

Input/Output

2024 Winter APS 105: Computer Fundamentals

Jon Eyolfson

Lecture 3

1.0.1

We Want Our Programs to Do Something!



Re-visiting Our First Program

```
#include <stdio.h>

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

Re-visiting Our First Program

```
#include <stdio.h>
int main(void) {
    printf("Hello world\n");
    return 0;
}
```

`stdio.h` is short for "standard input/output" and contains the **declaration** of `printf`

Re-visiting Our First Program

```
#include <stdio.h>
int main(void) {
    printf("Hello world\n");
    return 0;
}
```

`stdio.h` is short for "standard input/output" and contains the **declaration** of `printf`

the `printf` function outputs a **string** (sequence of characters) to the terminal
Note: `\n` should always end a line

Re-visiting Our First Program

```
#include <stdio.h>
int main(void) {
    printf("Hello world\n");
    return 0;
}
```

`stdio.h` is short for "standard input/output" and contains the **declaration** of `printf`

the `printf` function outputs a **string** (sequence of characters) to the terminal
Note: `\n` should always end a line

A string begins with a `"` followed by any number of characters until another `"` signifies the end (you don't see the double quotes)

Functions Can Take Input and Produce Output

`main` is a function that takes no inputs (that's what `void` means) and produces an `int` value as output

Functions Can Take Input and Produce Output

`main` is a function that takes no inputs (that's what `void` means) and produces an `int` value as output

For functions, we call the inputs to the function `arguments` and they're separated by commas if there's more than one

Functions Can Take Input and Produce Output

`main` is a function that takes no inputs (that's what `void` means) and produces an `int` value as output

For functions, we call the inputs to the function `arguments` and they're separated by commas if there's more than one

`printf` takes a string argument (we'll get to its type later) and returns an `int` value of how many characters were shown in the terminal

`printf` also does the work to show characters in the terminal

Functions Can Take Input and Produce Output

`main` is a function that takes no inputs (that's what `void` means) and produces an `int` value as output

For functions, we call the inputs to the function `arguments` and they're separated by commas if there's more than one

`printf` takes a string argument (we'll get to its type later) and returns an `int` value of how many characters were shown in the terminal

`printf` also does the work to show characters in the terminal

Running a function is called a `function call`

printf Accepts Multiple Arguments

```
#include <stdio.h>

int main(void) {
    printf("Hello world\n", 1);
    return 0;
}
```

printf Accepts Multiple Arguments

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

```
int main(void) {  
    printf("Hello world\n", 1);  
    return 0;  
}
```

← printf can now use the value 1
(the type of the value 1 is an `int`)

We Can Use Format Strings to Print Values

"Integer: %d\n" will replace %d with the characters representing the value of the first argument of printf after the format string

% is the escape character for a format specifier

Some format specifiers we'll use:

%d an int

%lf is for a double

%c is for an char

If you want to print a literal %, you need to use %%

printf Accepts Multiple Arguments

```
#include <stdio.h>

int main(void) {
    printf("Integer: %d\n", 1);
    return 0;
}
```

printf Accepts Multiple Arguments

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

```
int main(void) {  
    printf("Integer: %d\n", 1);  
    return 0;  
}
```

Values are used in order

A yellow bracket diagram is positioned below the printf statement. It consists of a horizontal line with a vertical line extending upwards from its center, ending in an arrowhead that points to the space between the format string and the argument '1'. This diagram visually demonstrates that the arguments are used in the order they are listed.

printf Accepts Multiple Arguments

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

```
int main(void) {  
    printf("Integer: %d\n", 1);  
    return 0;  
}
```

Values are used in order

After running you'll see
Integer: 1
in your terminal

We Can Output Variables to the Terminal


```
#include <stdio.h>

int main(void) {
    int x = 1;
    int y = 2;
    printf("Point: (%d, %d)\n", x, y);
    return 0;
}
```

We Can Output Variables to the Terminal

```
#include <stdio.h>

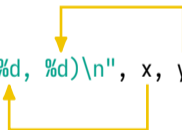
int main(void) {
    int x = 1;
    int y = 2;
    printf("Point: (%d, %d)\n", x, y);
    return 0;
}
```



We Can Output Variables to the Terminal

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

```
int main(void) {  
    int x = 1;  
    int y = 2;  
    printf("Point: (%d, %d)\n", x, y);  
    return 0;  
}
```



We Can Output Variables to the Terminal

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

