

Introduction to C

Day 1

Katia Bulekova

Research Computing Services



Schedule

9:30 – 10:30

10:30 – 10:45 – coffee break

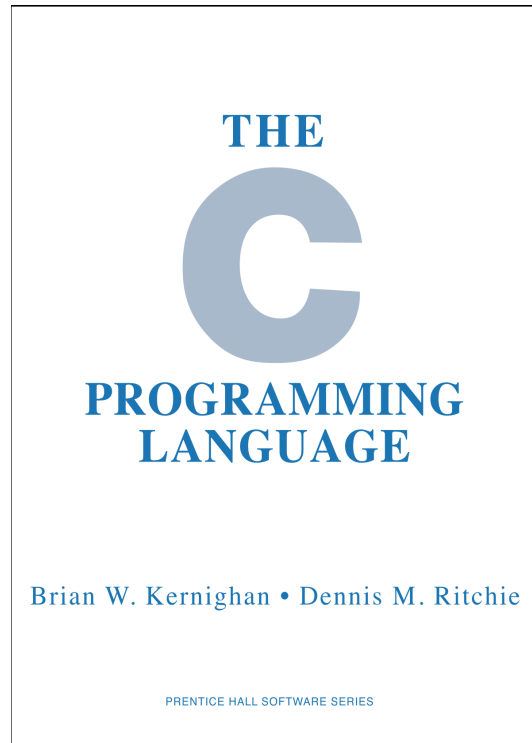
10:45 – 11:45

11:45 – 12:00 – break

12:00 – 13:00

C History

Developed by Dennis Ritchie at Bell Labs in 1969–1973



Official ANSI standard published in 1969 (“C 89”) and updated in 1999 (“C99”)

In 1985 Bjarne Stroustrup (Bell Labs) published C++ (“C with classes”)

Where is C used

- Operating Systems (Linux, Apple's OS X, Microsoft Windows)
- Databases (MySQL and others)
- Browsers (Google's Chromium)
- Adobe applications
- Many other desktop applications

Useful resources

- Brian Kernighan and Dennis Ritchie, *The C Programming Language*
- The C Language Specification:
<http://www.open-std.org/JTC1/SC22/WG14/www/docs/n1256.pdf>
- Learn C in minutes:
<https://learnxinyminutes.com/docs/c/>
- Online Tutorial (guru99.com):
<https://www.guru99.com/c-programming-language.html>

Compiled vs. Interpreted Languages

Interpreted Languages: Matlab, Python, R, Stata, SAS, etc.

Advantages:

- Interactive
- Allows fast code development

Disadvantages:

