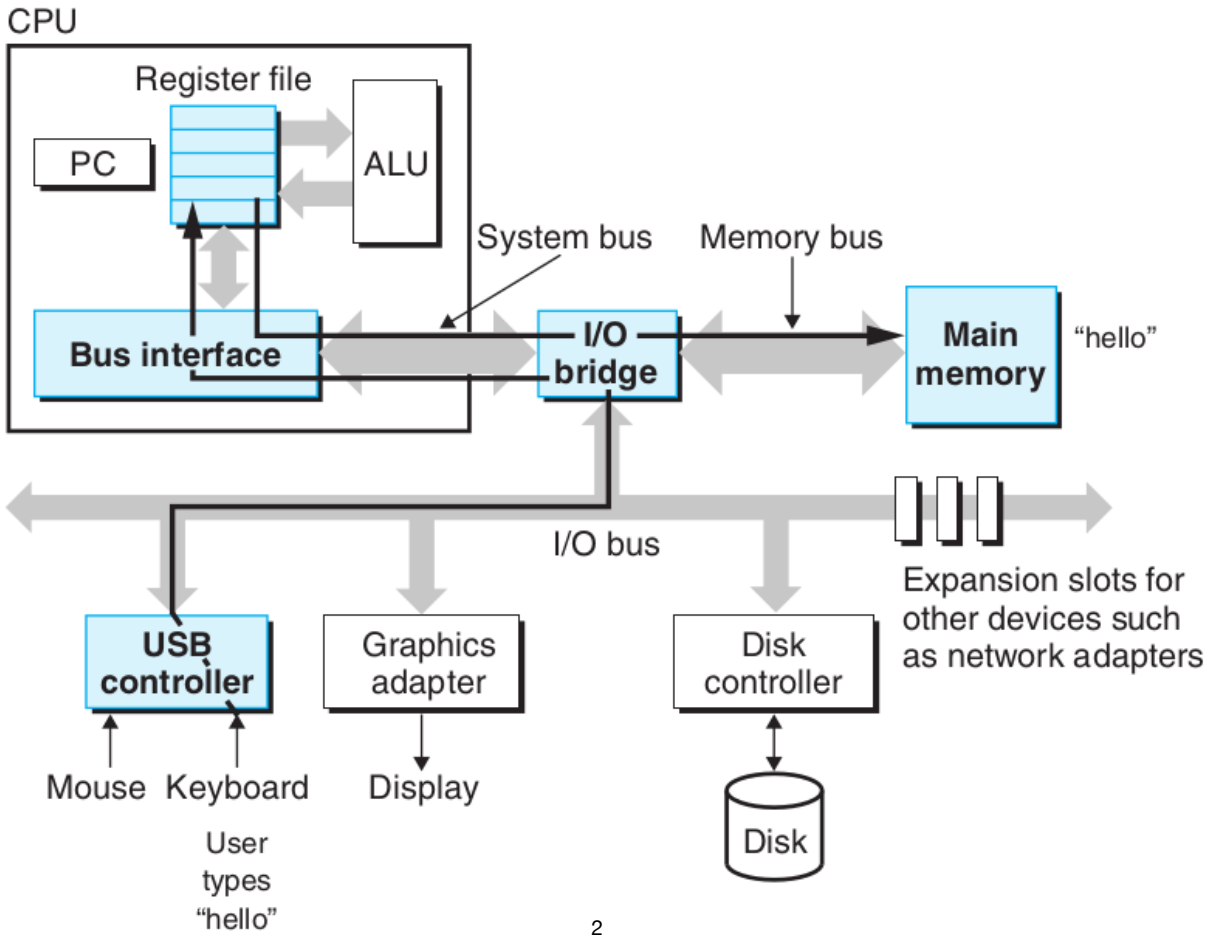
Introduction to Computer Science
Lecture 2

