Introduction to C Programming

Ignas Brazauskas

https://github.com/Ignas207/robotics-materials

What do computers do?

- Anything you tell them to do, no questions asked!

How to make a computer do something?

- Quite simple!

Just tell it what to do in a format it understands!

```
0100 1000 0000 0000 1000 0011 0000 0000 1110 1100 0010 1000 0000 0000 1110 1000 0000 0111 0000 0000
1110 1010 0000 0000 1111 1111 0000 0000 1111 1111 0100 1000 0000 0000 1000 1101 0000 1101 0000 0000
0001 0100 0000 0000 0000 0000 0000 0000 1110 1000 0010 1011 0000 0000 1111 1110 0000 0000 1111 1111
0000 0000 1111 1111 0011 0001 0000 0000 1100 0000 0100 1000 0000 0000 1000 0011 0000 0000 1100 0100
0010 1000 0000 0000 1100 0011 0000 0000 1001 0000 0000 0000 1001 0000 0000 0000 1001 0000
```

- *This is a disassembly of a **Hello World** program if you can't tell.*
- Okay, this may not be that simple...

Fortunately, we don't have to do this as **compilers do this for us**

Here is the same code

- Human readable **source code**

```c
#include <stdio.h>

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

- **We work on this!**

- Machine-readable **machine code**

```
0100 1000 0000 0000 1000 0011
0000 0000 1110 1100 0010 1000
0000 0000 1110 1000 0000 0111 ...
```

- **We run this!**

We need to comunicate our intent to the compiler

That's a programming language!

As with human languages, programming languages have rules that we must follow!

Statements end with ;

```
printf("Hello World!\n");
```

```
return 0;
```

We group items with {}

```
if(number == 2)
{
    number++;
    printf("The number is now %d!\n", number);
}
```

Variable names must be unique and not overlap!

- This **won't** work!

```
int number = 0;
int number = 12;
```

- This will work!

```
int number = 0;
int number2 = 12;
```

Variable datatypes and sizes

- `int` - integers *usually* $[-2147483647 \ldots 2147483647]$
- `char` - characters A, B, C, etc and integers $[-127 \ldots 127]$
- `float` and `double` - real numbers: $3,141592654$

Variable datatypes continued

- We can extend our bounds with the **unsigned** prefix.
- `unsigned int` - integers *usually* $[0 \ldots 4294967296]$
- `unsigned char` - characters, integers $[0 \ldots 255]$
- We can only do this with **integer types**
  - *char, short, int, long*

Arithmetic

- From **right** to **left**.

```
int c = 10;
int a = 3;
int b = 5;

// c is now 3 + 5 => 8
c = a + b;
```

Arithmetic shorthand forms

Normal

```
int c = 10;
int a = 3;

// c is now 3 + 10 => 13
c = a + c;

// c is now 13 + 1 => 14
c = c + 1;

// c is now 14 + 1 => 15
c = c + 1;
```

Shorthand

```
int c = 10;
int a = 3;

// c is now 3 + 10 => 13
c += a;

// c is now 13 + 1 => 14
c += 1;

// c is now 14 + 1 => 15
c++;
```

Logical operations

- `a == b` Is it equal?
- `a != b` Is it not equal?
- `a < b` Is a *smaller* than b
- `a && b` logical AND
- `a || b` logical OR
- `!` logical NOT

`False == 0` and `True == 1`

Bit operations

- `a & b` bitwise AND
- `a | b` bitwise OR
- `a ^ b` bitwise XOR
- `~a` bitwise INVERSION
- `a >> b` bitwise RIGHT SHIFT
- `a << b` bitwise LEFT SHIFT

Conditionals if

- If (**condition == True**) { Do this }

```c
int number = 0;

// Explicit condition
if (number == 0)
{
    printf("Explicit: The number is zero!\n");
}

// Implicit condition
if(!number)
{
    printf("Implicit: The number is zero!\n");
}
```

Conditionals else

- If (**condition == True**) { Do this }
- Else { Do this other thing}

```
int number = 1;

if (number == 0)
{
    printf("The number is zero!\n");
}
else
{
    printf("The number is one!\n");
}
// Output: The number is one!
```

Conditionals if + else

```c
int number = 5;

if (number == 0)
{
    printf("The number is zero!\n");
}
else if (number == 1)
{
    printf("The number is one!\n");
}
else
{
    printf("The number is neither one or zero!\n");
}
// Output: The number is neither one or zero!
```

Conditionals switch statement

