stdio.h>
int main(void)
{
    int i;
    for ( i=1; i<=3; i++ )
    {
        printf( "%d\n", i );
    }
    return 0;
}
```

Output

```
1
2
3
```

Various forms of `for` loop:

Text

1.

```
for ( num=10; num<20; num=num+1 )
```

2.

```
int num=10;
for( ; num<20; num++ )
```

3.

```
for( num=10 ; num<20; )
{
    // statements
    num++;
}
```

4.

```
int num=10;
for( ; num<20; )
{
    //Statements
    num++;
}
```

5.

```
for( num=20; num>10; num--)
```

Example 3.2 – Nested `for` Loop:

```
#include<stdio.h>
int main(void)
{
    for( int i=0; i<2; i++)
    {
        for( int j=0; j<4; j++)
        {
            printf( "%d,%d\n", i, j );
        }
    }
    return 0;
}
```

Output

??

very important , memory management———rows and colunms, element of the matrix

Example 3.2 – Nested `for` Loop:

```
#include<stdio.h>
int main(void)
{
    for( int i=0; i<2; i++)
    {
        for( int j=0; j<4; j++)
        {
            printf( "%d,%d\n", i, j );
        }
    }
    return 0;
}
```

sort the element in special order

Output

0,0

0,1

0,2

0,3

1,0

1,1

1,2

1,3

Example 3.3 – Multiple initialisation inside `for` Loop:

```
#include<stdio.h>
int main(void)
{
    int i, j;
    for( i=1, j=1; i<3 || j<5; i++, j++ )
    {
        printf("%d, %d\n",i,j);
    }
    return 0;
}
```

Output

??

Example 3.3 – Multiple initialisation inside `for` Loop:

```
#include<stdio.h>
int main(void)
{
    int i, j;
    for( i=1, j=1; i<3 || j<5; i++, j++ )
    {
        printf("%d, %d\n",i,j);
    }
    return 0;
}
```

Output

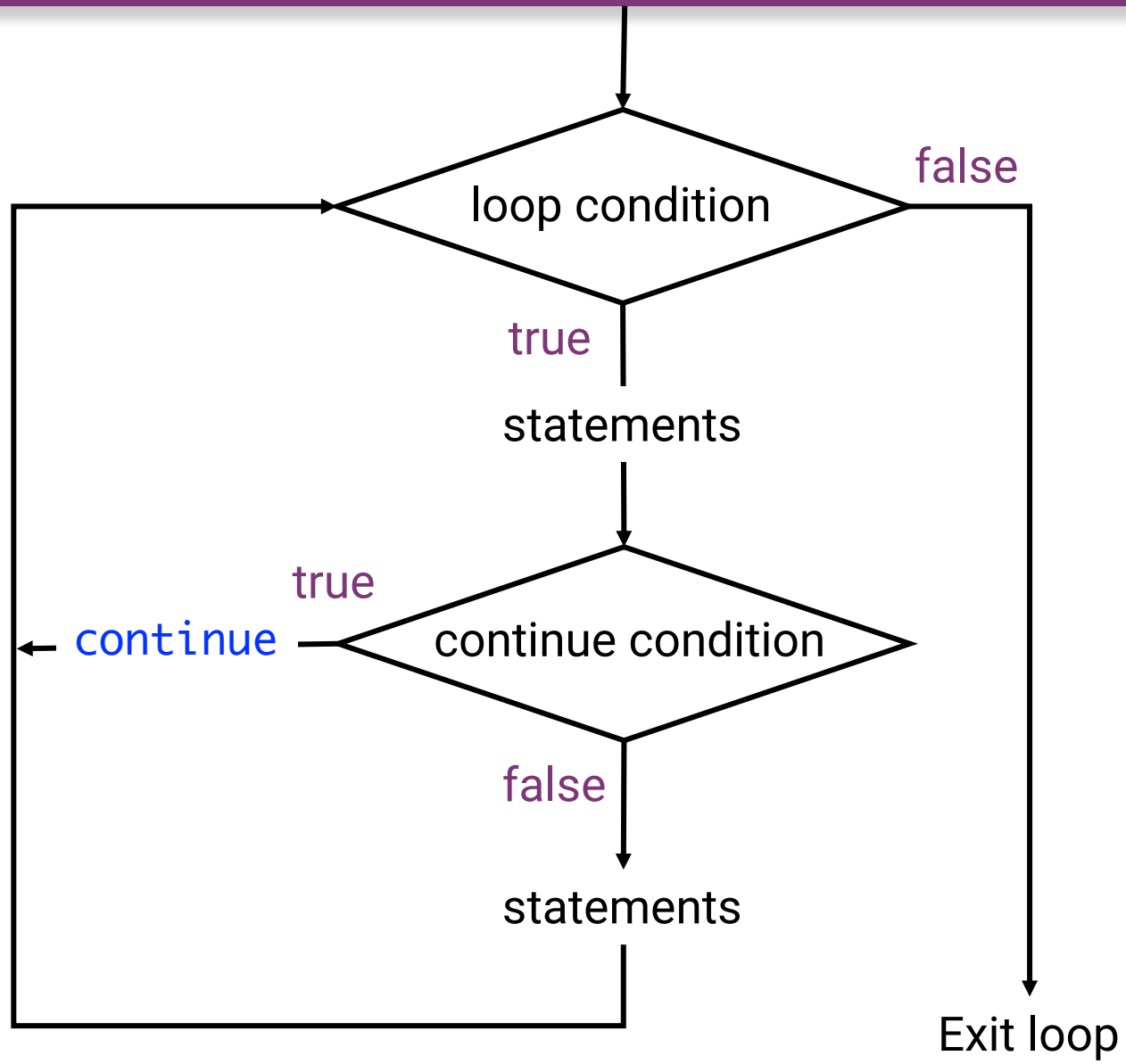
1, 1

2, 2

3, 3

4, 4

continue



The `continue` **statement** is used inside **loops**. When a continue statement is encountered inside a loop, control jumps to the beginning of the loop for next iteration, skipping the execution of statements inside the body of loop for the current iteration.

More efficient

Example 4.1 – `continue` statement inside `for` Loop:

```c
#include <stdio.h>
int main(void)
{
    for (int j=0; j<=8; j++)
    {
        if (j==4)
        {
            /* The continue statement is encountered when
             * the value of j is equal to 4.*/
            continue;
       }
       /* This print statement would not execute for the
        * loop iteration where j ==4  because in that case
        * this statement would be skipped.
        */
       printf("%d\n", j);
   }
   return 0;
}
```

Output

??

Example 4.1 – `continue` statement inside `for` Loop:

```c
#include <stdio.h>
int main(void)
{
    for (int j=0; j<=8; j++)
    {
        if (j==4)
        {
            /* The continue statement is encountered when
             * the value of j is equal to 4.*/
            continue;
       }
       /* This print statement would not execute for the
        * loop iteration where j ==4  because in that case
        * this statement would be skipped.
        */
       printf("%d\n", j);
   }
   return 0;
}
```

Output

```
0
1
2
3
5
6
7
8
```

Example 4.2 – `continue` statement inside `while` Loop:

