

# Advanced Flow Control

Song Liu ([song.liu@bristol.ac.uk](mailto:song.liu@bristol.ac.uk))

GA 18, Fry Building,

Microsoft Teams (search "song liu").

# Previously, we have talked about

- Conditional Statements
  - `if-else`
  - `if-else if` ladder
- Loops
  - `for` loop
  - `while` loop

# Issues in Previous Lab

- Forgetting semicolon ;
  - In C, all statements ends with ;
  - `printf("hello world!");` , `verse1();` , `double pi = 3.14;` , `return 0;`
- Conditional statements do not have to end with ;

```
if(a == 1){  
    printf("a equals to 1\n");  
} // NO semicolon
```

- Loops do not have to end with ;

```
while(a <= 1){  
    printf("looping...\n"); a = a + 1;  
} // NO semicolon
```

# Issues in Previous Lab

- Semicolon signals the end of the previous statement and prepares CPU for the next statement to be executed.
- In some other programming languages (such as python), the end of a statement is signaled by a line break.

- ```
# This is python code
print("test1") # end of a statement
print("test2")
```

- By using `;`, you can write multiple statements in one line:
  - `printf("looping...\n"); a = a + 1;`
  - This is not allowed in python.

# Issues in Previous Lab

- In `if-else if` ladder, the order matters!

```
if (condition1){  
    statment1;  
} else if(condition2){  
    statment2;  
} else if(condition3){  
    statment3;  
}  
...  
else{ //optional  
    statement0;  
}
```

- The program will check conditions **sequentially**.
- Once a true condition is found
  - It executes the associated statements.
  - then **bypasses the rest of the ladder**.

# Issues in Previous Lab

- Write an `if-else if` ladder, so it prints
  - `divided by 15`, if `a` can be divided by 15.
  - `divided by 5` if the number can **only be** divided by 5 but not by 15.

```
if(a % 15 == 0){ //this must be first!
    printf("divided by 15\n");
}else if(a % 5 == 0){
    printf("divided by 5\n");
}
```

# Today's Lecture

- Nested if-else
- Nested loop
- Advanced loop control
- Recursion

# Nested If-Else

- You can write conditional statement inside another conditional statement.

```
if(score >= 40){  
    printf("congratulations! ");  
    if(score >=70){  
        printf("first class!\n");  
    }else{  
        printf("passed!\n");  
    }  
}else{  
    printf("student has failed!\n");  
}
```

- The code prints out
  - student has failed! if score < 40 .
  - congratulations! passed! if 40 <=score < 70 .
  - congratulations! first class! if score >= 70 .



# Nested If-Else

- In some cases, nested `if-else` can be translated into a single `if-else if` ladder.

```
if(score >= 70){
    printf("congratulations!");
    printf("first class!\n");
else if (score >= 40){
    printf("congratulations!");
    printf("passed!\n");
}else{
    printf("student has failed!\n");
}
```

- Which one to use depends on which one leads to a cleaner code.
  - Notice, in the first example, `score < 40` and `40 <= score < 70` shares the first `printf`.

# Nested Loops

- Similarly, you can write one loop inside another loop.

```
for (int i = 1; i <= 4; i=i+1){  
    // print i-th line  
    for (int j = 1; j <= 4; j=j+1){  
        printf("*");  
    }  
    printf("\n"); // change line  
}
```

- It prints out a block of \*

```
****  
****  
****  
****
```

# Nested Loops

- Last year's exam question (simplified): Write a C program which prints out

```
*  
**  
***  
****
```

- Try it yourself!

# Early Loop Exit

- `break;` statement will exit the loop immediately.
- Find the smallest integer `a` from 1 to 100 that satisfies the inequality `a*a + a > 321`.

- ```
int a = 1;
while(a <= 100){
    if(a*a + a > 321){
        printf("%d\n", a);
        break; // exit the while loop immediately.
    }
    a = a + 1;
}
```

- No need to continue the search after you have found one as the question asks for the smallest.