- Uses more CPU and RAM
- Slower for a given task

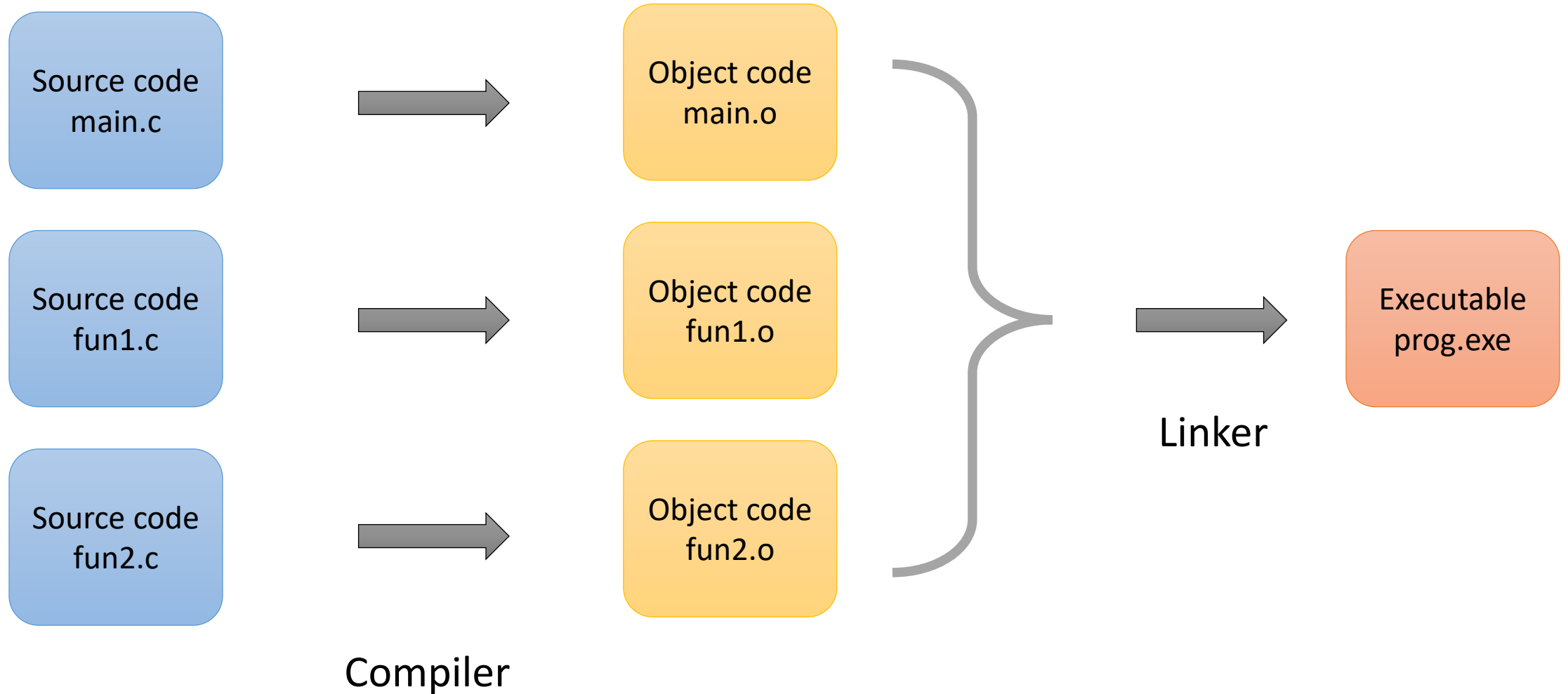
Compiled vs. Interpreted Languages

Compiled Languages: C, C++, FORTRAN, etc.

Source code is written using a text editor;

Source code then must be processed through **compiler**.

Big picture



C compiler for Windows and Mac

Windows:

Codeblocks: <http://www.codeblocks.org/downloads>

Mac:

<https://developer.apple.com/downloads/>

See <https://www.guru99.com/c-gcc-install.html> for directions

Get bootcamp materials

Copy **bc_day1.zip** file from RCS examples webpage

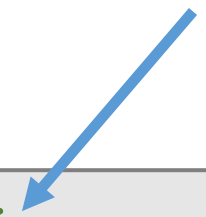
```
$ cp /project/scv/examples/c/bootcamp/bc_day1.zip .  
$ unzip bc_day1.zip
```

Almost a C program: *bc_00.c*

```
/*  
 * bc_00.c  
 *  
 * Date:           May 2022  
 */  
  
// This function has no input and returns no value  
void main() {  
}
```

Almost a C program: *bc_00.c*

Multiline comments




```
/*  
 * bc_00.c  
 *  
 * Date:           May 2022  
 */  
  
// This function has no input and returns no value  
void main() {  
}
```

Almost a C program: *bc_00.c*

Single line comment

```
/*  
 * bc_00.c  
 *  
 * Date:           May 2022  
 */  
  
// This function has no input and returns no value  
void main() {  
}
```



Almost a C program: *bc_00.c*

Compiling and Running:

```
$ gcc bc_00.c  
$ ./a.out  
$
```

First C program: *bc_01_hello.c*

```
#include <stdio.h>

int main() {
    printf("Hello, world\n");
    return 0;
}
```

First C program: *bc_01_hello.c*

Include macro (we will return to it later)



```
#include <stdio.h>

int main() {
    printf("Hello, world\n");
    return 0;
}
```


First C program: *bc_01_hello.c*

Declaration of function “main”, returning type “int”

Every C program contains at least one function – main()

```
#include <stdio.h>

int main() {
    printf("Hello, world\n");
    return 0;
}
```



First C program: *bc_01_hello.c*

Each statement must end with a semicolon

```
#include <stdio.h>

int main() {
    printf("Hello, world\n");
    return 0;
}
```