```
int main(void) {  
    int x = 1;  
    int y = 2;  
    printf("Point: (%d, %d)\n", x, y);  
    return 0;  
}
```



After running you'll see
Point: (1, 2)
in your terminal

scanf Allows You to Get Input from the Terminal

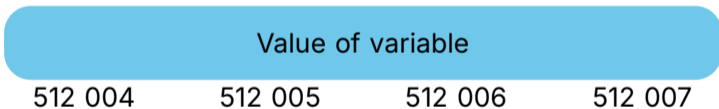
scanf is the opposite of printf and also uses a format string

However, instead of using a value, it assigns a value

scanf Allows You to Get Input from the Terminal

scanf is the opposite of printf and also uses a format string
However, instead of using a value, it assigns a value

You need to use the starting memory address of the variable



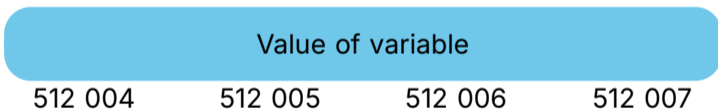
In this case the variable would start at 512 004

scanf Allows You to Get Input from the Terminal

scanf is the opposite of printf and also uses a format string

However, instead of using a value, it assigns a value

You need to use the starting memory address of the variable



In this case the variable would start at 512 004

Knowing the variable is an `int` means it requires 4 bytes

You Can Use & to Get the Starting Address of a Variable

Assume we have: `int x;`

`&` is a **unary operator**, e.g. `&x`

A unary operator means it only requires one operand

Note: addition, `+`, is a **binary operator**, e.g. `1 + 2`

You Can Use & to Get the Starting Address of a Variable

Assume we have: `int x;`

& is a **unary operator**, e.g. `&x`

A unary operator means it only requires one operand

Note: addition, `+`, is a **binary operator**, e.g. `1 + 2`

The result of `&` is a new value, with a new type

This type is the original type with a `*` at the end

The type of `&x` in this case is `int*` (more details later)

We Can Now Use Our Terminal for Input and Output

```
#include <stdio.h>

int main(void) {
    printf(" Input x: ");
    int x;
    scanf("%d", &x);
    printf("Output x: %d\n", x);
    return 0;
}
```

We Can Now Use Our Terminal for Input and Output

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

```
int main(void) {  
    printf(" Input x: ");  
    int x;  
    scanf("%d", &x);  
    printf("Output x: %d\n", x);  
    return 0;  
}
```

converts the characters we type
to an `int` value and assigns the
value to `x` using its address

scanf Returns the Number of Values Assigned

Note that the type of the arguments and the format specifiers should match
Otherwise you'll get very unpredictable results

Sometimes it's okay to use a different type for printf
`printf("The integer value of 'A' is: %d\n", 'A');`

We Can Create Variables That Can't Change

We can declare a variable by adding a `const` keyword before the type
A keyword is a name reserved by C, which you can't use

We could write: `const int x = 1;`
Now, we aren't allowed to re-assign `x`

It's good practice to add `const` to any variables that should never change

You Should Be Consistent with Variable Naming

Normal variable names should start with a lower case letter
Instead of a space, you should capitalize the next word

`const` variables should start with a capital letter

There's some special values where every letter is capitalized
and words are separated by underscores

A Consistent Coding Style is Important

The C compiler ignores whitespace, so it's just for us to read
There's a formatter for you, but you should know the rules

A Consistent Coding Style is Important

The C compiler ignores whitespace, so it's just for us to read
There's a formatter for you, but you should know the rules

Function definitions (like `main`) start at the beginning of a line

A Consistent Coding Style is Important

The C compiler ignores whitespace, so it's just for us to read
There's a formatter for you, but you should know the rules

Function definitions (like `main`) start at the beginning of a line

Every time you start a new set of curly brackets {
you need to indent the next lines

Every level of indentation is 4 spaces (for now we'll only have 1)

Let's Write A Program to Convert Inches to CM

```
#include <stdio.h>

int main(void) {
    const double InchesPerCM = 2.54;
    double inches;
    printf("Enter length (inches): ");
    scanf("%lf", &inches);
    double cm = inches * InchesPerCM;
    printf("Converted length (cm): %lf\n", cm);
    return 0;
}
```

Note: you can declare multiple variables of the same type by separating each name with a comma, e.g. `double inches, cm;`

A double Contain Approximately 16 Decimal Places

Format specifiers have sub-specifiers to change how to print the value

For numbers, after the starting % character, we can put a . followed by a whole number to indicate the number of decimal places to print

Full list of sub-specifiers here: CPlusPlus.com

Let's Only Output 2 Decimal Places

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

int main(void) {
    const double InchesPerCM = 2.54;
    double inches;
    printf("Enter length (inches): ");
    scanf("%lf", &inches);
    double cm = inches * InchesPerCM;
    printf("Converted length (cm): %.2lf\n", cm);
    return EXIT_SUCCESS;
}
```

Note: it's good practice not to use "magic values" like returning 0 from `main`. `stdlib.h` defines a `EXIT_SUCCESS` value to use instead if there's no errors