# Early Loop Restart

- `continue;` statement will restart the loop immediately.
- Once the program encounters a `continue;` statement, it will **skip over the rest** of the statements in the loop and start the next iteration immediately.

```
int i;
for(i = 1; i < 10; i = i + 1){
    if(i % 2 == 0){
        continue; //skip all even numbers
    }
    printf("%d ", i);
}
printf("\n");
// print 1 3 5 7 9
```

# Early Loop Restart

- What will happen if you run the code below?

```
int i = 1;
while(i < 10){
    if(i % 2 == 0){
        continue;
    }
    printf("%d ", i);
    i = i + 1
}
// ???
```

Take a guess.

- 1 3 5 7 9
- 2 4 6 8
- other

# Early Loop Restart

Answer: it will print out `1` then stuck (loop will not stop).

`continue` will skip overall statements in the loop body, including the increment of `i`.

# Recursion

- You **cannot** define a function inside another function.
- You can **call** a function inside another function.
  - A function can call itself!
  - A function calling itself is called recursion.



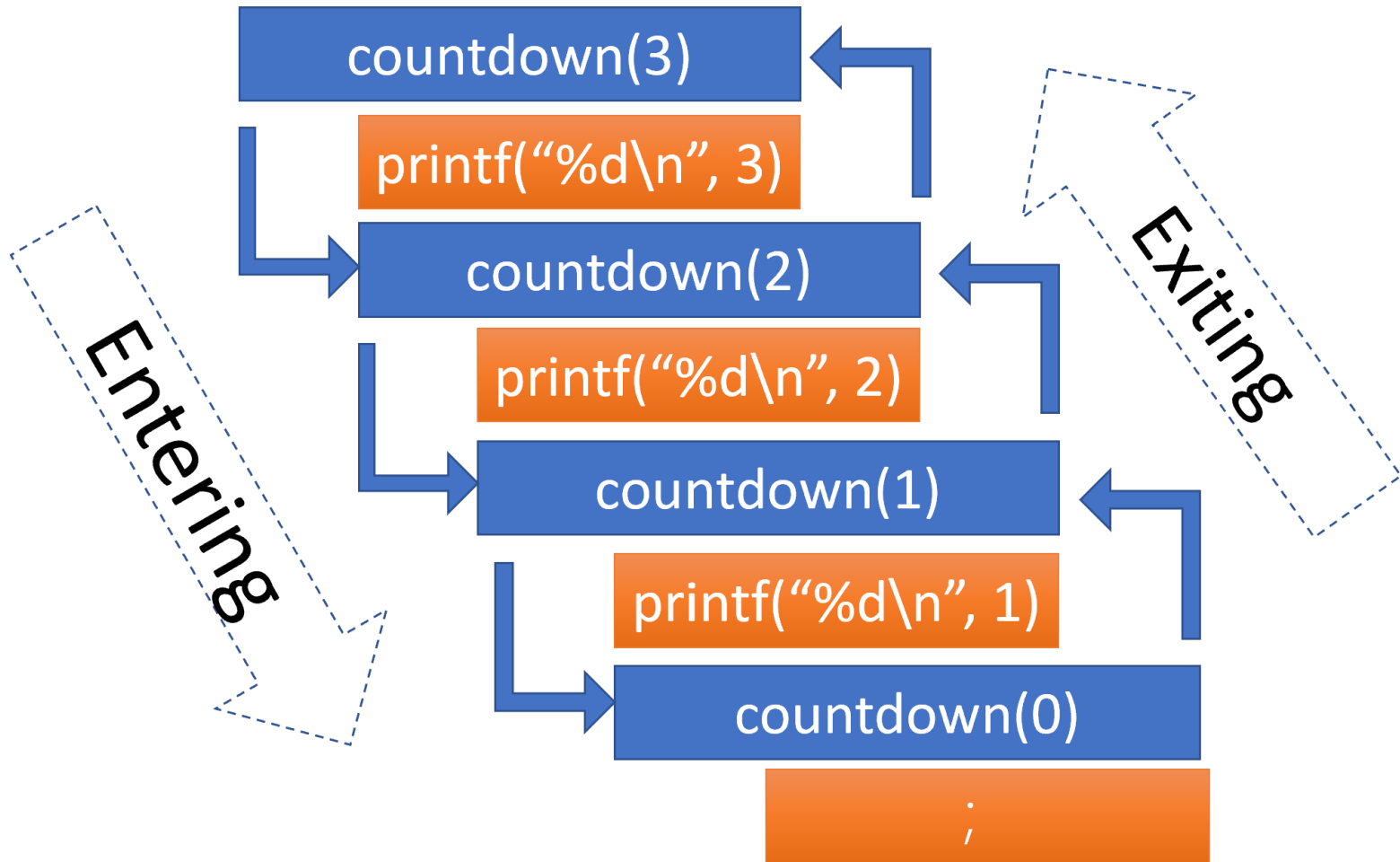
# Recursion

- ```
void countdown_to_1(int n){
    if(n >= 1 ){
        printf("%d\n", n);
        countdown_to_1(n - 1);
    }
}

void main(){
    countdown_to_1(3);
}
```