```
#include <stdio.h>
int main(void)
{
    int counter = 10;
    while (counter >= 0)
    {
        if (counter == 7)
        {
            counter--;
            continue;
        }
        printf("%d\n", counter);
        counter--;
    }
    return 0;
}
```

Output

??

Example 4.2 – `continue` statement inside `while` Loop:

```
#include <stdio.h>
int main(void)
{
    int counter = 10;
    while (counter >= 0)
    {
        if (counter == 7)
        {
            counter--;
            continue;
        }
        printf("%d\n", counter);
        counter--;
    }
    return 0;
}
```

Output

```
10
9
8
6
5
4
3
2
1
0
```

The print statement is skipped when counter value was 7.

Example 4.3 – `continue` statement inside `do...while` Loop:

```
#include <stdio.h>
int main(void)
{
    int j = 0;
    do
    {
        if ( j == 7 )
        {
            j++;
            continue;
        }
        printf("%d\n", j);
        j++;
    }while( j < 10 );
    return 0;
}
```

Output

??

Example 4.3 – `continue` statement inside `do…while` Loop:

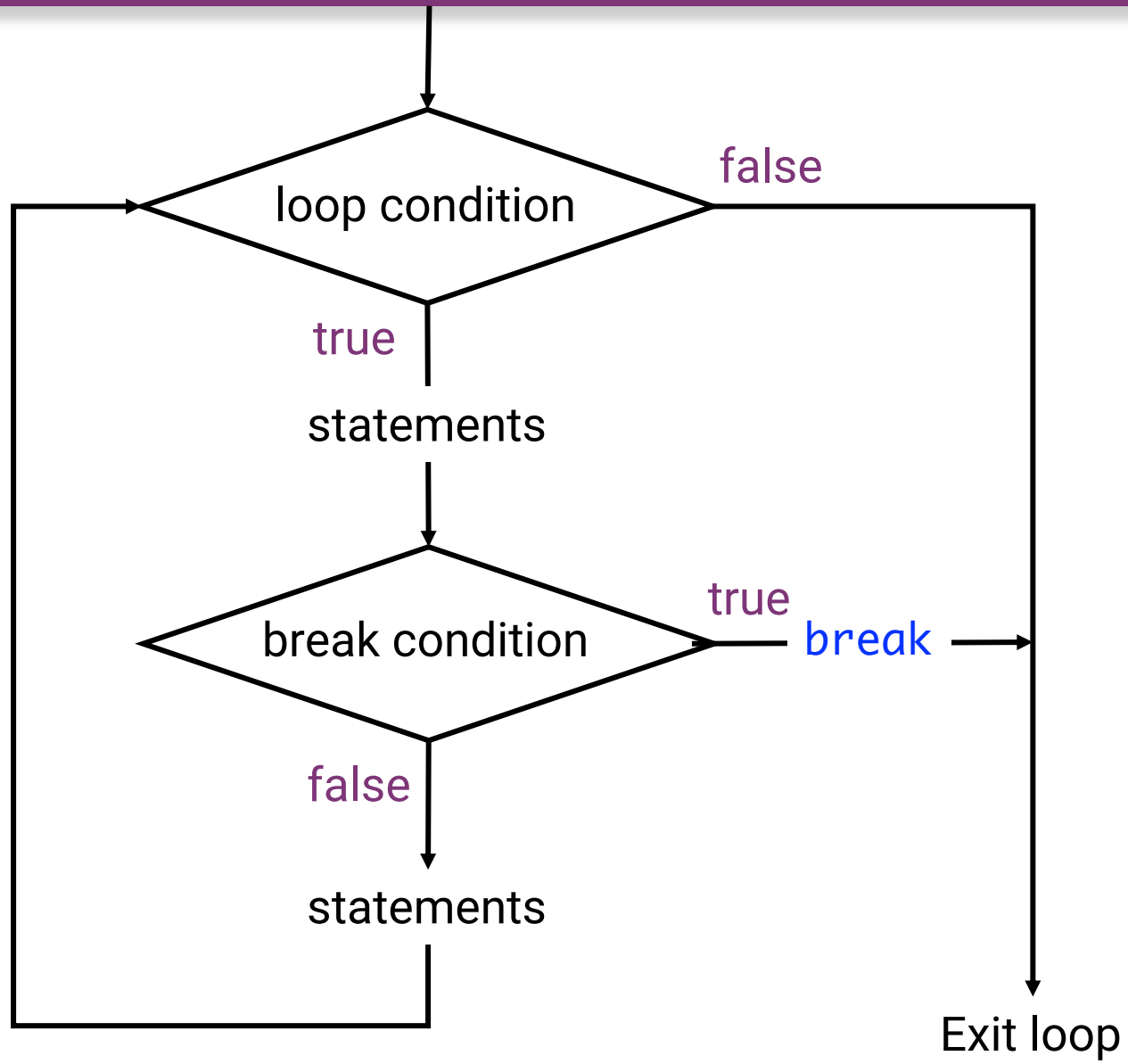
```c
#include <stdio.h>
int main(void)
{
    int j = 0;
    do
    {
        if ( j == 7 )
        {
            j++;
            continue;
        }
        printf("%d\n", j);
        j++;
    }while( j < 10 );
    return 0;
}
```

Output

```
0
1
2
3
4
5
6
8
9
```

The print statement is skipped when counter value was 7.

break



- It is used to come out of the **loop** *instantly*. When a **break** statement is encountered inside a loop, the control directly comes out of loop and the loop gets terminated. It is used with **if** statement whenever used inside loop.
- This can also be used in **switch case control structure**. Whenever it is encountered in switch-case block, the control comes out of the switch-case.

Example 5.1 – Use of break in a `while` Loop:

```
#include <stdio.h>
int main(void)
{
    int num =0;
    while( num <= 100)
    {
        printf("%d\n", num);
        if (num==2)
        {
            break;
        }
        num++;
    }
    printf("Out of while-loop");
    return 0;
}
```

Output

??

Example 5.1 – Use of break in a `while` Loop:

```
#include <stdio.h>
int main(void)
{
    int num =0;
    while( num <= 100)
    {
        printf("%d\n", num);
        if (num==2)
        {
            break;
        }
        num++;
    }
    printf("Out of while-loop");
    return 0;
}
```

Output

```
0

1

2


Out of while-loop
```

Example 5.2 – Use of break in a `for` Loop:

```
#include <stdio.h>
int main(void)
{
    int var;
    for ( var = 100; var >= 10; var -- )
    {
        printf("%d\n", var);
        if ( var == 99 )
        {
            break;
        }
    }
    printf("Out of for-loop");
    return 0;
}
```

Output

??

Example 5.2 – Use of break in a `for` Loop:

```
#include <stdio.h>
int main(void)
{
    int var;
    for ( var = 100; var >= 10; var -- )
    {
        printf("%d\n", var);
        if ( var == 99 )
        {
            break;
        }
    }
    printf("Out of for-loop");
    return 0;
}
```

Output

```
100
99


Out of for-loop
```

Example 5.3 – Use of break in a `switch-case`:

