

ECE 220

Lecture x0006 - 02/05

Slides based on material originally by: Yuting Chen, Yih-Chun Hu & Thomas Moon

Recap

Requires `#include<stdbool.h>`

- Last time we discussed C language:
 - Dynamic vs. static typing
 - Compiled vs. interpreted languages
 - Variables in C
 - Identifiers, scope, linkage, storage class

- Data types:

- `int, float, char, bool`

- qualifiers

- `static, extern`
- `const`

Makes a variable *immutable*

Example

```
#include <stdio.h>

int main(){
// defining integer constant using const keyword
const int int_const = 25;

// defining character constant using const keyword
const char char_const = 'A';

// defining float constant using const keyword
const float PI;
PI = 3.14;

printf("Printing value of Integer Constant: %d\n", int_const);
printf("Printing value of Character Constant: %c\n", char_const);
printf("Printing value of Float Constant: %f", PI);

    return 0;
}
```

Illegal, declaration & definition must be combined!

Remark: const

Note: **const** variables **not** immune to pointer manipulation just like **static** variables.

```
#include <stdio.h>

int main(){
    // defining an integer constant
    const int var = 10;

    printf("Initial Value of Constant: %d\n", var);

    // defining a pointer to that const variable
    int* ptr = &var;

    // changing value
    *ptr = 500;
    printf("Final Value of Constant: %d", var);
    return 0;
}
```

Lesson objectives

- Understand basic relational, logical and arithmetic operations in C.
- Understand operator precedence vs. order of evaluation
- Understand basic output (`printf`) and input (`scanf`) functions and use of *address-of* operator.
- Understand basic control, branch and loop structures in C.

Operators: basic concepts

- **Operator precedence**
- Associativity
- Statements vs. expressions
- Order of evaluation

The “rank” of an **operator** is called its **precedence**, and an operation with a higher **precedence** is performed before operations with lower **precedence**.

Operators: basic concepts

- Operator precedence
- **Associativity**
- Statements vs. expressions
- Order of evaluation

The **associativity** of an operator is a property that determines how operators of the *same precedence* are grouped in the absence of parentheses.

Left associative $a + b + c = (a + b) + c$

Right associative $a + b + c = a + (b + c)$

Operators: basic concepts

- Operator precedence
- Associativity
- **Statements vs. expressions**
- Order of evaluation

Statements represent a *complete* unit of work to be carried out by the digital hardware.

Expressions are syntactically valid groupings of variables, operators, and *literal* values.

$$2 * (x + 2)$$
$$k = k + 1;$$

Operators: basic concepts

- Operator precedence
- Associativity
- Statements vs. expressions
- **Order of evaluation**

Expressions are evaluated in order of precedence following associativity rules

$$2 + 3 - 4 + 5 = ((2 + 3) - 4) + 5$$

Operators: basic concepts

- Operator precedence
- Associativity
- Statements vs. expressions
- **Order of evaluation**

Note: The compiler order of evaluation is independent of precedence and associativity and may change between consecutive calls to the same code snippet.

$f1() + f2() + f3()$ is parsed as $(f1() + f2()) + f3()$ due to left-to-right associativity of operator $+$, but the function call to $f3$ may be evaluated first, last, or between $f1()$ or $f2()$ at run time.

Operators: basic types

- **Assignment**
 - Arithmetic
 - Bitwise
 - Relational
 - Logical
 - Increment/decrement
- Evaluates whatever is to the right of “=” and assigns that value to whatever is to the left of the “=”
 - Beware comparison vs assignment: == vs =

Operators: basic types

- Assignment
- **Arithmetic**
- Bitwise
- Relational
- Logical
- Increment/decrement

Table 12.1 Arithmetic Operators in C

Operator symbol	Operation	Example usage
*	multiplication	x * y
/	division	x / y
%	integer remainder	x % y
+	addition	x + y
-	subtraction	x - y