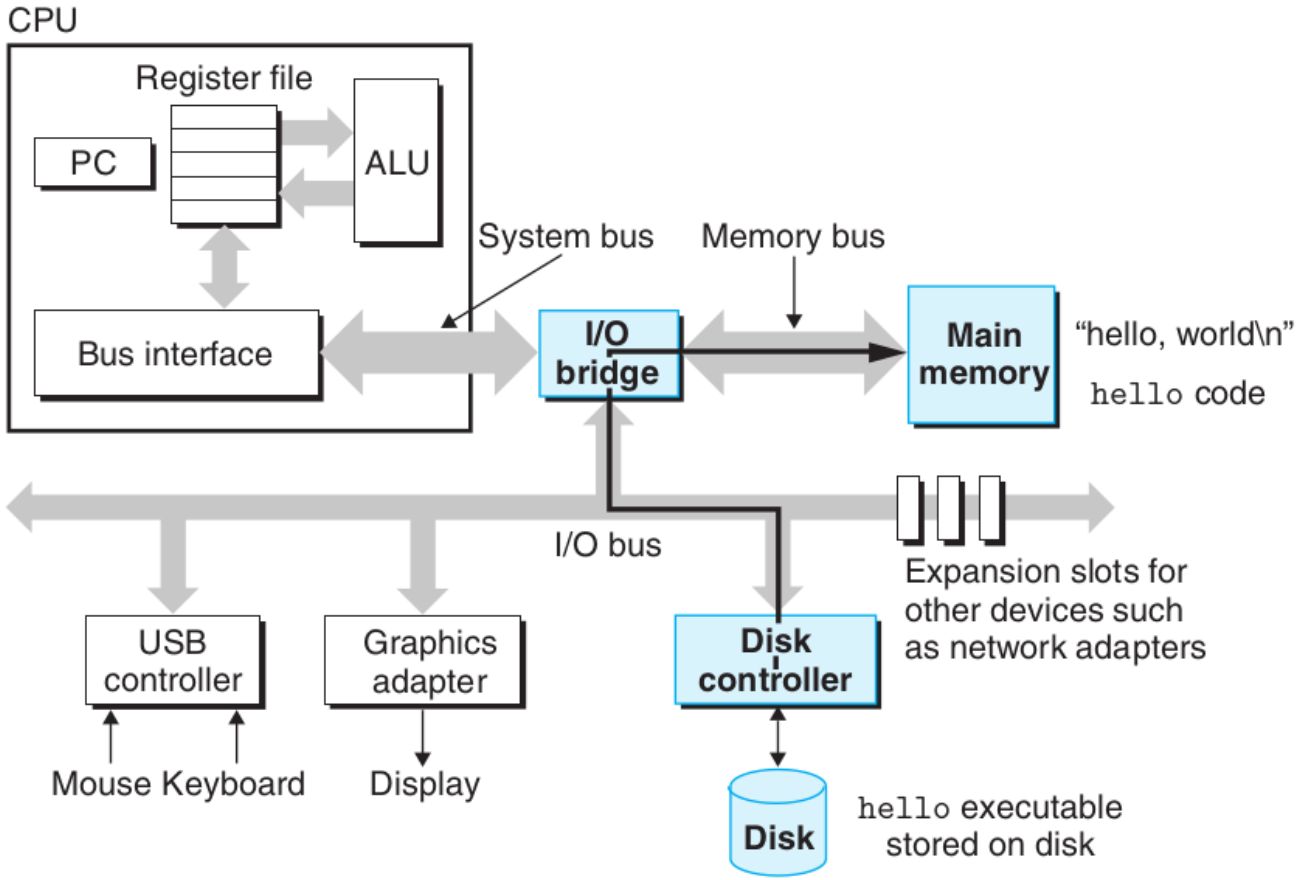
Nazareno Aguirre

(based on material by Guillaume Hoffmann)