First C program: *bc_01_hello.c*

Function used for printing

```
#include <stdio.h>

int main() {
    printf ("Hello, world\n");
    return 0;
}
```

First C program: *bc_01_hello.c*

Return the value 0

```
#include <stdio.h>

int main() {
    printf("Hello, world\n");
    return 0;
}
```



First C program: *bc_01_hello.c*

Compiling and Running:

```
$ gcc bc_01_hello.c -o hello
$ ./hello
Hello, world
$
```

First C program: *bc_01_hello.c*

Hands-on exercises:

1. Delete one of the semicolons and try to compile the code. What error messages do you get?
2. Add another `printf()` function. Check what output you get if you do not use “`\n`” symbol at the end of the string.

First C program (a closer look): *bc_01_hello.c*

Where does `printf()` function come from?

```
#include <stdio.h>

int main() {
    printf("Hello, world\n");
    return 0;
}
```

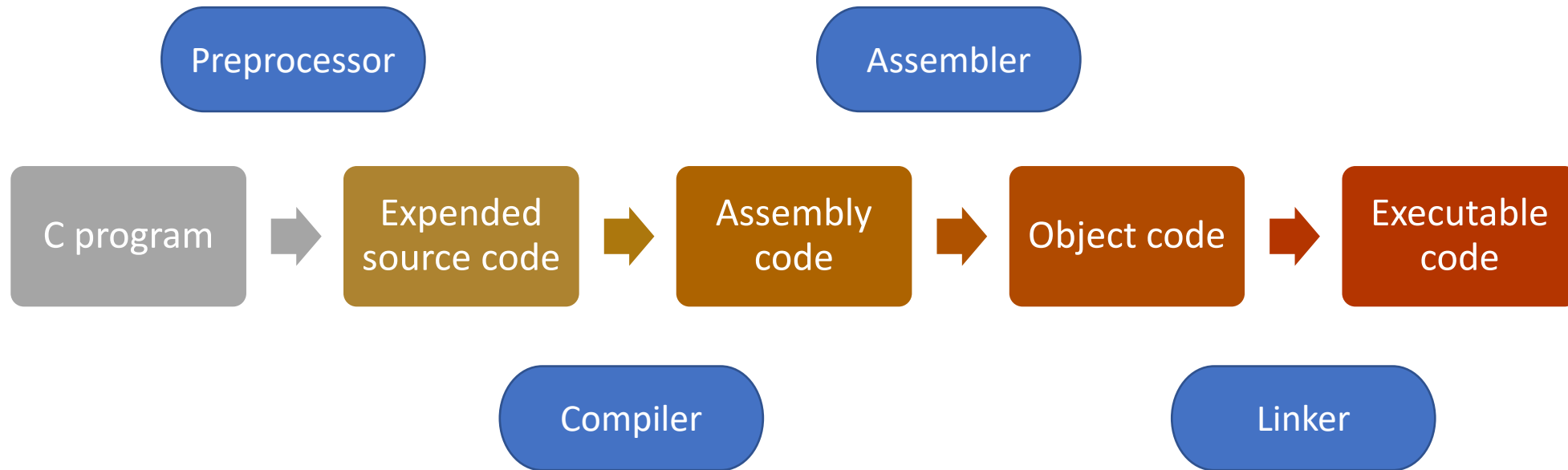
First C program (a closer look): *bc_01_hello.c*

The preprocessor replaces the `#include` macro with the contents of `stdio.h` file which contains the declaration of `printf()` function

```
#include <stdio.h>

int main() {
    printf("Hello, world\n");
    return 0;
}
```


First C program (a closer look)



First C program (a closer look)

Preprocessor:

The source code is first passed to the preprocessor which expands the code;

Compiler:

converts the code into assembly code;

Assembler:

assembly code is converted to the object code

First C program

Expand code using preprocessor

```
$ gcc -E bc_01_hello.c
```

The expanded code is almost a thousand lines long!