Operators: basic types

- Assignment
- Arithmetic
- **Bitwise**
- Relational
- Logical
- Increment/decrement

Table 12.2 Bitwise Operators in C

Operator symbol	Operation	Example usage
~	bitwise NOT	~ x
&	bitwise AND	x & y
	bitwise OR	x y
^	bitwise XOR	x ^ y
«	left shift	x « y
»	right shift	x » y

Operators: basic types

- Assignment
- Arithmetic
- Bitwise
- **Relational**
- Logical
- Increment/decrement

Table 12.3 Relational Operators in C

Operator symbol	Operation	Example usage
>	greater than	$x > y$
>=	greater than or equal	$x \geq y$
<	less than	$x < y$
<=	less than or equal	$x \leq y$
==	equal	$x == y$
!=	not equal	$x != y$

Operators: basic types

- Assignment
- Arithmetic
- Bitwise
- Relational
- **Logical**
- Increment/decrement

Table 12.4 Logical Operators in C

Operator symbol	Operation	Example usage
!	logical NOT	!x
&&	logical AND	x && y
	logical OR	x y

Operators: basic types

- Assignment
 - Two flavors pre and post
- Arithmetic
- Bitwise
- Relational
- Logical
- **Increment/decrement**

```
x=4;
```

```
y=x++;
```

```
z=++x;
```

Operator precedence

Table 12.5 Operator Precedence and Associativity in C

Precedence Group	Associativity	Operators
1 (highest)	left-to-right	() (function call) [] (array index) . (structure member) -> (structure pointer dereference)
2	right-to-left	++ -- (postfix versions)
3	right-to-left	++ -- (prefix versions)
4	right-to-left	* (indirection) & (address of) + (unary) - (unary) ~ (bitwise NOT) ! (logical NOT) sizeof
5	right-to-left	(type) (type cast)
6	left-to-right	* (multiplication) / (division) % (integer division)
7	left-to-right	+ (addition) - (subtraction)
8	left-to-right	<< (left shift) >> (right shift)
9	left-to-right	< (less than) > (greater than) <= (less than or equal) >= (greater than or equal)
10	left-to-right	== (equals) != (not equals)
11	left-to-right	& (bitwise AND)
12	left-to-right	^ (bitwise XOR)
13	left-to-right	(bitwise OR)
14	left-to-right	&& (logical AND)
15	left-to-right	(logical OR)
16	left-to-right	& : (conditional expression)
17 (lowest)	right-to-left	= += -= *= etc.. (assignment operators)

More complete table: https://en.cppreference.com/w/c/language/operator_precedence

Operator precedence

- Based on the operator precedence table rewrite the following expression using parentheses to indicate precedence:

$x \& z + 3 \mid \mid 9 - w \% 6$



Basic output

- We already saw the use cases for `printf` command.
- **Exercise:** Go here: <https://en.cppreference.com/w/c/io/fprintf>. Read about format specifiers. What will the following output?
 - `printf("%+d is a prime number\n", 43);`
 - `printf("43+59 in hexadecimal is: %x\n", 43+59);`
 - `printf("%.3f is approximately PI.\n", 22.0/7);`

Basic input

- The command for reading console input is `scanf` with the following syntax.

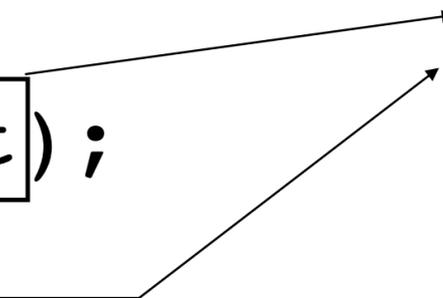
```
scanf(format_specifier, varMemAddress)
```