```c
int number = 5;

if (number == 0)
{
    printf("The number is zero!\n");
}
else if (number == 1)
{
    printf("The number is one!\n");
}
else
{
    printf("The number is neither one or zero!\n");
}
// Output: The number is neither one or zero!
```

```c
int number = 5;

switch (number)
{
    case 0:
        printf("The number is zero!\n");
        break;

    case 1:
        printf("The number is one!\n");
        break;

    default:
        printf("The number is neither one or zero!\n");
        break;
}
// Output: The number is neither one or zero!
```

- **No logical operations!**
- **Don't forget the `break`**

Loops while

- While (**Condition == True**) { Do this }

```c
int i = 0;
while (i < 10)
{
    printf("%d ", i);
    i++;
}
puts("");
// Output: 0 1 2 3 4 5 6 7 8 9
```

Loops do while

- {Do this} while (**Condition == True**)

```
int i = 0;
do
{
    printf("%d ", i);
    i++;
}
while (i < 10);
puts("");
// Output: 0 1 2 3 4 5 6 7 8 9
```

Loops for

- for({Iterator setup}; {**Condition == True**}; {Iterator operation})

```c
int i = 0;
for(i = 0; i < 10; i++)
{
    printf("%d ", i);
}
puts("");
// Output: 0 1 2 3 4 5 6 7 8 9
```

We declare arrays with []

```
int arr[10]; // Declared array of size 10
int arr2[10] = {0}; // Zero initialized array
char emptyText[10] = {0};
char asignedText[] = "Hello world!"; // Automaticaly determined size

char asignedText2[13] = "Hello world!";
```

- Important to leave enough room in the array for our data!

```
char thisIsProblematic[10] = "Hello world!";
// Output: Hello worl
```

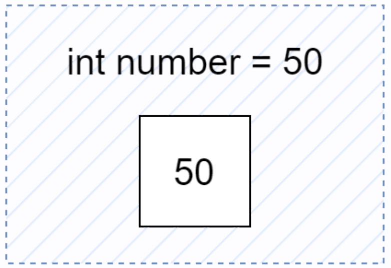
Arrays example

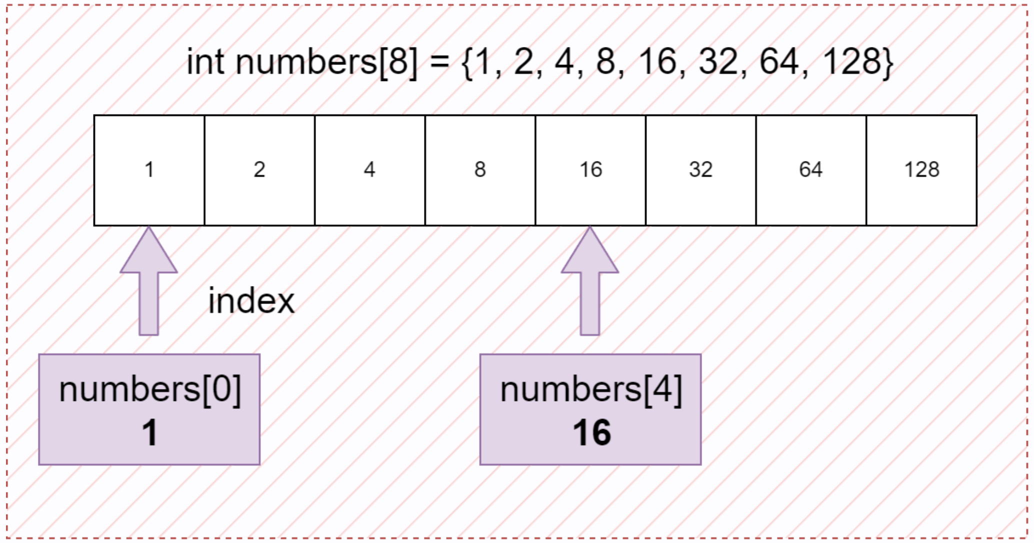
```c
#include <stdio.h>

#define ARRAY_SIZE 8

int main(void)
{
    int numbers[] = {1, 2, 4, 8, 16, 32, 64, 128};

    for (int i = 0; i < ARRAY_SIZE; i++)
    {
        printf("%d, ", numbers[i]);
    }
    printf("\n");
    return 0;
}
// Output: 1, 2, 4, 8, 16, 32, 64, 128,
```





Printing to a terminal

Printing printf

- Print a formatted string to the terminal.
- `%` - format specifier
  - `%d` - integers
  - `%f` - float, double
  - `%s` - string