First C program

Assembly code:

```
$ gcc -S bc_01_hello.c  
$ less bc_01_hello.s
```

Compile to generate an object code:

```
$ gcc -c bc_01_hello.c
```

First C program

Link object together with system libraries

```
$ gcc -o bc_01_hello bc_01_hello.o
```

Variables in C

All variables in C must be declared

```
#include <stdio.h>
int main() {
    float tc = 100.0;
    float tf;
    tf = 9.0/5.0 * tc + 32.0;
    printf("%f celcius = %f fahrenheit\n", tc, tf);
}
```

Variables in C

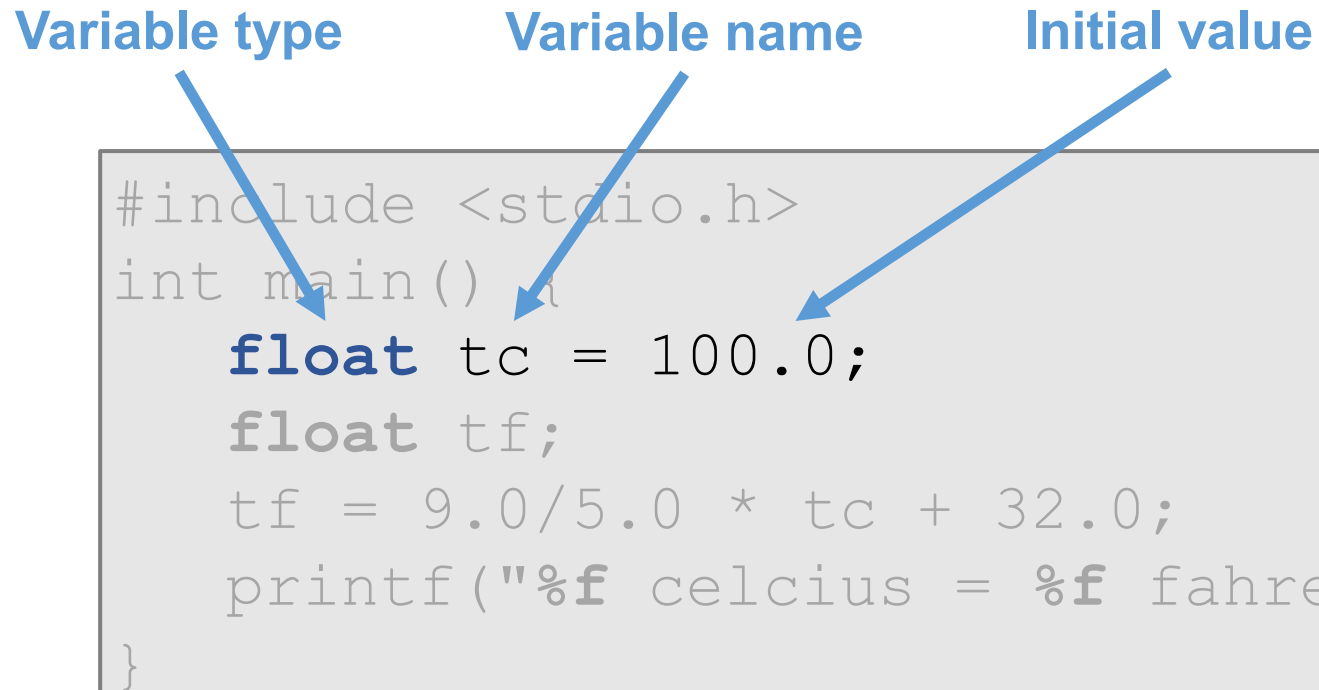
All variables in C must be declared

Variable type

Variable name

Initial value

```
#include <stdio.h>
int main() {
    float tc = 100.0;
    float tf;
    tf = 9.0/5.0 * tc + 32.0;
    printf("%f celcius = %f fahrenheit\n", tc, tf);
}
```



Variables in C

1. Primitive data types

- **int** for integer
- **char** for character
- **float** for single precision floating numbers
- **double** for double precision floating numbers
- **void**