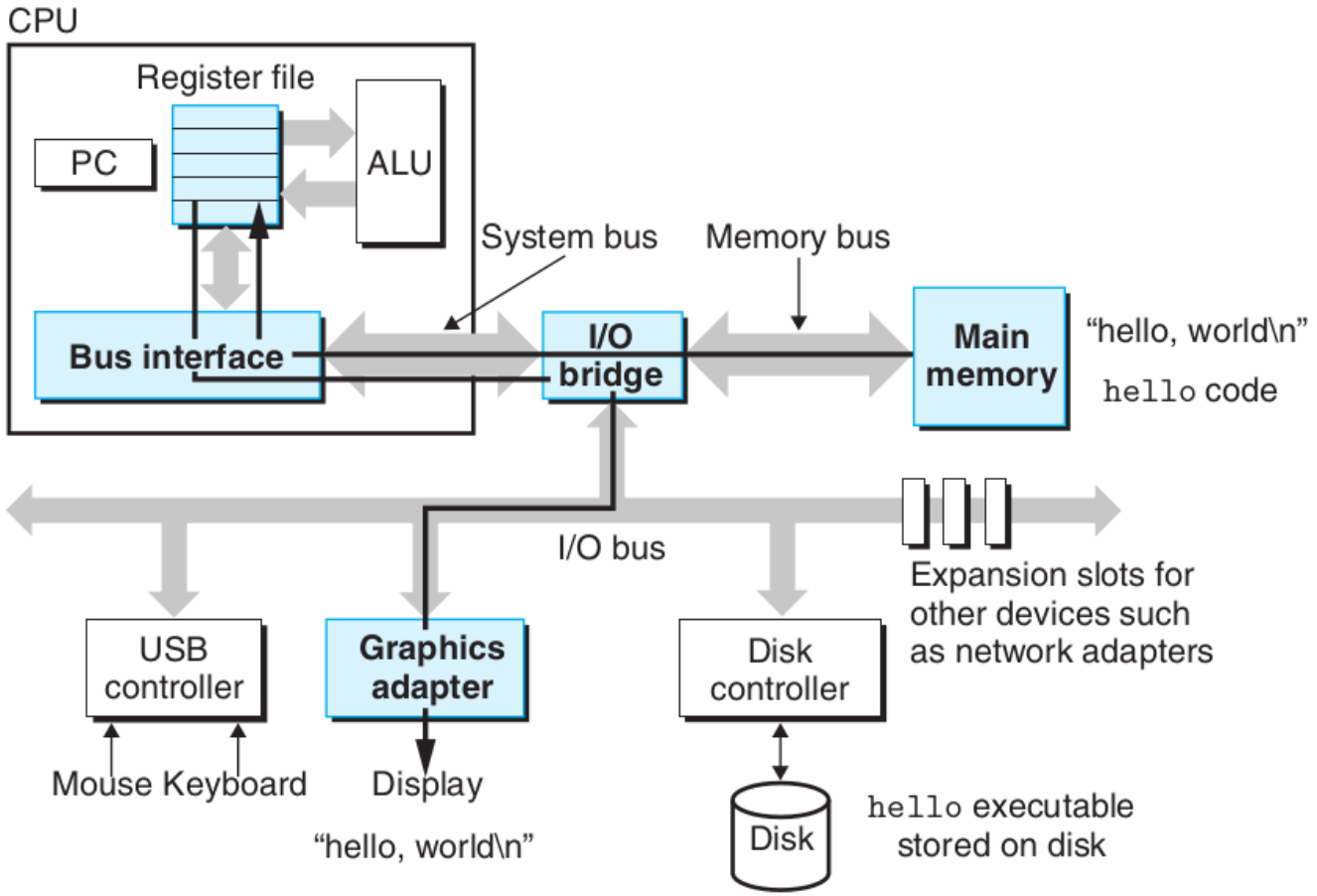
Running `./hello` (1/3): typing the program name