```
int lenght = 12;
float pi = 3.14;
char helloText[] = "Hello!";

printf("Lenght is %d, pi is %f, text: %s\n", lenght, pi, helloText);
// Output: Lenght is 12, pi is 3.140000, text: Hello!
```

Getting user input scanf

- Users input in a specified format
  - `%` - format specifier

```c
int number = 0;
scanf("%d", &number);
printf("Number is %d\n", number);

char userInput[10] = {0};
scanf("%9s", userInput);
printf("User input is %s\n", userInput);
/*
Input: 120
    Output: Number is 120
Input: hello
    Output: User input is hello
*/
```

Oditties with scanf

When **NOT using** strings, we pass the variable with `&`

```c
int number;
float pi;
char letter;
scanf("%d", &number);
scanf("%f", &pi);
scanf("%c", &letter);
```

- With stings, we **DON'T NEED** the `&`

```c
char userInput[10];
scanf("%9s", userInput);
```

- *Don't forget to leave 1 extra space in strings!*

Functions in C

Juicy stuff!

Functions syntax

[return datatype] functionName(input parameters)

```
int Multiply(int multiplicant, int multiplier)
{
    int result = multiplicant * multiplier;
    return result;
}
```

- Usage:

```
int result = Multiply(5, 3);
```

Functions return type

- Functions **can return only *1 item!***
- Can be any type
    - `int`, `short`, `float`, `uint8_t`, etc...
- If the datatype is `void`, nothing is returned!

Functions C oddities

- Before we can use a function, we need a **prototype**
  - *How the function "works"*
- Function prototype

```c
int Multiply(int multiplicant, int multiplier);
```

- Function body

```c
int Multiply(int multiplicant, int multiplier)
{
    int result = multiplicant * multiplier;
    return result;
}
```

Functions main

- In C a program starts from the `main` function.
- This function should be an `int` type and `return 0`.
- **Doesn't** need a prototype!

```c
int main(void)
{
    /* Program logic */
    return 0;
}
```

Calculator example

```c
#include <stdio.h>

int Multiply(int multiplicant, int multiplier);

int main(void)
{
    int first, second;
    printf("Enter multiplicant: ");
    scanf("%d", &first);

    printf("Enter multiplier: ");
    scanf("%d", &second);

    int result = Multiply(first, second);
    printf("Result is %d!\n", result);
    return 0;
}

int Multiply(int multiplicant, int multiplier)
{
    int result = multiplicant * multiplier;
    return result;
}
```

Header files

Extra functionality!

Header files

- A way to include functions from other libraries, and better organize our project
- Include from a library: `#include <[header name].h>`

A few examples previously shown:

```
#include <stdio.h> // Standart library printf, scanf
#include <string.h> // String operations, strcpy, etc...
```

- *More can be found:* *https://en.cppreference.com/w/c/header*

#define directive

- Make the code more readable by eliminating **magic numbers!**
- `#define` just **replaces our text** in place!

```
for (int i = 0; i < 10; i++)
{
    // do something ...
}
```

```
#define MAX_COUNTER 10

for (int i = 0; i < MAX_COUNTER; i++)
{
    // do something ...
}
```

- These will compile the same result!

Aditional resources

- https://blue.pri.ee/ttu/programming-i/labs/
- https://www.learn-c.org/

Now an exercise

- Make a simple **calculator**!
  - Takes the users input, **verify it!**
  - *Two numbers*
- And do *math with those numbers!*
- Implement the following functions:
  - Addition
  - Multiplication
  - Powers **with loops!**
  - etc..
- ***Fancy printing!***

Additional exercise: string manipulation

- `#include <string.h>`
    - https://cplusplus.com/reference/cstring/
    - https://www.learn-c.org/en/Strings
- Get a string as a *user input*
    - Copy this into a *buffer*, pick your size!
    - Get another string and *concatinate them*
    - `strncat`
- Get a string as a user input, Check if this string matches a defined "*password*"
    - `strncmp`

Why to use srtn[cat, cpy, etc...] functions?

- `strcpy(char * destination, const char * source)`
  - *Copies a string from **destination** to **source***
- `strncpy(char * destination, const char * source, size_t num);`
  - *Copies **num charactess** from **destination** to **source***
- By using str**n** functions, we avoid **buffer overflows**!
  - We declare the **maximum number** of characters to modify!