

# Programing basics on C

Lecture 2

# Schedule

- Variables
- Variable types (basic)
- Arithmetic operations
- Scanning user input

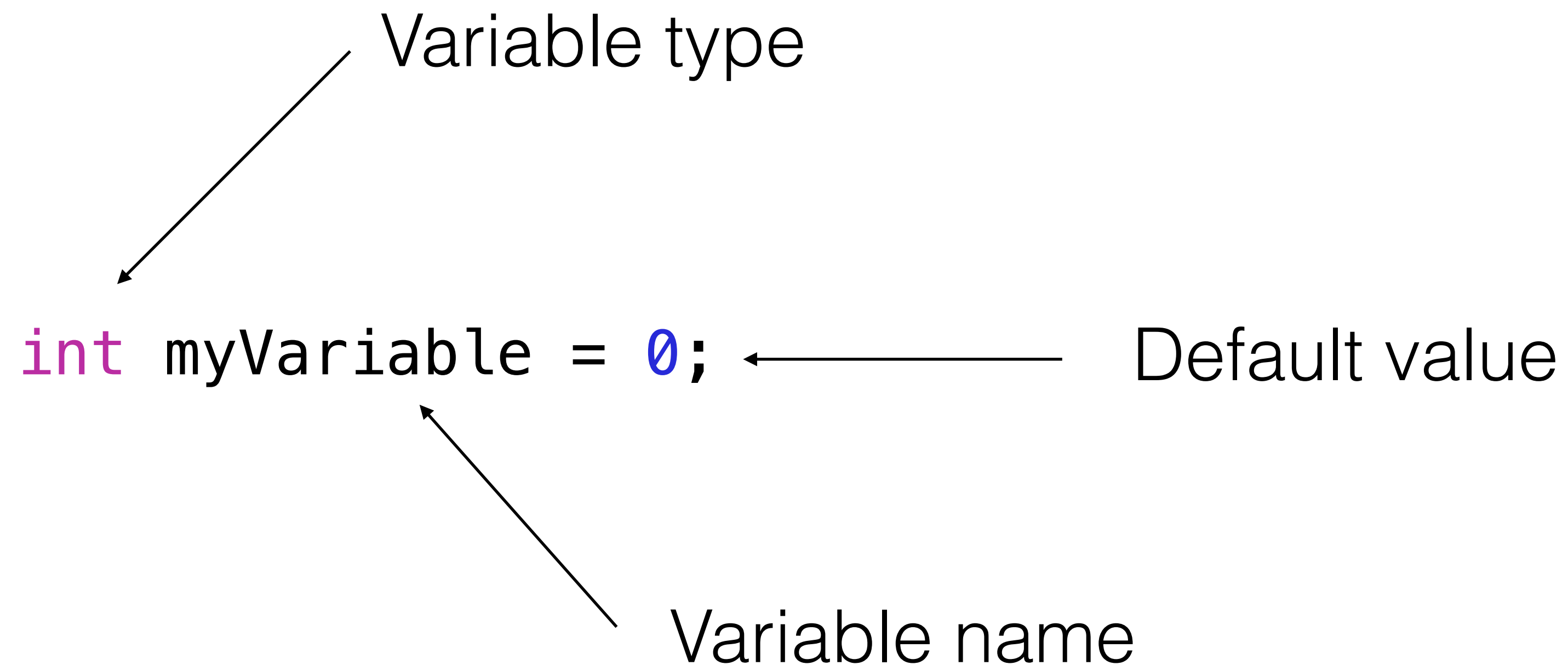
# Variables

**Declare variables,**  
not war.

# Variables

- Data that is being used in a software with certain name and type.
- Upon “declaring” the developer is giving unique name (in the current scope) to the data and a type.
- In the process of declaring the developer is not obligated to provide a value to the variable.
- It is recommended to provide default value for the variable, because not doing this could lead to code anomalies.

# Declaring a variable



Variable type

`int` myVariable = 0;

Default value

Variable name

The diagram illustrates the components of a variable declaration in C. The code snippet `int myVariable = 0;` is shown. Three arrows point from labels to specific parts of the code: 'Variable type' points to `int`, 'Variable name' points to `myVariable`, and 'Default value' points to `0`. The `int` is colored purple, and the `0` is colored blue.

# Declaring multiple variables

```
int myVariable0 = 0;  
int myVariable1 = 1;  
int myVariable2 = 22;  
int myVariable3 = 63;  
int myVariable4 = 94;  
int myVariable5 = 2;  
int myVariable6 = 988;  
int myVariable7 = 18476;
```

# Printing variables of type int

```
int main() {  
  
    int myVariable0 = 0;  
    int myVariable1 = 1;  
    int myVariable2 = 22;  
  
    printf("%d %d %d\n", myVariable0, myVariable1, myVariable2);  
  
    return 0;  
}
```

# Variables without default value

```
int main() {  
  
    int myVariable0;  
    int myVariable1;  
    int myVariable2;  
  
    printf("%d %d %d\n", myVariable0, myVariable1, myVariable2);  
  
    return 0;  
}  
  
// result : 24630 32767 1606415696
```



# Variable types

- Task :
- Try to declare a variable for the number 2 147 483 647. Print that variable
- Task time = 3 min.