Running `./hello` (2/3): program copied from disk to memory



Running `./hello` (3/3): execute program from memory

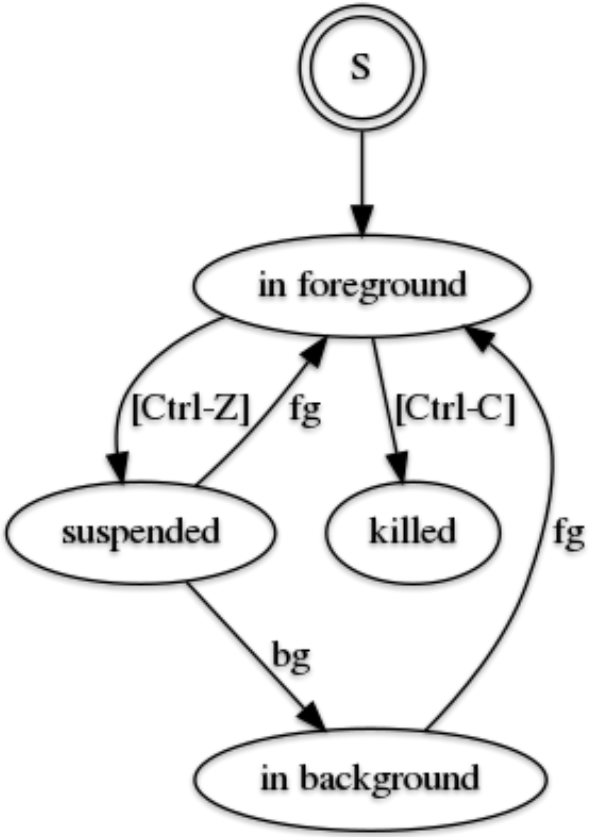


Programs and Processes

- Once a program is copied into the main memory of the computer and ready to be executed, we no longer call the program a program but a **process**
  - A single program stored on disk, if executed several times, can create several processes running at the same time
- Commands **`ps`** and **`jobs`** show all current processes in terminal.
  - Command **`ps aux`** shows all the processes currently running in the system (many of them are from the operating system itself)
- Each process has a number (PID, process ID). You can try and terminate some process using the **`kill`** command followed by the PID of the corresponding process.
  - If this is not enough (for instance, to terminate vi), use **`kill -9`**.

Controlling processes in the Linux terminal

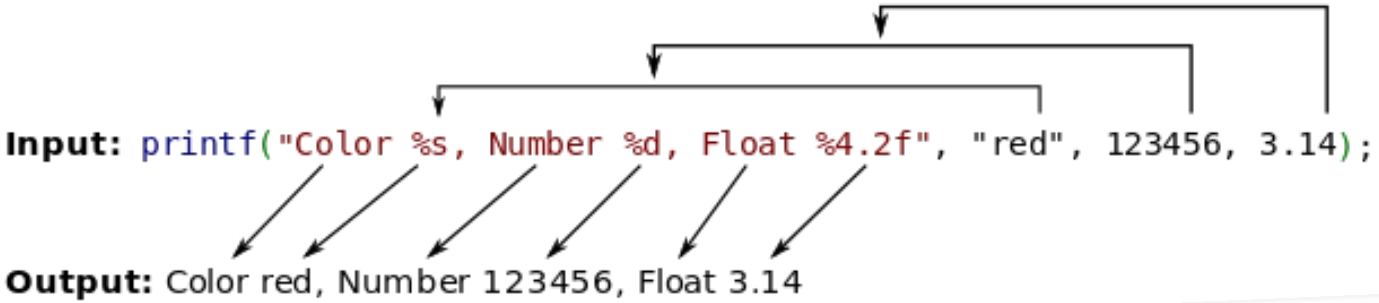
- `CTRL+C`: terminates process that is currently executing.
- `CTRL+Z`: suspends (pauses) current process. After that:
  - command **`fg`** resumes program execution in *foreground*
  - command **`bg`** resumes program execution in *background*
- **`ps`** and **`jobs`** show the processes of the current terminal
- try with the commands `sleep` and `watch`



Back to C programming: `putchar()`

- `putchar()` is a function similar to `printf()` but instead of taking a string constant, it takes a **character constant**
- Character constants are written `'c'` where c is some character
- Careful: `"c"` is a string constant (a string with only one character)
- Examples:
  - `putchar('a');`
  - `putchar('\n');`

Use of `printf()`



- `%d`, `%i`: prints an integer
- `%f`: prints a floating-point number
- `%s`: prints a string
- `%c`: prints a character
- `%%`: prints one % character
- `\n` prints a newline

Comments

- A **comment** is some text placed in the source code, that is *ignored* during the compilation process (not treated as normal source code)
- Useful to provide explanations/descriptions for developers
- Single-line comments start with the `//` characters until the end of the line
- Multi-line comments start with `/*` and end with `*/`

Exercise: what is the output of these programs?

```
main(){
  printf("GTI");
  /* printf("STU"); */
  printf("IT\n");
}
```

```
main(){
  /* printf("Ho"); */
  printf("\nCho");
  /* printf("la"); */
  putchar('o');
  /* printf("!\n"); */
  printf("se W");
  /* printf("man"); */
  printf("isel");
  putchar('y');
  printf("\n");
}
```