```
int main(void)
{
    int num;
    printf("Enter value of num: ");
    scanf("%d",&num);
    switch(num)
    {
        case 1:
            printf("You have entered value 1.\n");
            break;
        case 2:
            printf("You have entered value 2.\n");
            break;
        case 3:
            printf("You have entered value 3.\n");
            break;
        default:
            printf("Input value is other than 1, 2 & 3.\n");
    }
    return 0;
}
```

Output

Enter value of num: 2

??

Example 5.3 – Use of break in a `switch-case`:

```
int main(void)
{
    int num;
    printf("Enter value of num: ");
    scanf("%d",&num);
    switch(num)
    {
        case 1:
            printf("You have entered value 1.\n");
            break;
        case 2:
            printf("You have entered value 2.\n");
            break;
        case 3:
            printf("You have entered value 3.\n");
            break;
        default:
            printf("Input value is other than 1, 2 & 3.\n");
    }
    return 0;
}
```

Output

Enter value of num: 2

You have entered value 2.

Example 5.3 – Use of break in a `switch-case`:

```
int main(void)
{
    int num;
    printf("Enter value of num: ");
    scanf("%d",&num);
    switch(num)
    {
        case 1:
            printf("You have entered value 1.\n");
            break;
        case 2:
            printf("You have entered value 2.\n");
            break;
        case 3:
            printf("You have entered value 3.\n");
            break;
        default:
            printf("Input value is other than 1, 2 & 3.\n");
    }
    return 0;
}
```

Output

Enter value of num: 999

??

Example 5.3 – Use of break in a `switch-case`:

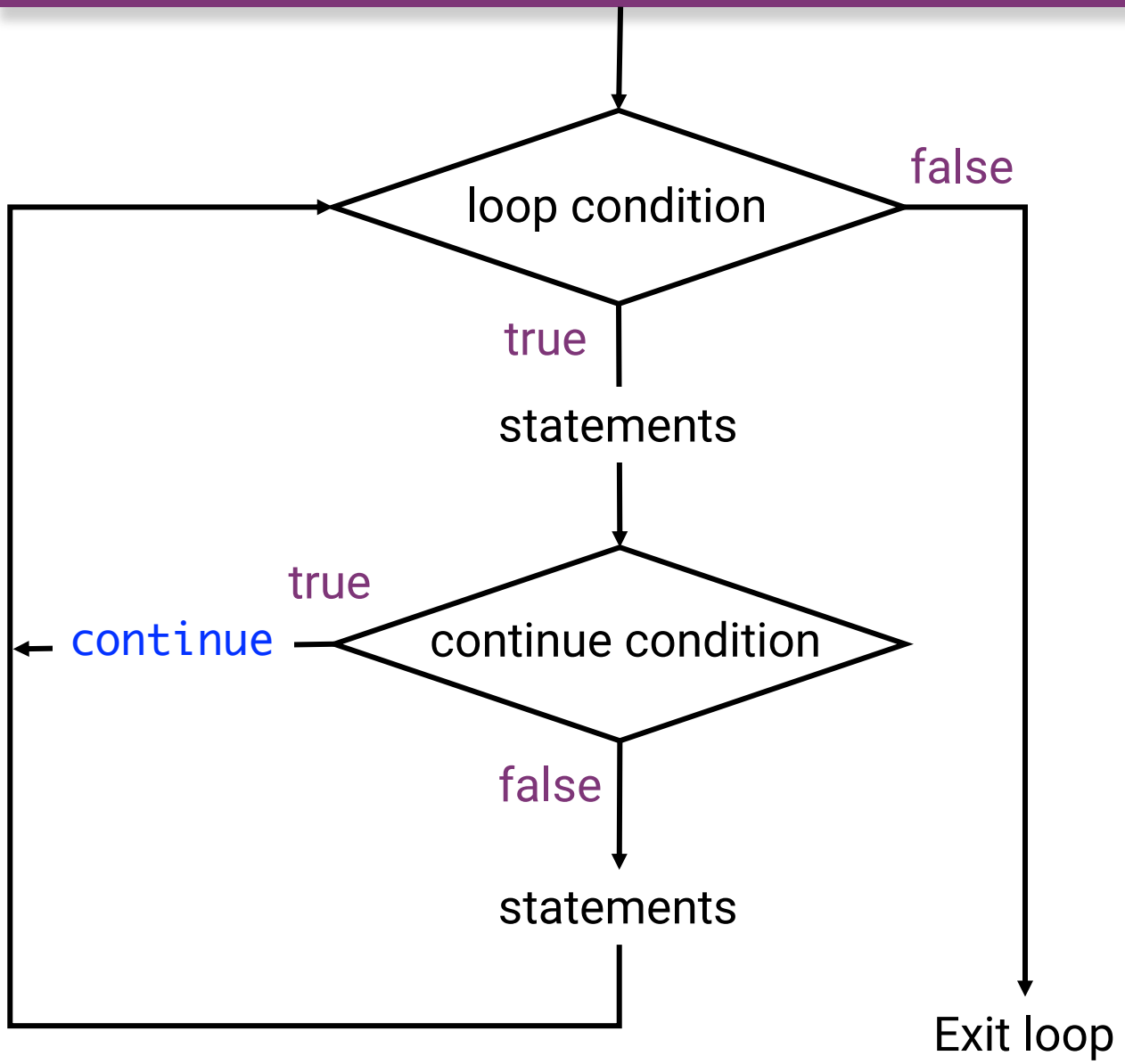
```
int main(void)
{
    int num;
    printf("Enter value of num: ");
    scanf("%d",&num);
    switch(num)
    {
        case 1:
            printf("You have entered value 1.\n");
            break;
        case 2:
            printf("You have entered value 2.\n");
            break;
        case 3:
            printf("You have entered value 3.\n");
            break;
        default:
            printf("Input value is other than 1, 2 & 3.\n");
    }
    return 0;
}
```

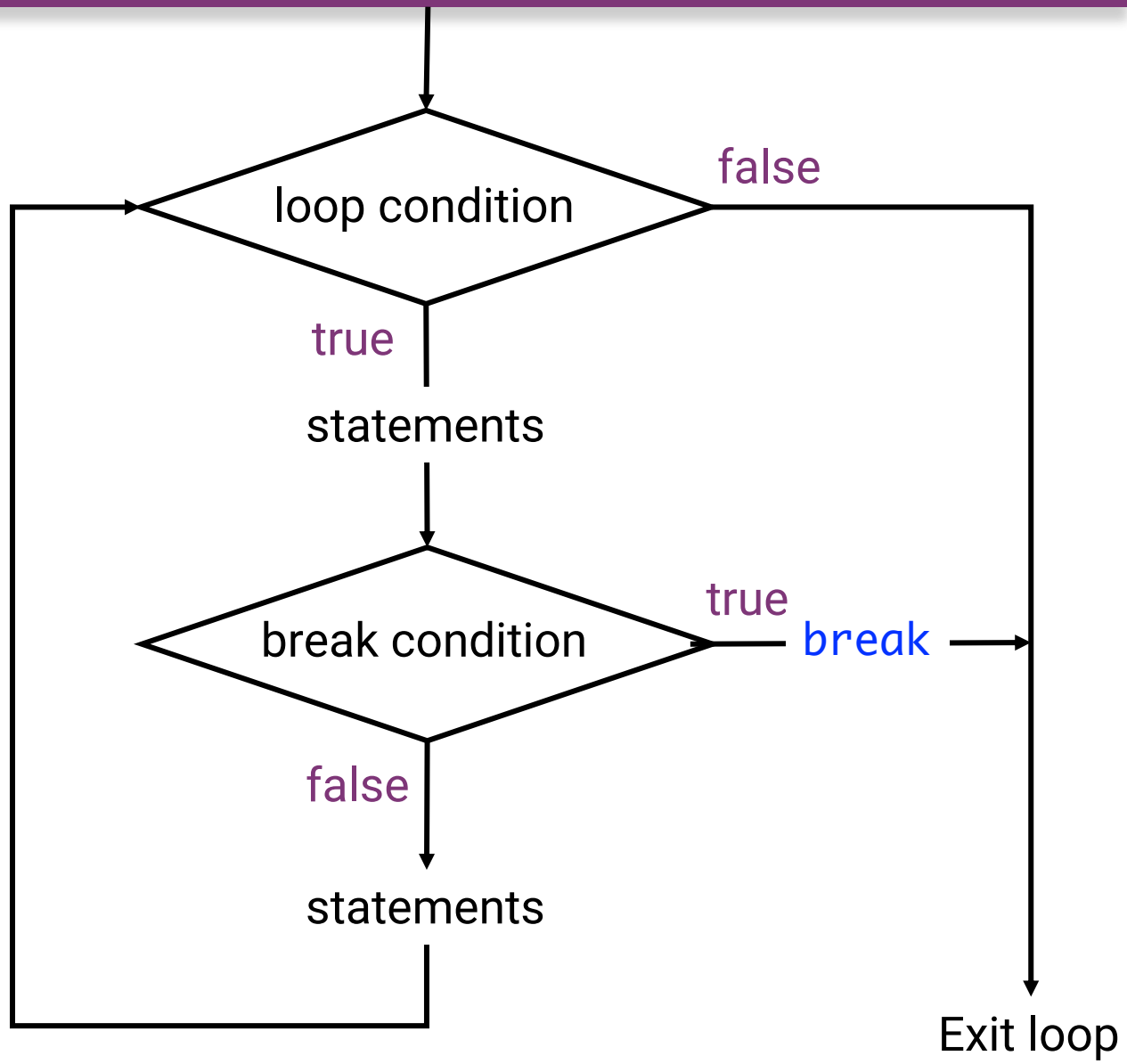
Output

Enter value of num: 999

Input value is other than 1, 2 & 3

continue vs break





goto