What is the result ?

# Variable types

- Task:
- Try to declare a variable for the number  $2\ 147\ 483\ 647 + 3$ . Print that variable
- Task time : 2 min

What is the result ?

The result is WTF ??!

# Variable types

- In programming the variables have different types.
- When you are declaring a variable you will be using it in a certain way, that is the reason why there are different types.
- Another reason is because you must preserve space and processing power.

# Variable types

- Each variable type has certain amount of bytes that it is using. Most of the variable types has predefined number of bytes they are using, others depend on the processor architecture.

# Variable types

- Int
- The most used variable type in programming.
- Size - depends on the processor architecture (x16 - 2 bytes, x32 - 4 bytes, x64 - 8 bytes (x64 must be defined in the compiler))
- Standard bounds for x32 = -2 147 483 648 to 2 147 483 647
- Printed with “%d”



# Variable types

- short
- Rarely used in standard development. More frequently used in embedded development.
- Size = 2 bytes
- Bounds = -32 768 to 32 767
- Printed with “%d”

# Variable types

- long
- Stands for Long integer. Used for large variables.
- Size - double size of the integer (for x32 systems - 8 bytes)
- Bounds :  $-9.223 * 10^{18}$  to  $9.223 * 10^{18}$
- Printed with “%ld”

# Variable types

- Task:
- Define the best variable types for the following numbers:
  - 1 877 728 478
  - 19873
  - 4 981 777 162
  - 7
  - 82 794

# Answers

- 1 877 728 478 - int (or long)
- 19873 - int (or short)
- 4 981 777 162 - long
- 7 - int (or short)
- 82 794 - int

# Unsigned and signed variables

- There are cases when we are going to have variables with only positive value. In this way we should use unsigned variable types.
- That way we are doubling the number that we can contain.
- unsigned short bounds : 0 - 64535
- unsigned int bound : 0 - 4 294 967 296
- unsigned long bounds : 0 -  $1.844 * 10^{19}$

# Variable types

- We have been speaking for variable that contain only decimal numbers.
- What if we are having a number with floating point?

# Floating point variables

- In order to collect and calculate floating point variables we must use floating point variable types.

# Variable types

- float
- Stands for float number.
- Used for numbers with floating point. It is not too precise in the calculation and representations. It has size of 4 bytes.
- Bounds - not exact bounds. Recommended for numbers that has up to 4 digits after the floating point.
- Printed with “%f”



# Variable types

- double
- Stands for double float number
- Used for more precise calculations than float. It has size of 8 bytes.
- Bouds - recommended for numbers with up to 8 digits after the floating point.
- printed with “%lf”

# long double

- long double
- Stands for long double float
- Used for really precise calculations.
- Size - 10 or 16 bytes (depending on the platform)
- Printed with “%llf”

# Variable types

- char
- used for defining characters
- size = 1 byte
- bounds - -128 to 127
- printed with “%c”

```
char myCharacter = 'c';
```

Questions ?

# Task

- Give examples for all the variables that have been presented.

# Variable actions

- The core in the programming is the actions that can be done between the variables.

# Sum

```
int aNumber = 6;  
int anotherNumber = 8;  
int result = aNumber + anotherNumber;
```

# Subtraction

```
int main() {  
    int aNumber = 6;  
    int anotherNumber = 8;  
    int result = aNumber - anotherNumber;  
  
    return 0;  
}
```



# Multiplication

```
int main() {  
    int aNumber = 6;  
    int anotherNumber = 8;  
    int result = aNumber * anotherNumber;  
  
    return 0;  
}
```

# Division

```
int main() {  
    int aNumber = 6;  
    int anotherNumber = 8;  
    int result = aNumber / anotherNumber;  
  
    return 0;  
}
```

# Advanced functions

```
int main() {  
    int aNumber = 6;  
    int anotherNumber = 8;  
    aNumber += anotherNumber;  
    aNumber -= anotherNumber;  
    aNumber *= anotherNumber;  
    aNumber /= anotherNumber;  
    return 0;  
}
```

# Incrementation/Decrementation

```
int main() {  
    int aNumber = 6;  
    aNumber++; // aNumber += 1  
    aNumber--; // aNumber -= 1  
}
```

Questions ?

Scanning user input

# Scanning user input

- In order to have the user to input data in the application you must scan for user input. The easiest and most used way in C development is the scanf(...) function.

# Scanning user input

```
int main() {  
    int aNumber = 0; // variable we will save data into  
    printf("Please input a number:");  
    scanf("%d", &aNumber);  
    printf("%d", aNumber);  
    return 0;  
}
```

function

Where we are saving the value

Argument format (decimal number)



# Scanning user input

```
int main() {  
  
    //scanf with multiple arguments, separated by ','  
    int a,b,c,d,e,f;  
  
    printf("Please input a numbers:\n");  
    scanf("%d,%d,%d,%d,%d,%d", &a, &b, &c, &d, &e,&f);  
  
    printf("%d\n", d);  
    return 0;  
}
```

# Task

- Scan 10 numbers from keyboard inputted by user. Then calculate the average number of them.