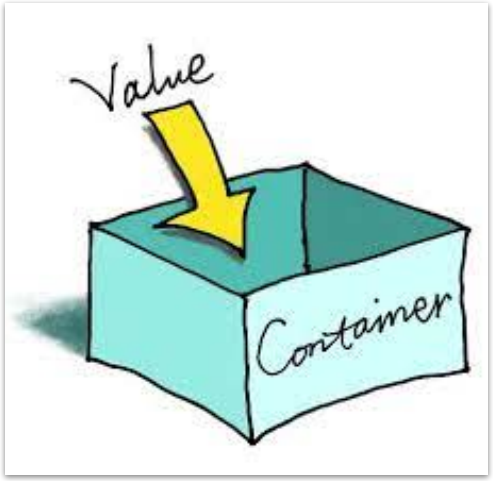
A one-liner:

```
main(){printf("Hey\n");}
```

Variables

A **variable** is the combination of the following:

- a name (or identifier) that you choose
- an address in the memory of the computer
- a value stored at that address
  - value may change during program execution
- a *type*, that specifies how much space this value occupies in memory, and how to interpret this value (signed/unsigned...)



To use a variable, you need first to **declare** it.

Variable declaration

A variable declaration is a statement of the following shape:

```
type name;
```

or:

```
type name = value;
```

The name should start with a lowercase letter. It may contain uppercase letters, numbers, underscore symbols. For instance:

```
a, b, c, x10, state, tagName, inputStr,  ...
```

Types

For now let us consider two types of the C language: **int** and **char**.

- The **int** type uses **4 bytes** in memory, it holds values from –2,147,483,648 to +2,147,483,64 (if considered signed) or from 0 to 4,294,967,295 (if considered unsigned).
- The **char** type uses **1 byte** in memory, it holds values from -128 to 127 or from from 0 to 255.

```
main() {
   int time = 100;
   char c = 50;
   ...
}
```

Assignments: changing a variable's value

An assignment is a statement of the form:

```
lvalue = expression;
```

Where `lvalue` (as in "left value") can be the name of a variable, but *cannot be a constant*.

The assignment changes the value of the `lvalue` variable to the value of `expression`

```
main() {
  int time = 100;
  printf("%d", time);   // will print 100
  time = 60;
  printf("%d", time);   // will print 60
}
```

Assignments from a constant or from a variable

```
main() {
  int time = 100;
  int another_time = 300;
  printf("%d", time);           // will print 100
  time = 60;                    // assignment from a constant
  printf("%d", time);           // will print 60
  time = another_time;          // assignment from a variable
  printf("%d", time);           // will print 300
}
```

A little riddle

```
main(){
  int time = 100;
  int another_time = 300;
  time = 60;
  another_time = 150;
  time = another_time;
  printf("%d", time);  // will print... ?
}
```

About assignments

- the assignment syntax *is not symmetric*: value gets copied from right to left
- You may find in some C code some assignments of the form:

```
a = b = 10;
```

- It is equivalent to doing `b = 10;` then `a = b;`.

Expressions

An expression can be built from:

- constants
- variables
- operators, including arithmetic and logic operators
- function calls (we will see them later)

Expression examples

- `10`
- `'c'`
- `varName`
- `10 + 20`
- `(30 * 5) / (44 - 24)`
- `(a * 5) / (44 - b) + 'c'`
- `45 % 10`
- operators `+`, `-`, `*` (multiplication), `/` (division), `%` (modulo) are **arithmetic operators**
- An expression has a **value**, calculated from its constants, variables and operators.

The Division (/) and Modulo (%) operators

- An expression of the form `x / y` has as value the **quotient** of `x` by `y`.

  `100 / 10` : value is 10

  `100 / 15` : value is 6

  `10 / 100` : value in 0

- An expression of the form `x % y` has value the **remainder** of the division of `x` by `y`:

  `100 % 15` : value is 10

  `123 % 10` : value is 3

Riddle

```
main(){
  int x = 100;
  int y = 300;
  x = y + 200;
  y = x / 5;
  x = x + y;
  printf("%d", x);  // will print... ?
}
```

Riddle

```
main(){
  int a = 5;
  int b = 7;
  int c = 8;
  a = b - c;
  b = a - c;
  c = a * b;
  printf("%d", c);  // will print... ?
}
```