```
goto <label_name>;
…
…
<label_name>: <statements>
```

go some where

normally do not use it

- When a `goto` statement is encountered, the control **jumps directly** to the label mentioned in the goto statement.
- The goto statement is rarely used because it makes program confusing, less readable and complex. Also, when this is used, the control of the program won't be easy to trace, hence it makes testing and debugging difficult.

Example 6 – Use goto:

```
#include <stdio.h>
int main(void)
{
    int sum=0;
    for( int i = 0; i <= 10; i++ )
    {
        sum = sum + i;
        if( i==5 ){
            goto addition;
        }
    }
    addition:
        printf( "%d\n", sum );
    return 0;
}
```

exit the whole loop

Output

??

Example 6 – Use goto:

```
#include <stdio.h>
int main(void)
{
    int sum=0;
    for( int i = 0; i <= 10; i++ )
    {
        sum = sum + i;
        if( i==5 ){
            goto addition;
        }
    }
    addition:
    printf( "%d\n", sum );
    return 0;
}
```

Output

15

Example 6 – Use goto:

```
#include <stdio.h>
int main(void)
{
    int sum=0;
    for( int i = 0; i <= 10; i++ )
    {
        sum = sum + i;
        if( i==5 ){
            goto addition;
        }
    }
    addition:
        printf( "%d\n", sum );
    return 0;
}
```

Output

15

In this example, we have a label addition and when the value of i (inside loop) is equal to 5 then we are jumping to this label using goto. This is reason the sum is displaying the sum of numbers till 5 even though the loop is set to run from 0 to 10.

Online Judge

Online Judge

- OJ -

- Programming exercises
- Assignment submissions

https://student.csc.liv.ac.uk/JudgeOnline

COMP281

Logout