- **Examples:**

- `scanf("%d", &some_int);`

- `scanf("%f", &some_float);`

Takes memory address of
`some_int` and `some_float`



Basic input/output

- **Exercise:** What will be the output of the following code snippet?

```
#include <stdio.h>

int main(void){
    int num1, num2;
    printf("Enter the first number:\t");
23 → scanf("%d", &num1);
    printf("Enter the second number:\t");
ef → scanf("%x", &num2);
    int mysum = num1 + num2;
    printf("The sum of %i and %d is: %d", num1, num2, mysum);
    return 0;
}
```



Remark about floats

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

```
int main(void){
```

```
    float my_float = 3.14;
```

```
    if (my_float==3.14)
```

```
        printf("My float is PI\n");
```

```
    else
```

```
        printf("My float is not PI\n");
```

My float is not PI

```
    double my_double = 3.14;
```

```
    if (my_double == 3.14)
```

```
        printf("My double is PI\n");
```

My double is PI

```
    else
```

```
        printf("My double is not PI\n");
```

```
    return 0;
```

```
}
```

Add this line to see why. What is the fix?

```
printf("%lu, %lu, %lu\n", sizeof(3.14), sizeof(3.14f), sizeof(my_float));
```

Remark about floats

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

```
int main(void){
```

```
    float my_float = 3.14;
```

```
    if (my_float==3.14f)
```

```
        printf("My float is PI\n");
```

```
    else
```

```
        printf("My float is not PI\n");
```

My float is not PI

```
    double my_double = 3.14;
```

```
    if (my_double == 3.14)
```

```
        printf("My double is PI\n");
```

My double is PI

```
    else
```

```
        printf("My double is not PI\n");
```

```
    return 0;
```

```
}
```

Add this line to see why. What is the fix?

```
printf("%lu, %lu, %lu\n", sizeof(3.14), sizeof(3.14f), sizeof(my_float));
```

Control structures in C

1. Conditional

Making a decision about which code to execute, based on evaluated expression

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

2. Iteration

Executing code multiple times, ending based on evaluated expression

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

The if-else statement

```
if (condition)
    action_if;
else
    action_else;
```

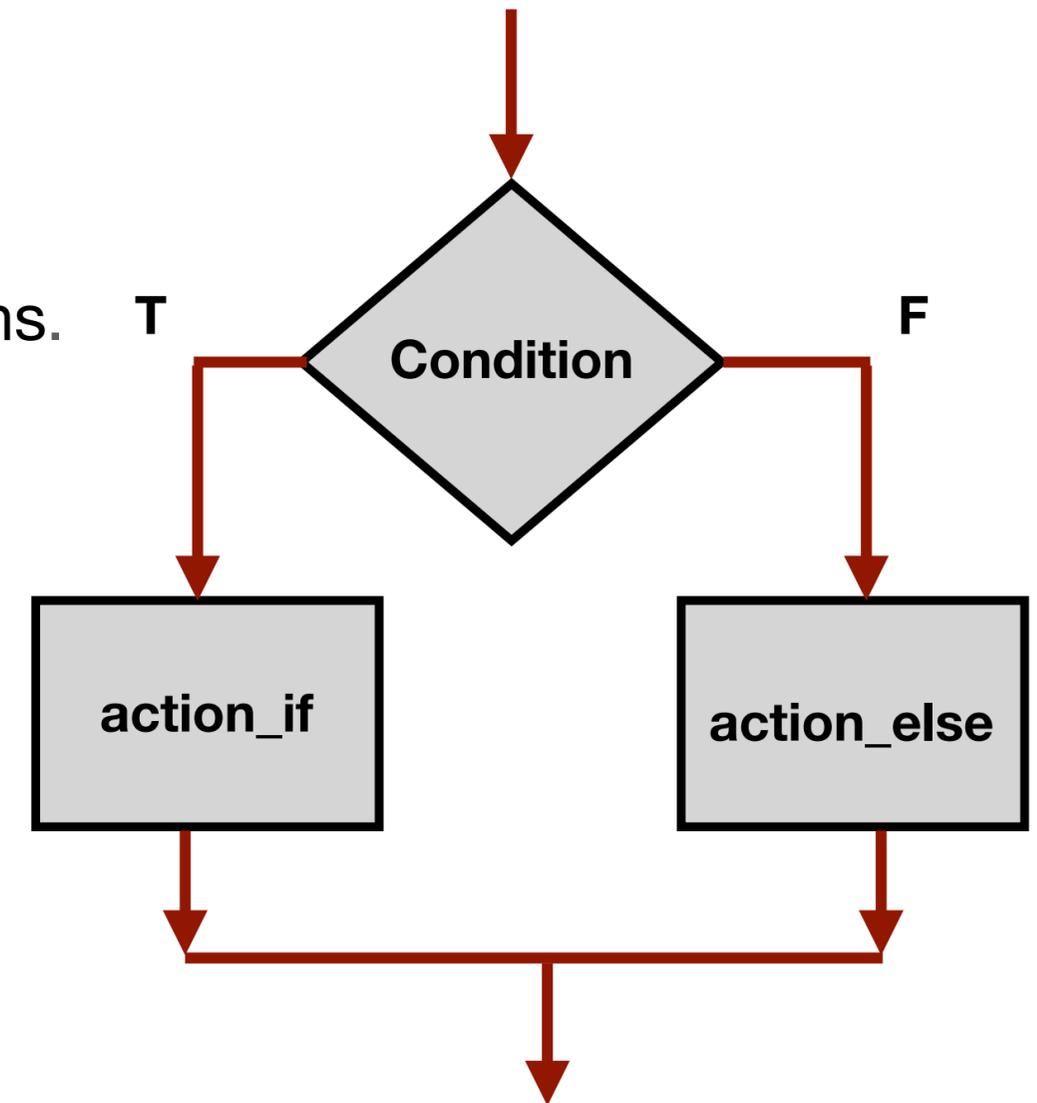
Else: allows choice between two mutually-exclusive actions.