Expressions in `printf` arguments

```
main(){
   int x = 100;
   int y = 300;
   y = y / 5;
   x = x + 40;
   printf("%d", x + y - 100);  // will print... ?
}
```

`scanf()`

- the `scanf()` function makes our programs pause to get user input
  - The user types a value and presses `ENTER` to confirm.
  - The received value is stored in a variable
- like `printf()`, it works with a string argument that contains placeholders, but it is a little trickier
- here is the syntax that we will use to ask for an integer and store it in the variable x:

```
scanf("%d", &x);
```

- Notice the **&** before the variable name

`scanf()` in action

```
main(){
    int age;
    printf("Please enter your age.\n");
    scanf("%d", &age);
    printf("Your age is %d\n", age);
}
```

`scanf()` in action, again

```
main(){
    int age, year;
    printf("Please enter your age.\n");
    scanf("%d", &age);
    printf("Please enter the current year.\n");
    scanf("%d", &year);
    printf("Your were born in the year %d\n", year-age);
}
```

Comparison operators

Comparison operators enable to compare the values of two expressions:

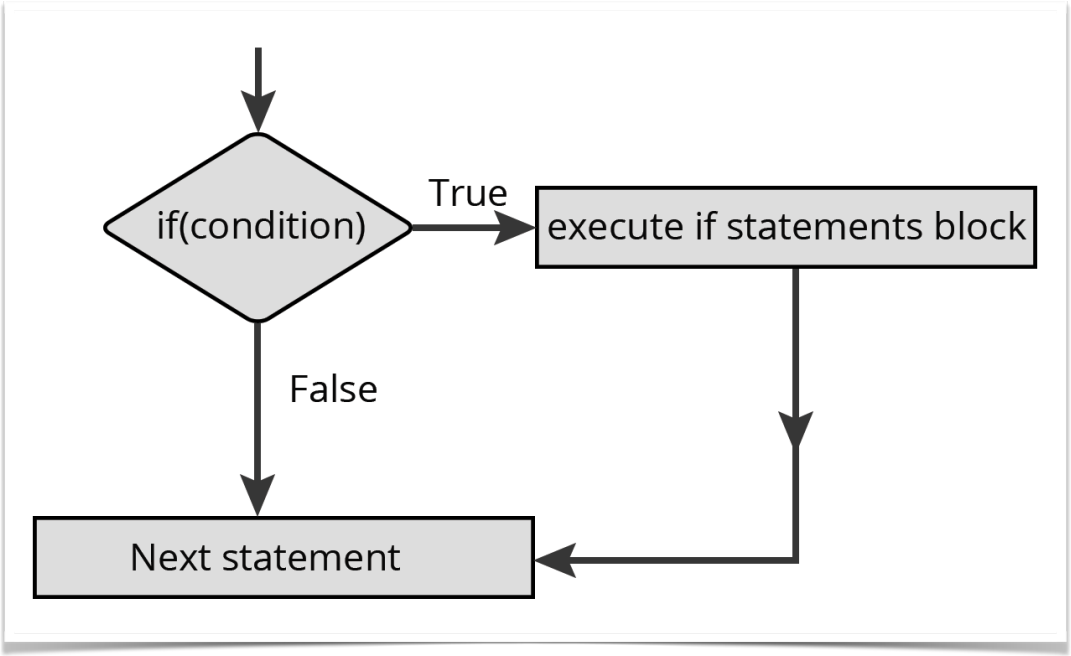
- `EXPR > EXPR`
- `EXPR >= EXPR`
- `EXPR < EXPR`
- `EXPR <= EXPR`
- `EXPR == EXPR`
- `EXPR != EXPR`

These boolean expressions, or conditions, can be used in the context of **conditional statements** and **loops**.

if statement

```
...
if (CONDITION)
   [STATEMENT]
[NEXT STATEMENTS]
...


if (CONDITION) {
   [STATEMENTS]
}
[NEXT STATEMENTS]
...
```