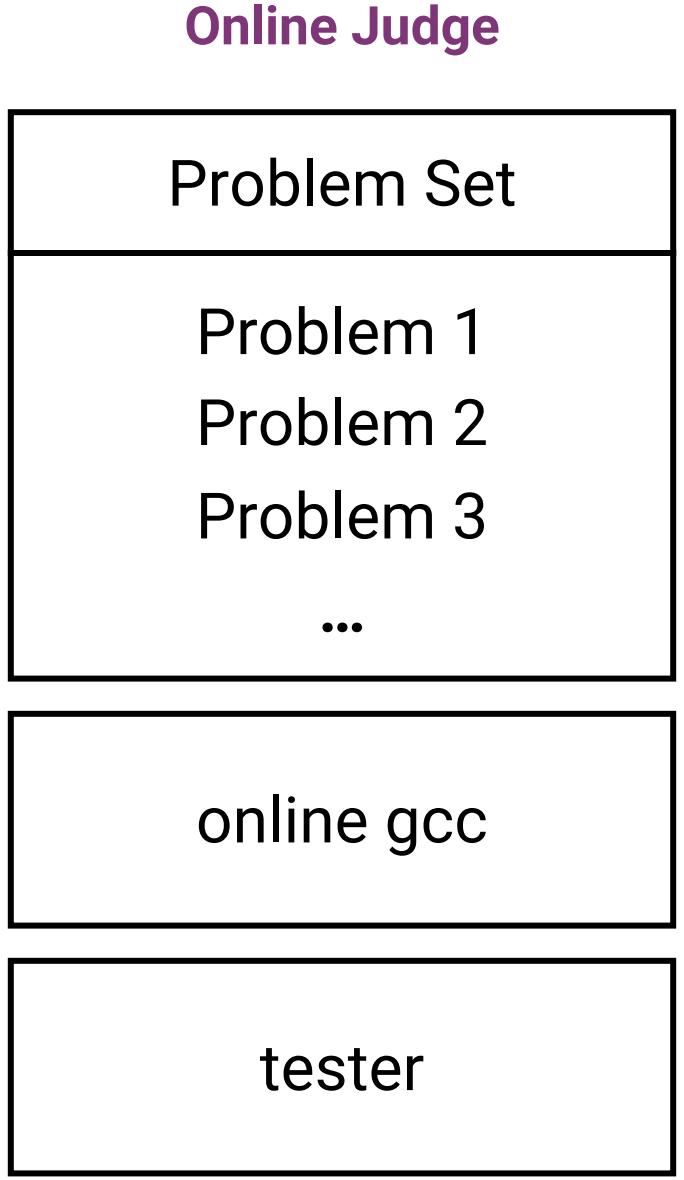
Home | ProblemSet | Status | Add New Problem

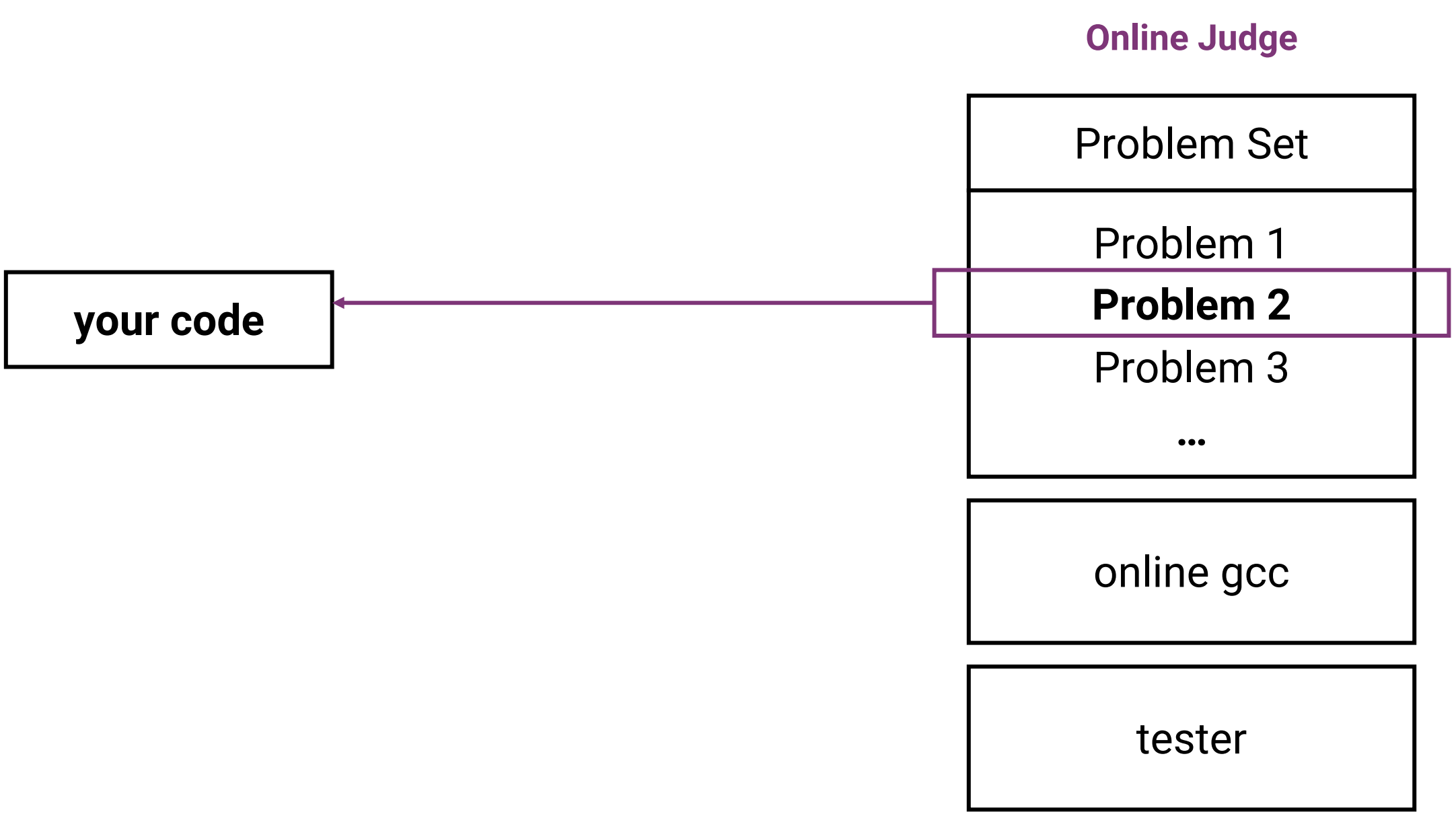
Problem ID Go

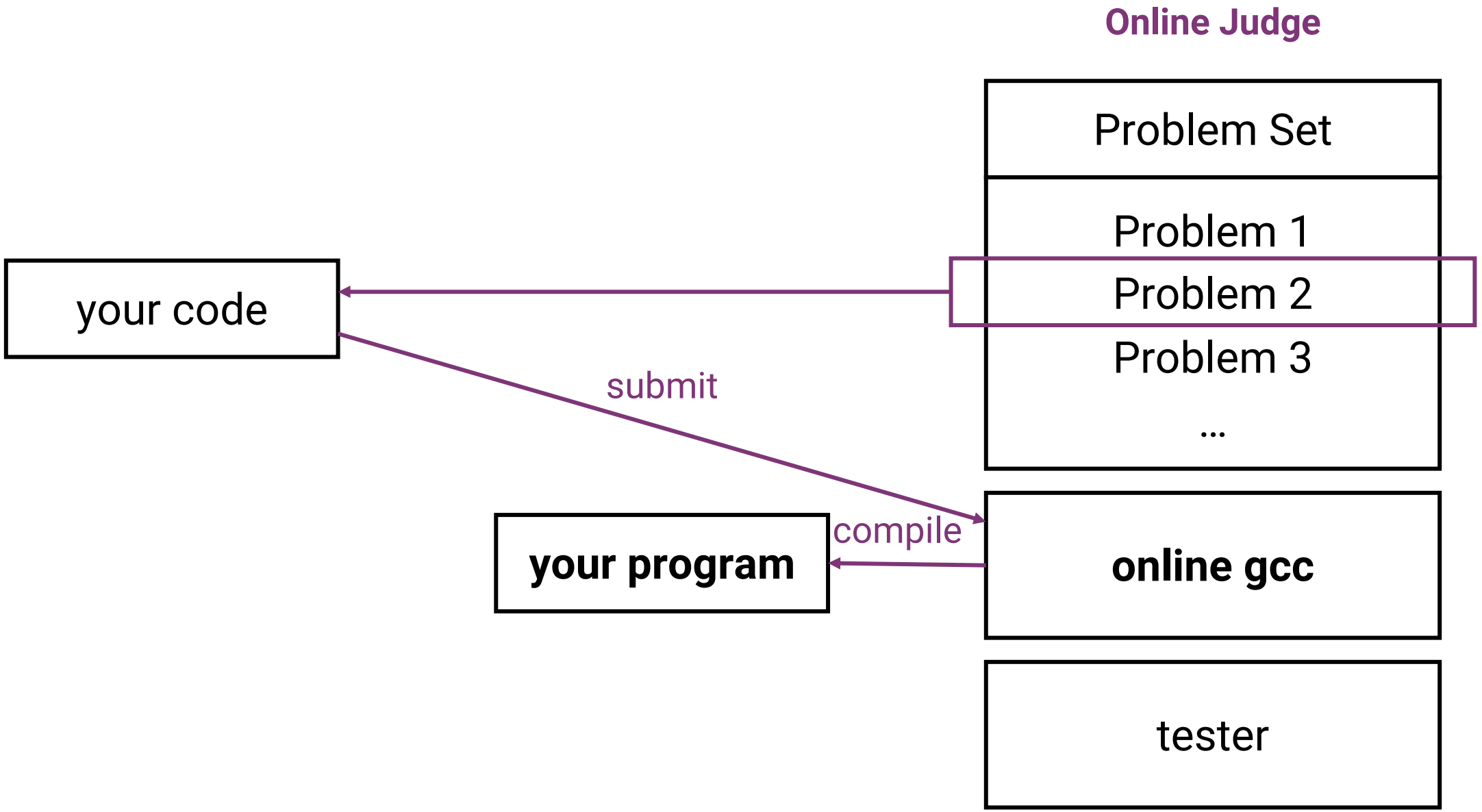
Search

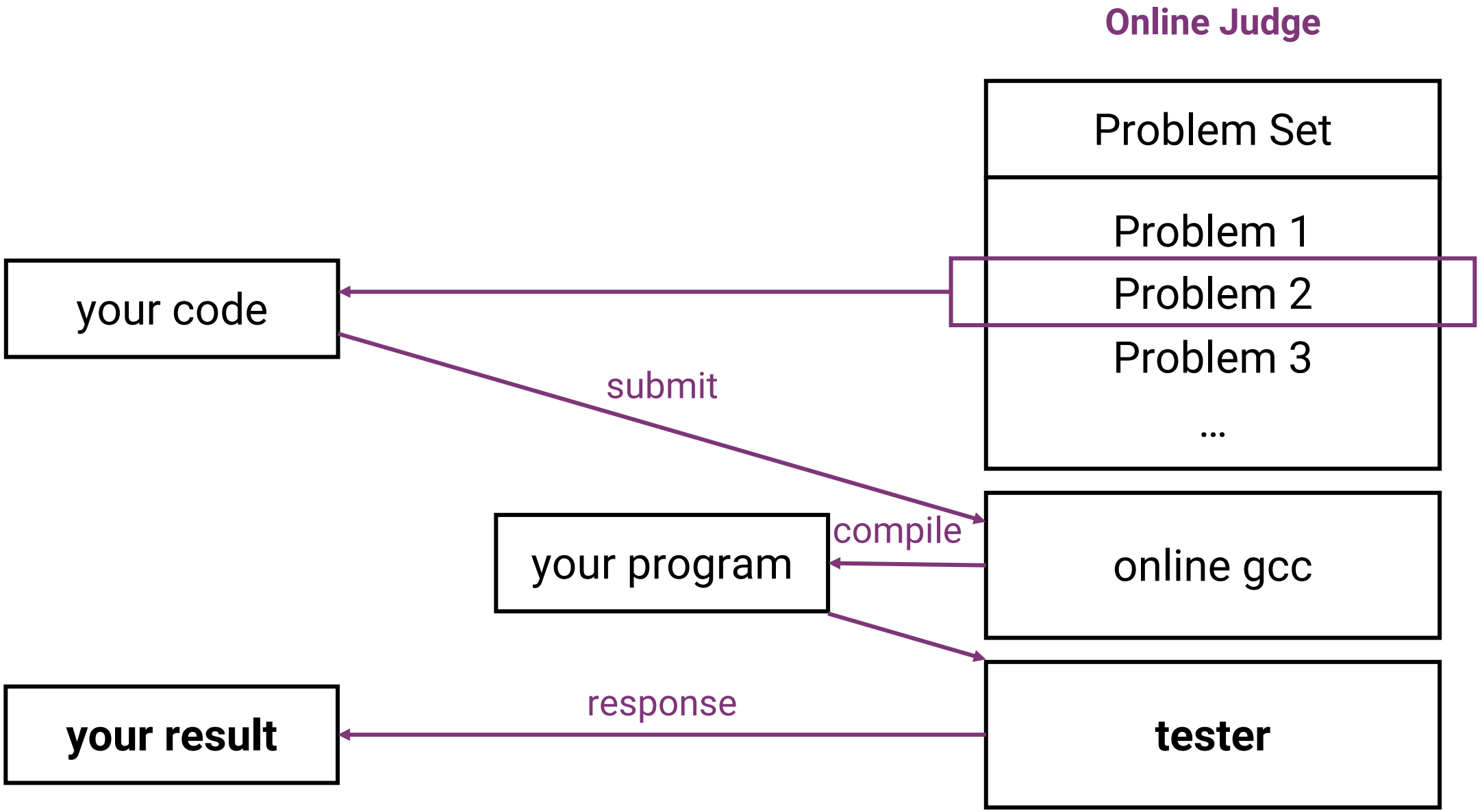
Problem ID	Title	Source	Accepted	Submit
1006	A+B+C=?	Anonymiz	0	0
1007	Sort integers	Anonymiz	0	0
1008	Swap array elements	Anonymiz	0	0
1009	Compute expression	Anonymiz	0	0
1010	Compute piece-wise function	Anonymiz	0	0
1011	Celsius-Fahrenheit conversion table	Anonymiz	0	0
1012	Find matrix element	Anonymiz	0	0
1013	Record marks	Anonymiz	0	0

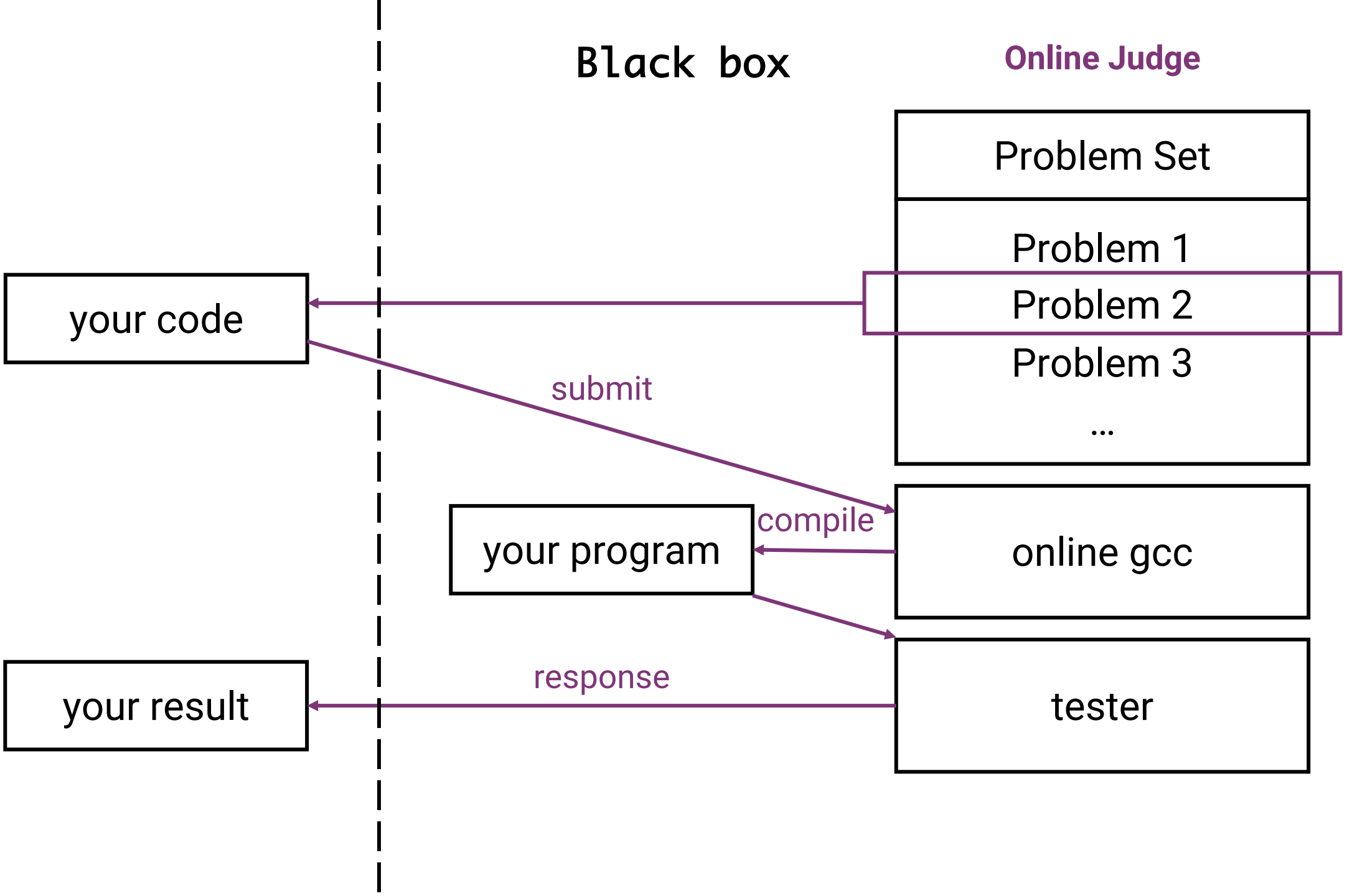
How does Online Judge work?

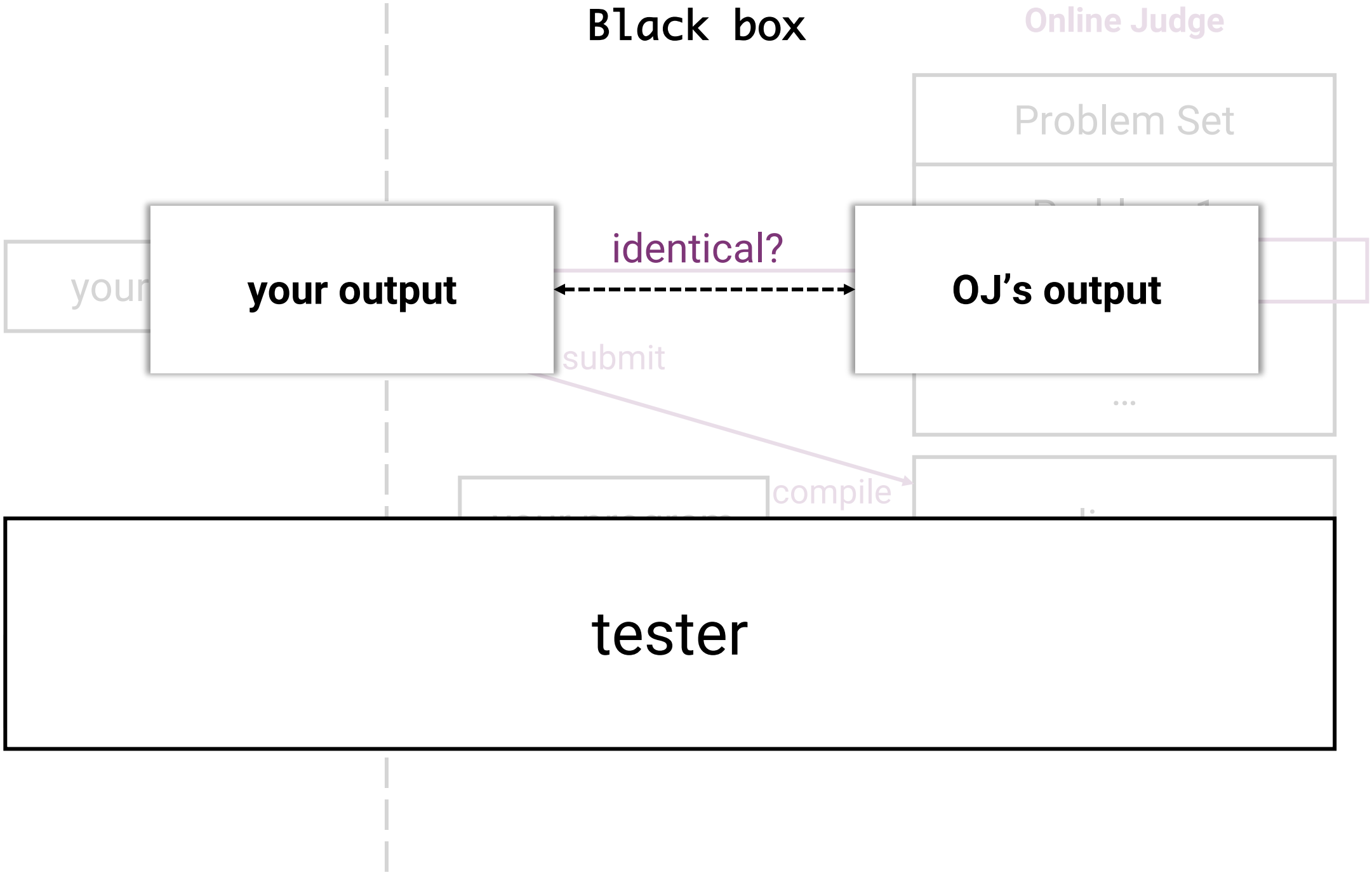

Online Judge
Problem Set
Problem 1
Problem 2
Problem 3
...
online gcc
tester


Online Judge
Problem Set
Problem 1
Problem 2
Problem 3
...
your code
online gcc
tester


Online Judge
Problem Set
Problem 1
Problem 2
Problem 3
...
your code
submit
compile
your program
online gcc
tester


Online Judge
Problem Set
Problem 1
Problem 2
Problem 3
...
your code
submit
your program
compile
online gcc
your result
response
tester


Black box
Online Judge
Problem Set
Problem 1
Problem 2
Problem 3
...
your code
submit
your program
compile
online gcc
your result
response
tester


Black box
Online Judge
Problem Set
your output
identical?
OJ's output
submit
compile
tester

Reproducing OJ

1. Prepare `test.in`
   a text file with input in the format specified in assignment
2. Compile
   ```
   % gcc YOURCODE.c
   ```
3. Run your code 'piping' the input into it
   ```
   % ./a.out <test.in> test.out
   ```

This generates the output in the file `test.out`.

Inspect it to see if there is any unexpected elements in your output

Example 7