Example 1

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

Example 2

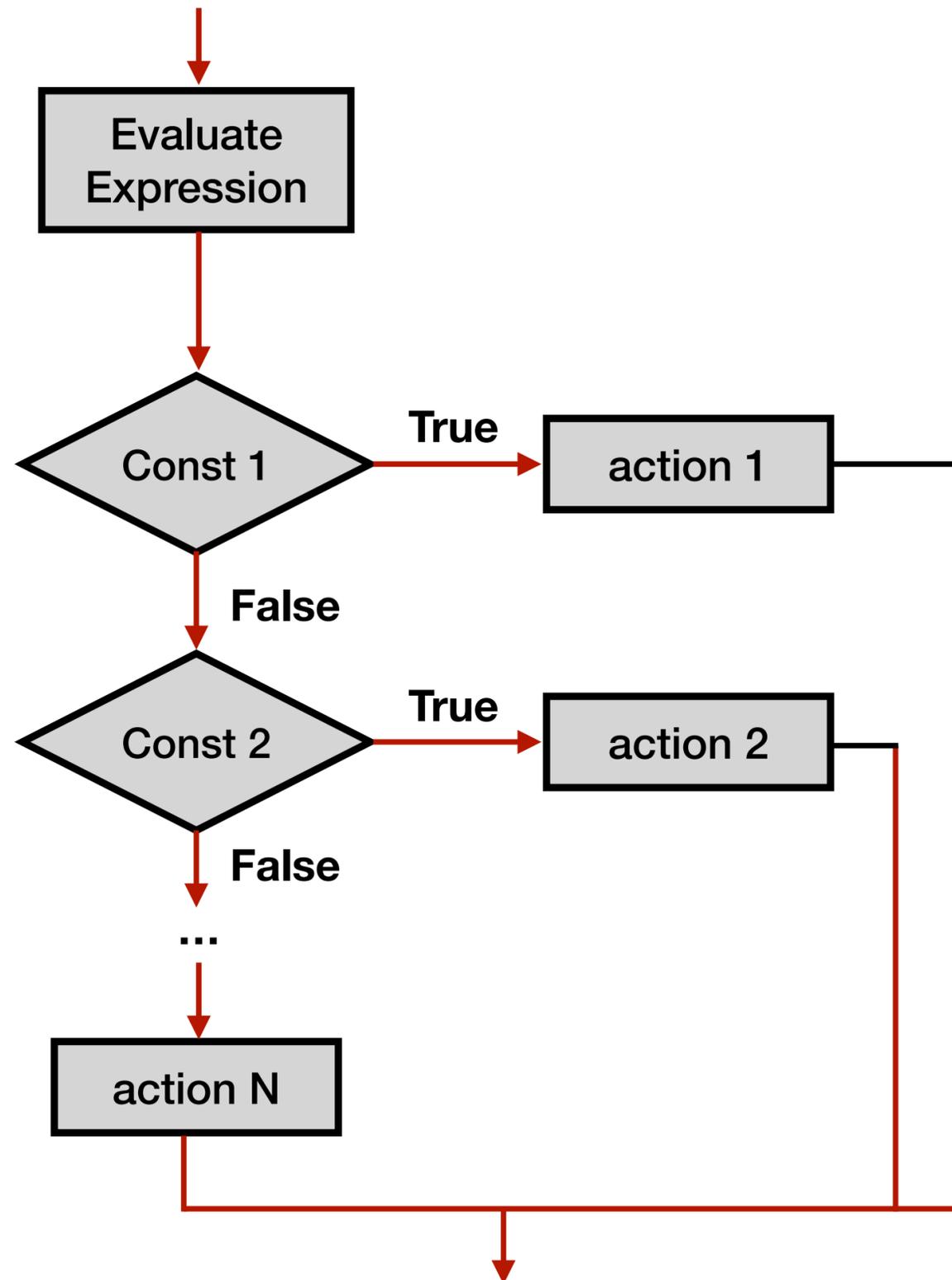
```
if ((x > 5) && (x < 25))
{
    y = x * x + 5;
    printf("y = %d\n", y);
}
else
    printf("x = %f\n", x);
```