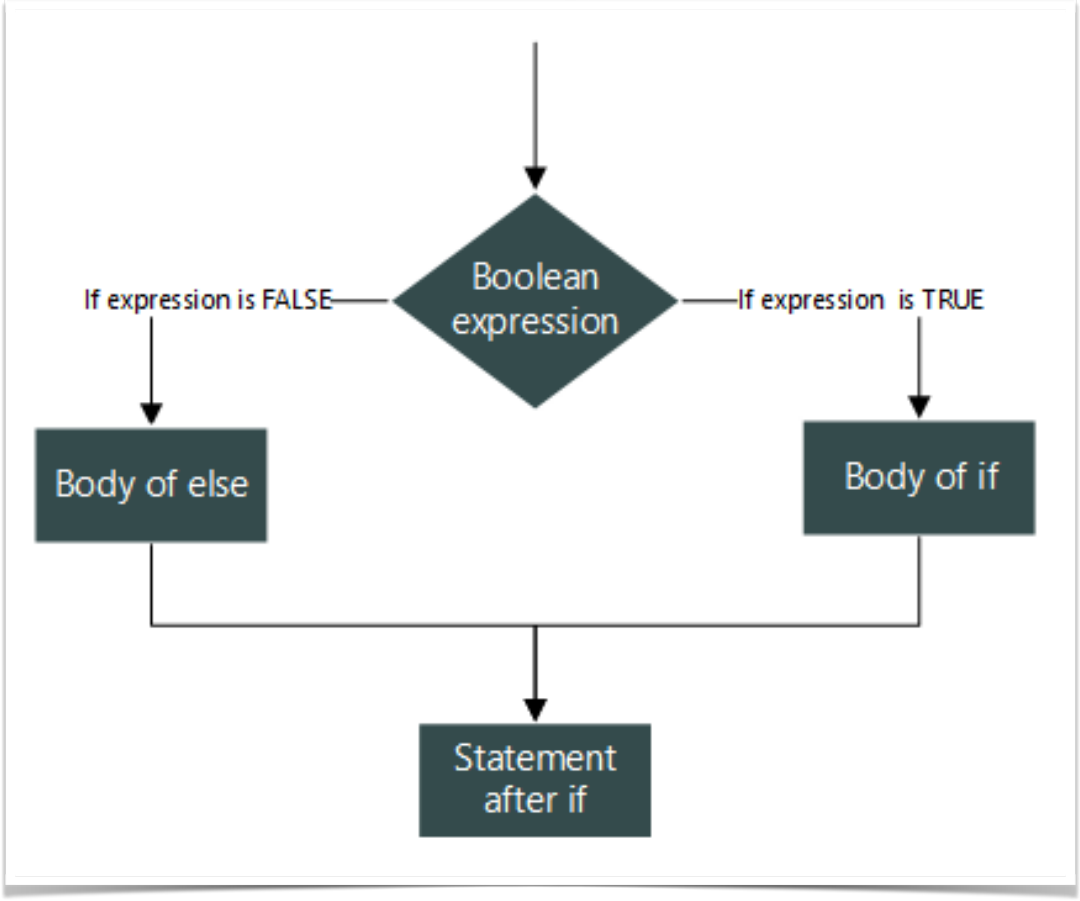
if statement example

```
main(){
    int age;
    printf("Please enter your age.\n");
    scanf("%d", &age);
    if (age < 18)
        printf("You are minor.\n");
    printf("Your age is %d.\n", age);
}
```

if-else statement

```
if (CONDITION)
   [STATEMENT]
else
   [STATEMENT]
[NEXT STATEMENTS]
...

if (CONDITION){
   [STATEMENTS]
} else {
   [STATEMENTS]
}
[NEXT STATEMENTS]
```



if-else statement example

```
main(){
    int age;
    printf("Please enter your age.\n");
    scanf("%d", &age);
    if (age < 18)
        printf("You are minor.\n");
    else
        printf("You are major.\n");
    printf("Your age is %d.\n", age);
}
```

Favor if-else over consecutive if's

```
// not great:
if (age < 18)
      printf("You are minor.\n");
if (age >= 18)
      printf("You are major.\n");
printf("Your age is %d.\n", age);
```

```
// much better:
if (age < 18)
      printf("You are minor.\n");
else
      printf("You are major.\n");
printf("Your age is %d.\n", age);
```

Lecture Summary

- Linux processes
- Variables, assignment
- `putchar()` and `printf()`
- `scanf()`
- Arithmetic operators
- Comparison operators
- if and if-else statements