```
#include <stdio.h>
int main(void)
{
    int x, y;
    scanf("%i %i", &x, &y);
    printf( "%i %i", x*2, y*2);
    return 0;
}
```

Terminal

```
./a.out <test.in> test.out
```

Reproducing OJ

`-Wall`

enables all warnings...this is **useful**!!

```
gcc -Wall [options][source files][object files][-o output file]
```

Example 8

```
#include <stdio.h>
int main()
{
    printf("Program run!\n");
    int i=10;
}
```

Example 8

```
#include <stdio.h>
int main()
{
    printf("Program run!\n");
    int i=10;
}
```

Terminal

```
gcc -Wall 8.c
8.c:6:9: warning: unused variable 'i' [-Wunused-variable]
    int i=10;
        ^
1 warning generated.
```

Reproducing OJ

Try `getchar( )` and `putchar( )` in your code!!!

second assessment is really useful!

A variant of **Problem 1006: A+B+C=?**

- **Description**
  Calculate a+b+c
- **Input**
  Three integers a, b, c (0<=a<=10, 0<=b<=10, 0<=c<=5)
- **Output**
  Output a+b+b
- **Sample Input**
  1 2 3
- **Sample Output**
  6

- **Solution**

```c
#include <stdio.h>
int main(void)
{
    // declare variables before using them
    int a, b, c, answer;

    // read a integer number as input and store it in a
    scanf("%i", &a);

    // read a integer number as input and store it in b
    scanf("%i", &b);

    // read a integer number as input and store it in c
    scanf("%i", &c);

    // compute the solution and store it in answer
    answer = a + b + c;

    // print out the solution as a integer number
    printf("%i\n", answer);
}
```

- **Solution**

```c
#include <stdio.h>
int main(void)
{
    /* declare variables before using them */
    int a, b, c, answer;

    /* read a integer number as input and store it in a */
    scanf("%i", &a);

    /* read a integer number as input and store it in b */
    scanf("%i", &b);

    /* read a integer number as input and store it in c */
    scanf("%i", &c);

    /* compute the solution and store it in answer */
    answer = a + b + c;

    /* print out the solution as a integer number */
    printf("%i\n", answer);
    return 0;
}
```

- **Solution**

For the assignment submission

```
/*
 * Student ID: 201234567
 * Student Name: Bruce Lee
 * Email: bruce.lee@student.liverpool.ac.uk
 *
 * User: sgxyz6
 *
 * Problem ID: 1006
 * RunID: 22456
 * Result: Accepted
 */

#include <stdio.h>
int main(void)
{
      /* declare variables before using them */
      int a, b, c, answer;

      /* read a integer number as input and store it in a */
      scanf("%i", &a);
```

Text

- Solution

```
#include <stdio.h>
int main(void)
{
        /* declare vari
        int a, b, c, an

        /* read a integ
        scanf("%i", &a)

        /* read a integ
        scanf("%i", &b)

        /* read a integ
        scanf("%i", &c)

        /* compute the
        answer = a + b

        /* print out th
        printf("%i\n",
        return 0;
}
```

Let’s try OJ

https://student.csc.liv.ac.uk/JudgeOnline…

OJ’s Response

- **In Queue (QU)**: The judge is busy and can't attend your submission. It will be judged ASAP.
- **Accepted (AC)**: OK! Your program is correct! It produced the right answer in reasonable time and within the limit memory usage. Congratulations!
- **Presentation Error (PE)**: Your program outputs are correct but are not presented in the correct way. Check for spaces, justify, line feeds...
- **Wrong Answer (WA)**: Correct solution not reached for the inputs. You'll have to spot the bug.

Do read the code

- **Compile Error (CE)**: The compiler could not compile your program. Of course, warning messages are not error messages.

OJ’s Response

- **Runtime Error (RE)**: Your program failed during the execution (segmentation fault, floating point exception, etc.). Ensure that your program returns a 0 code to the shell.
- **Time Limit Exceeded (TL)**: Your program tried to run during too much time; this error doesn't allow you to know if your program would reach the correct solution to the problem or not.
- **Memory Limit Exceeded (ML)**: Your program tried to use more memory than the judge allows.
- **Output Limit Exceeded (OL)**: Your program tried to write too much information. This usually occurs if it goes into a infinite loop.

OJ’s Response

- **Runtime Error (RE)**: Your program failed during the execution (**segmentation fault**, floating point exception, etc.). Ensure that your program returns a 0 code to the shell.
- **Time Limit Exceeded (TL)**: Your program tried to run during too much time; this error doesn't allow you to know if your program would reach the correct solution to the problem or not.
- **Memory Limit Exceeded (ML)**: Your program tried to use more memory than the judge allows.
- **Output Limit Exceeded (OL)**: Your program tried to write too much information. This usually occurs if it goes into a infinite loop.

OJ's Response

- **Runtime Error (RE)**: Your program failed during the execution (segmentation fault, floating point exception, etc.). Ensure that your program returns a 0 code to the

segmentation fault

- A "segmentation fault" is a common runtime error.
- When you run your program and the system reports a "segmentation fault", it means your program has attempted to access an area of **memory** that it is not allowed to access.
- In other words, it attempted to stomp on **memory** ground that is **beyond the limits** that the OS has allocated for your program.

This usually occurs if it goes into a infinite loop.

Troubleshooting http://web.mit.edu/10.001/Web/Tips/tips_on_segmentation.html

When OJ says “No”

- Before you start debugging... **Read OJ’s response!**
  - There may be useful information there!
  - e.g., is it a 'run-time' error or 'compilation' error?
- Things to do:
  - Read the problem again
    - Are you sure you have interpreted it correctly?

When OJ says “No”

- Things to do:
  - Test it yourself:
    - Did you try different test cases?
    - What **assumptions** did you make about the test cases?
    - Did you assume the test case(s) give you all of the possibilities?
    - Did you try large numbers, small numbers, negative numbers, etc.?
  - Check the output
    - Does the output match exactly?
    - Whitespace matters!
    - Don’t print anything else to the screen.

When OJ says “Compilation Error”

- “...and it works on my computer”
- That is great, but not sufficient!
  - remember: compilers are different
- But compilation errors are easy to fix: compiler tells you what went wrong, so try using gcc.
- Also, check the output box in OJ
  - it gives you the error message!

When OJ says “Runtime Error”

- Problem: this typically is a segmentation fault.
- Unfortunately, more difficult to resolve
- Approaches:
  - **Testing:** try different inputs until you can reproduce the error.
  - **Print debugging**: use print statements to understand where the problem occurs.
  - **Use a debugger tool**: use the debugger gdb (in later lecture) to find out what causes the error.
- If that does not work, resolve at the lab!

Key Points

- You are free to submit any of the problems
  - as often as you like.
  - it's all there to practice, so go for it.
- OJ shows
  - Runtime
  - memory usage
  - keep you programs as efficient as possible!
- When you 'submit', OJ will generate a **RunID**
  - this is important when submitting to the departmental system.

Summary

Today

- C Language Basics
  - **Loop**
- Online Judge

Next

- Function