- Prints out 3 2 1.
- For each  $n > 0$ , it prints out  $n$  and initiate countdown with a smaller number  $n-1$ .

# Calling a Function: Recursion



Recursive function must have an entering and exiting path!

# Recursion

What will happen if we do

```
void countdown_to_1(int n){  
    printf("%d\n", n);  
    countdown_to_1(n - 1);  
}  
  
void main(){  
    countdown_to_1(3);  
}
```

# Conclusion

In this lecture, we talked about some more advanced flow-control techniques:

- Nested `if-else`
- Nested loops
- Early loop stop and restart
- Recursion

# Homework 1

1. Download today's lab files from github, unzip and place them into your labpack.
  - The same you did for the last week's lab.
2. Double click lab3.bat to run the labpack.

# Homework 2

1. Open `nestedif.c` , trace the execution using debugger (by pressing F5 then step over).
2. Make sure you understand the workflow of nested if.
3. Make modifications on `nestedif.c` , so the program outputs:
  - `student has failed! if score < 40 .`
  - `congratulations! passed! if 40 <=score < 50 .`
  - `congratulations! 2:2 if 50 <=score < 60 .`
  - `congratulations! 2:1 if 60 <=score < 70 .`
  - `congratulations! first class! if score >= 70 .`

# Homework 3 (Submit)

Open `max.c`

Write a function `max` at the specified place. The function takes three integer inputs: `a,b,c` . It returns the maximum.

# Homework 4

1. Open `nestedfor.c` , trace the execution using debugger (by pressing F5 then step over).
2. Make sure you understand the workflow of the nested loops.
3. Make modifications on `nestedif.c` , so the program outputs:

```
*  
***  
*****  
*****  
*****
```

You must use for loop for that.

This is the actual exam question from the last year.



# Homework 5 (Submit)

Open `prime.c` .

Write a program that prints out all prime numbers from 1 to 1000. To do this question you need to use conditional statements and a nested loop.

See the next slide if you want some hints. Otherwise, stay on this slide until you finish.

# Homework 5 (submit)

You can imagine a program with the following structure:

```
for i from 1 to 1000
  numfactors = 0
  for j from 1 to i
    if(i can be divided by j)
      numfactors = numfactors + 1

  if numfactors equals to 2 //primes have only 2 factors.
    print out i
```

- The above code is NOT C code. Please translate them into C code.
- This algorithm can be made more efficient (how?)

# Submission

- Please creating a zip file containing both `max.c` and `prime.c` files
  - Please search online for help if you are not sure how to create a zip file.
- Rename the file to `ab1234.zip` where `ab1234` is your email account before the @ symbol.
- Uploaded it to the blackboard.