Chaining if-else

```
if (month == 4 || month == 6 || month == 9 || month == 11) {
    printf("Month has 30 days. \n");
}
else if (month == 1 || month == 3 || month == 5 ||
        month == 7 || month == 8 || month == 10 ||
        month == 12 ) {
    printf("Month has 31 days. \n");
}
else if (month == 2) {
    printf("Month has 28 or 29 days. \n");
}
else {
    printf("Don't know that month. \n");
}
```

The switch statement



```
if  
else if  
else if  
...  
else
```



```
switch (expression)  
{  
    case const 1:  
        action 1;  
        break;  
    case const 2:  
        action 2;  
        break;  
    ...  
    default:  
        default action;  
        break;  
}  
// notice the use of break
```

If **break** is not used, then cases fall through!

The switch statement

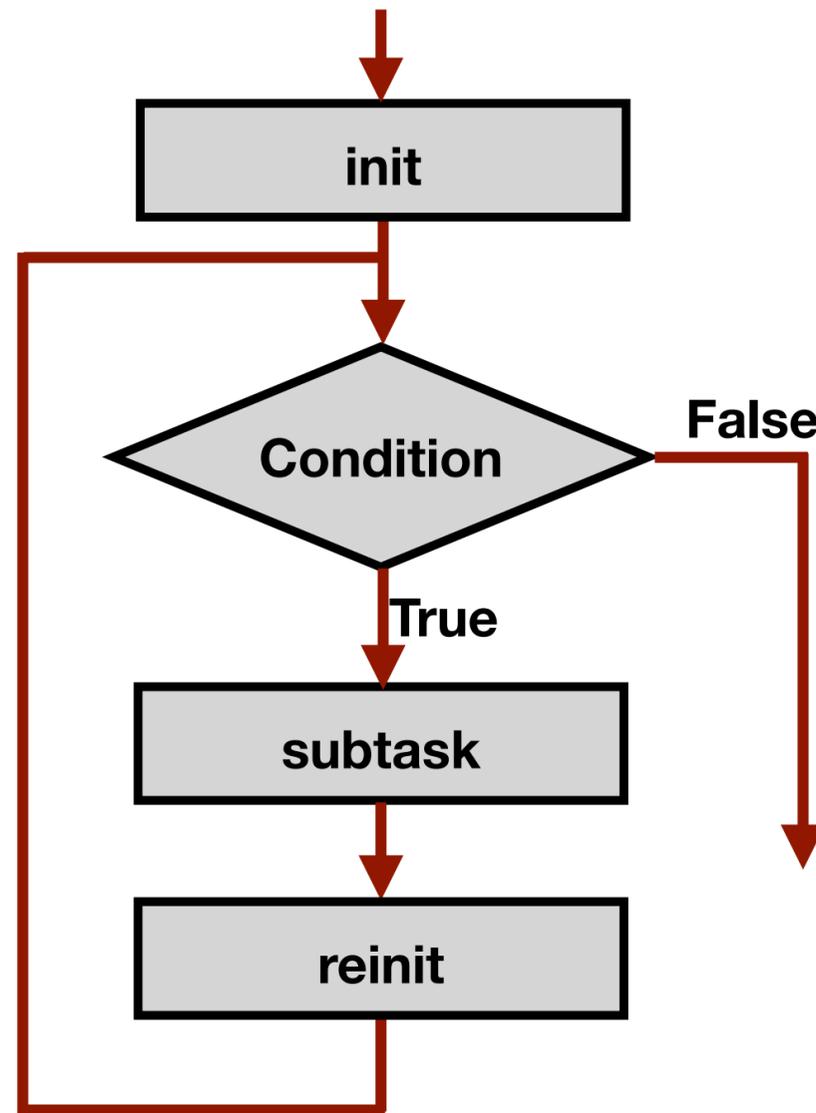
```
a = 1;
switch(a){
    case 1:
        printf("A");
        break;
    case 2:
        printf("B");
        break;
    default:
        printf("C");
        break;
}
```

Output : A

```
a = 1;
switch(a){
    case 1:
        printf("A");
    case 2:
        printf("B");
    default:
        printf("C");
}
```

Output : ABC

The for statement



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

```
for (x = 0; x < 10; x++)  
{  
    if (x == 5)  
        break;  
    printf("x=%d\n", x);  
}
```

for (init; end-test; update)
statement

break vs. continue

- **break**

- Used only in switch or iteration statement
- Used to exit a loop/block before terminating condition occurs

```
for (i = 0; i < 10; i++){  
    if(i == 5)  
        break;  
    printf("%d ", i);  
}
```

Output : 0 1 2 3 4

- **continue**

- Used only in iteration statement
- End the current iteration and start the next