2. Derived data types

3. User-defined data types

There is no “*boolean*” (or logical) data type in C

Variables in C

| Type | Storage size | Print format | Value range |
|----------------|--------------|--------------|--|
| char | 1 byte | %d | -128 to 127 or 0 to 255 |
| unsigned char | 1 byte | %d | 0 to 255 |
| signed char | 1 byte | %d | -128 to 127 |
| int | 2 or 4 bytes | %d | -32,768 to 32,767 or -2,147,483,648 to 2,147,483,647 |
| unsigned int | 2 or 4 bytes | %u | 0 to 65,535 or 0 to 4,294,967,295 |
| short | 2 bytes | %d | -32,768 to 32,767 |
| unsigned short | 2 bytes | %d | 0 to 65,535 |
| long | 8 bytes | %ld | -9223372036854775808 to 9223372036854775807 |
| unsigned long | 8 bytes | %lu | 0 to 18446744073709551615 |
| float | 4 bytes | %f or %g | 1.2E-38 to 3.4E+38 (6 decimal places) |
| double | 8 bytes | %f or %g | 2.3E-308 to 1.7E+308 (15 decimal places) |
| long double | 10 bytes | %lf | 3.4E-4932 to 1.1E+4932 (19 decimal places) |

Comments in C

`/*` classic C comments
can be use over multiple lines `*/`

`//` C++ comments; only used for comments on a single line

```
#include <stdio.h>
int main() {
    float tc = 100.0; /* temperature in celcius */
    float tf;
    tf = 9.0/5.0 * tc + 32.0;
    printf("%f celcius = %f fahrenheit\n", tc, tf);
}
```

C program: *bc_02_vars.c*

Hands-on exercises:

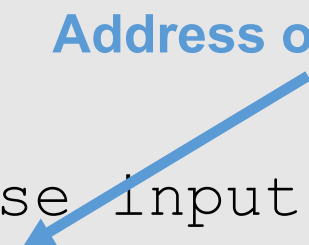
1. Copy `bc_02_vars.c` code to `ex_02.c`
2. Define integer variables **year1** and **year2** and assign them some values. For example, a year when you graduated from school and the current year. Calculate the difference and print it.

Reading in values: *bc_03_read.c*

```
#include <stdio.h>
int main() {
    float tc;
    float tf;           Address of operator

    printf("Please input temperature in celcius: ");
    scanf("%f", &tc);

    tf = 9.0/5.0 * tc + 32.0;
    printf("%f celcius = %f fahrenheit\n", tc, tf);
}
```



Reading in values

Hands-on exercises:

1. Copy `bc_03_read.c` to `ex_03.c`
2. Define integer variables **year1** and **year2** and read them using **scanf()** function. Do not forget **&** symbol.

Arithmetic Operators

| Operator | Description | Example |
|----------|----------------------------------|---------------|
| + | addition | $y = x + 5;$ |
| - | subtraction | $y = x - 3;$ |
| * | multiplication | $y = x * 25;$ |
| / | division | $y = x / 2$ |
| % | remainder after integer division | $j = i \% 2$ |
| ++ | increment by one | $j = i++$ |
| -- | decrement by one | $j = i--$ |

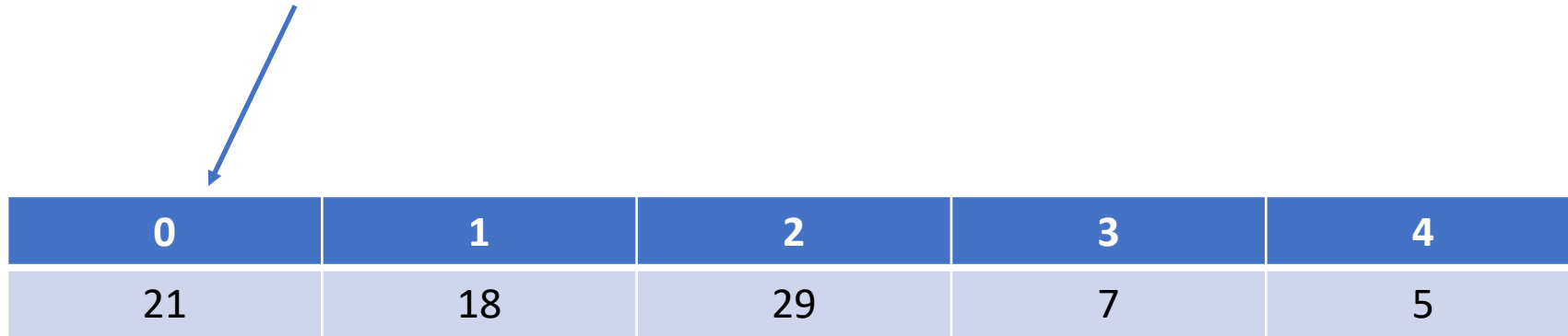
Arithmetic Operators

Hands-on exercises:

1. Use arithmetic operators to compute value k.
2. Print value of k.

Arrays

Array indices start at **zero**!



| 0 | 1 | 2 | 3 | 4 |
|----|----|----|---|---|
| 21 | 18 | 29 | 7 | 5 |

Size of array is 5

Arrays

Declare arrays using square brackets: []

Examples:

```
float x[3];  
x[0] = 0.5;  
x[1] = -0.3;  
x[2] = 2.5;
```

or

```
float x[] = {0.5, -0.3, 2.5}
```

Arrays

Hands-on exercises:

1. Declare an array with 3 elements
2. Define values for all 3 elements and print them.
3. Try to access `x[3]` element.

Arrays review: Out of bounds example

```
int main() {  
    int x[5] = {1, 2, 3, 4, 5};  
  
    x[10] = 11;  
    printf(" x[10] = %d\n", x[10]);  
  
    return 0;  
}
```

Arrays review: Out of bounds example

Compile and run:

```
$ gcc -o oob bc_05_array.c
```

```
$ ./oob
```

Strings

A C string is an array of characters

The last element in the string array must be a NULL character – '\0'

```
char my_string[] = "Hello";
```

or

```
char my_string[] = {'H', 'e', 'l', 'l', 'o' , '\0'};
```

Either way this string has length 5.

Note, that in the first example we use double quotes and in the second – single quotes.

Strings

Hands-on exercises:

There are many string functions declared in `string.h` header file.

For example, there are functions

`strcpy()` – copy string

`strcat()` – append one string to another

See https://en.wikibooks.org/wiki/C_Programming/string.h for a list of other string functions

Control Flow: **if**

```
if (condition) {  
    // statements to be executed if condition is true  
}
```

```
float x = 20.;  
float y;  
  
if ( x >= 0 ) {  
    y = sqrt(x);  
}
```

Control Flow: **if-else**

```
if (condition) {  
    // statements to be executed if condition is true  
} else {  
    // statements to be executed if condition is false  
}
```


Control Flow: `if-else`

```
float x = 20.;  
float y;  
  
if ( x >= 0 ) {  
    y = sqrt(x);  
} else {  
    y = sqrt(-x);  
}
```

*Note: `sqrt()` function declaration is in `<math.h>` header file

Control Flow: `if-else`

Since we use a function `sqrt()` that comes from a math library, We need to add a path to the system library where this function is implemented. It will be used by the linker:

```
$ gcc -lm -o bc_07_if bc_07_if.c
```

Control Flow: `if-else`

Hands-on exercises:

1. Add an `else` clause to the `if()` statement
2. For a negative value of `x`, compute `sqrt(-x)`
3. In the `print` statement add `i` letter to indicate imaginary number

Control Flow: **for** loop

```
for (init; condition; update) {  
    // statements  
}
```

Control Flow: **for** loop

```
for ( ; ; ){  
    printf("This loop will run forever\n");  
}
```

Control Flow: `for` loop

initialization

condition

update

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

Side note

Operators ++ and --

++ // increase value by 1

-- // decrease value by 1

```
int i, j;

i = 5;
j = i++;
printf( "i=%d  j=%d\n", i, j);

j = ++i;
printf( "i=%d  j=%d\n", i, j);
```

Side note

Operators +=, -=, *=, /=

a += b is the same as **a = a + b**

a -= b is the same as **a = a - b**

Control Flow: for loop

```
int i;
for ( i = 0, dotprod=0. ; i < 3 ; i++ ){
    dotprod += x[i] * y[i];
}
```

Compile:

```
$ gcc -lm -o bc_08_for bc_08_for.c
```

Control Flow: `for` loop

Hands-on exercises:

Let's modify the code and instead of calculating a dot product, we will calculate a unit vector given input vector x :

1. Find length of a vector $\sqrt{x[1]^2 + x[2]^2 + x[3]^2}$
2. If the length is greater than 0, divide the input vector x by its length