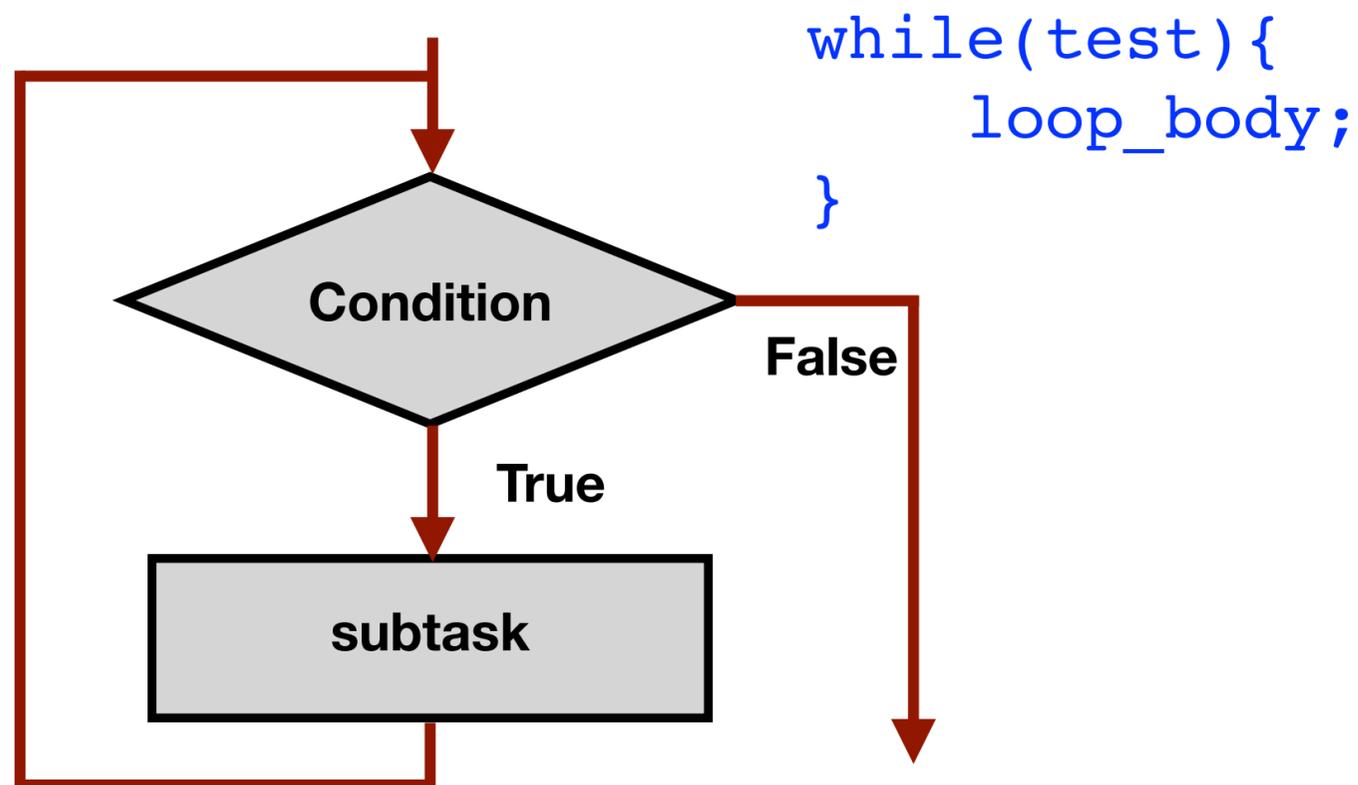
```
for (i = 0; i < 10; i++){  
    if (i == 5)  
        continue;  
    printf("%d ", i);  
}
```

Output : 0 1 2 3 4 6 7 8 9

The while / do-while statement

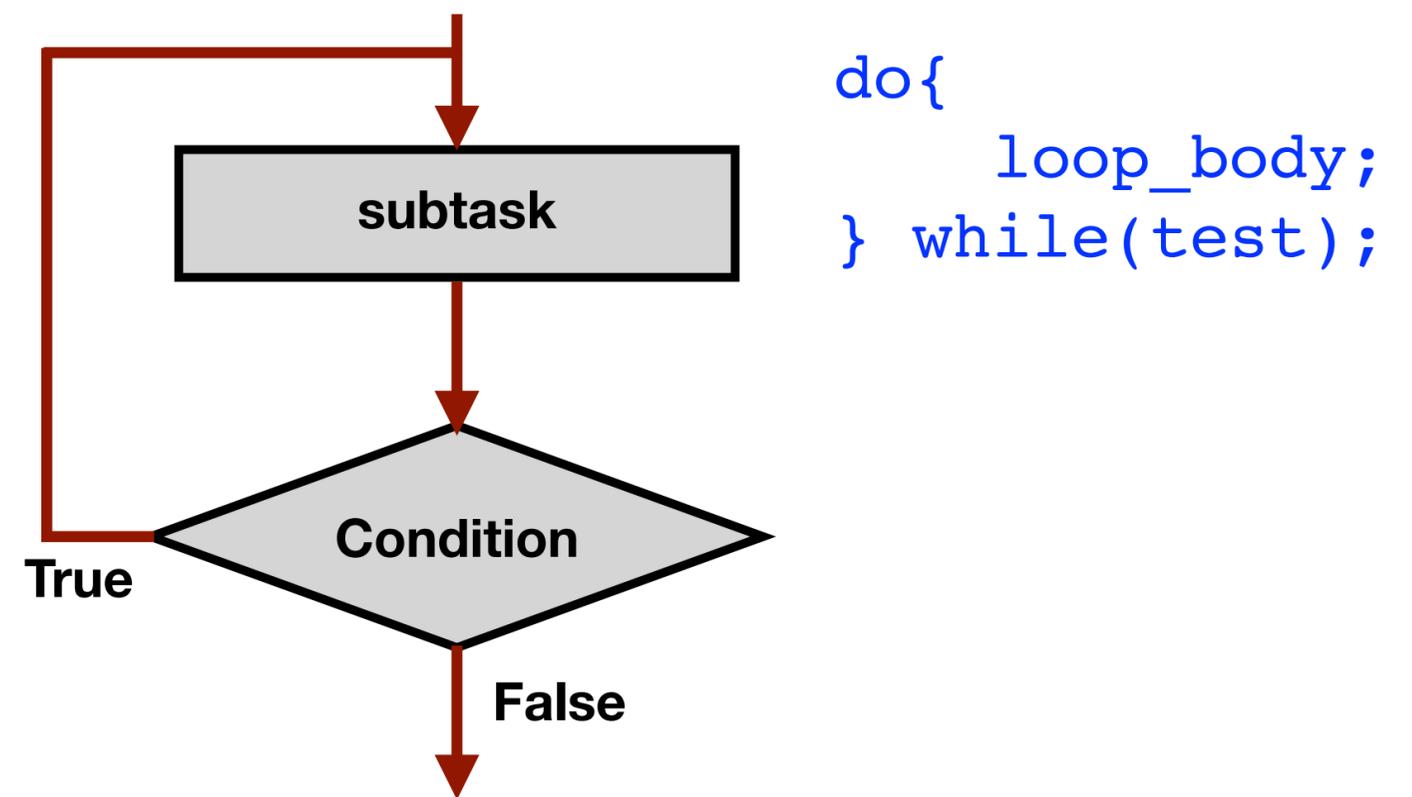
while statement

- Loop body may or may not be executed even once
- Test is evaluated **before** executing the loop.



do-while statement

- Loop body will be executed at least once
- Test is evaluated **after** executing loop body



Exercises

- Write a program that prompts and accepts an integer valued temperature reading in Fahrenheit and displays its decimal equivalent in degrees Celsius.
- Can you modify the program to keep running until the user enters a temperature below absolute zero in Fahrenheit?

Exercises

- Write a program that prompts and accepts an integer n from the user and then provided that $1 \leq n \leq 8$, prints out a $n \times n$ identity matrix to the console.
- How would you modify the program to make it print out a *lower triangular* or *upper triangular* identity matrix?

$$\begin{bmatrix} 1 & & & \\ 0 & 1 & & \\ 0 & 0 & 1 & \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$\begin{bmatrix} 1 & 0 & 0 & 0 \\ & 1 & 0 & 0 \\ & & 1 & 0 \\ & & & 1 \end{bmatrix}$$

Exercise

- Can you rewrite using switch case?

```
if (month == 4 || month == 6 || month == 9 || month == 11) {
    printf("Month has 30 days. \n");
}
else if (month == 1 || month == 3 || month == 5 ||
        month == 7 || month == 8 || month == 10 ||
        month == 12 ) {
    printf("Month has 31 days. \n");
}
else if (month == 2) {
    printf("Month has 28 or 29 days. \n");
}
else {
    printf("Don't know that month. \n");
}
```