

Computer Systems, B1-2 2022

Introduction to C

Troels Henriksen

Based on slides by Michael Kirkedal Thomsen

September 5, 2022

Outline

- Syntax - Statements/Expressions/Values
- `main`
- Arrays
- Reading and writing to/from `stdio`

- Syntax - Statements/Expressions/Values
- `main`
- Arrays
- Reading and writing to/from `stdio`

Programming is a craft

I will do live coding—feel free to ask questions and make comments.

Recommended compiler flags

Compiler call

```
gcc -Wall -Wextra -pedantic -std=c11 -c [file]
```

- Wall Turns on (almost) all warnings
- Wextra Add warnings for unassigned values and more
- pedantic Makes warnings if you goes outside ISO C
- std=c11 Chooses the latest C standard

Makefile

Tool that makes it easy to build your program.

```
CC=gcc
CFLAGS=-std=c11 -Wall -Werror -Wextra -pedantic

.PHONY: clean all

cmd: updatedfile(s).c
    $(CC) $(CFLAGS) -o $@ -c $<
```

This will be detailed more over time.

C Style Guide – Code is made to be read

Always use curly braces. The opening brace should be on the same line as the declaration.

```
int f() {  
    for (int i = 0; i < 100; i++) {  
        if (i % 2 == 0) {  
            // Do something.  
        }  
    }  
    // Return something.  
}
```

- Use 2 spaces for indentation. Indent so as to make the structure of your code clear.
- All functions must return a value. Returning void is not allowed.
- Code must be warning-free.

The main function

- Simple main

```
int main() {  
    ...  
}
```

- Main with arguments

```
int main(int argc, char* argv[]) {  
    ...  
}
```

Back to reality - Return from main function

```
int main(int argc, char* argv[]) {  
    ...  
    return ???;  
}
```


Back to reality - Return from main function

```
int main(int argc, char* argv[]) {  
    ...  
    return ???;  
}
```

- Use the `exit` function with returning from `main` function
- Lookup the codes that you need (use `echo $?` to show result.)
- If nothing: `return EXIT_SUCCESS;` is assumed

```
#include <stdlib.h>  
  
int main(int argc, char* argv[]) {  
    ...  
    return EXIT_SUCCESS;  
}
```

Statements: conditional

```
#include <stdlib.h>

int main(int argc, char* argv[]) {
    if (argc == 2) { // 1 argument
        return EXIT_SUCCESS;
    } else if (argc == 4) { // 3 arguments
        return EXIT_SUCCESS;
    } else {
        return EXIT_FAILURE;
    }
}
```

- This is actually a statement with a nested statement

Statements: switch-case

```
#include <stdlib.h>

int main(int argc, char* argv[]) {
    switch (argc) {
        case 2:
            return EXIT_SUCCESS;
            break;
        case 4:
            return EXIT_SUCCESS;
            break;
        default:
            return EXIT_FAILURE;
            break;
    }
}
```

Statements: switch-case alternative

```
#include <stdlib.h>

int main(int argc, char* argv[]) {
    switch (argc) {
        case 2:
        case 4:
            return EXIT_SUCCESS;
            break;
        default:
            return EXIT_FAILURE;
            break;
    }
}
```

- Cases only works on constant values (not expressions like conditionals).

Statements: while-loop

```
#include <stdio.h>

int main(int argc, char* argv[]) {
    int i = 1;
    while (i < argc) {
        printf("Argument_number_%d:_%s\n",
               i, argv[i]);
        i = i + 1;
    }
}
```

- Use any expression
- Easy to make it diverge

Statements: for-loop

```
#include <stdio.h>

int main(int argc, char* argv[]) {
    for (int i = 1; i < argc; i++) {
        printf("Argument_number_%d:_%s\n",
            i, argv[i]);
    }
}
```

- Good for iteration
- Keep it simple

Assignments

```
int main(int argc, char* argv[]) {  
    int a = 1;  
    int b = 2 + a;  
    a += b; // a = a + b  
    a *= 2;  
    a++; // a = a + 1  
    a--;  
    ++a; // a = a + 1  
}
```

- Useful short-hand writing styles.
- Easy to understand when writing simple programs

Watch your side

What is the values of a, b, and c?

```
#include <stdio.h>

int main(int argc, char* argv[]) {
    int a = 3;
    int b = 5;
    int c = a++ + ++b;
    printf("a=_%d\n", a);
    printf("b=_%d\n", b);
    printf("c=_%d\n", c);
}
```


Watch your side

What is the values of a, b, and c?

```
#include <stdio.h>

int main(int argc, char* argv[]) {
    int a = 3;
    int b = 5;
    int c = a++ + ++b;
    printf("a=_%d\n", a);
    printf("b=_%d\n", b);
    printf("c=_%d\n", c);
}
```

- Informally, ++a first increments a, then returns its value
- Informally, b++ first returns the value of b, then increments
- Be very careful of side-effects

Watch you side – lets try again

What is the values of a, b, and c?

```
#include <stdio.h>

int main(int argc, char* argv[]) {
    int a = 3;
    int b = 5;
    int c = -2;
    a += (c = a++ - (b += --c)) + ++b;
    printf("a=_%d\n", a);
    printf("b=_%d\n", b);
    printf("c=_%d\n", c);
}
```

Watch you side – lets try again

What is the values of a, b, and c?

```
#include <stdio.h>

int main(int argc, char* argv[]) {
    int a = 3;
    int b = 5;
    int c = -2;
    a += (c = a++ - (b += --c)) + ++b;
    printf("a_=_%d\n", a);
    printf("b_=_%d\n", b);
    printf("c_=_%d\n", c);
}
```

- Assignments are expressions with side-effects that return the value that is assigned to a variable.
- **Order of evaluation in expression is unspecified.**

Arrays

```
#include <stdio.h>

int main(int argc, char* argv[]) {
    int a[16];
    a[0] = 1;
    for (int i = 1; i < 16; i++) {
        a[i] = a[i-1] + 1;
    }
    printf("final_=%d\n", a[15]);
}
```

- No check for out-of-bounds
- Arrays cannot be assigned to
- Arrays cannot be compared
- Do it element wise

Writing to stdout

```
int result = fprintf("Show_%s_of_value_%d", ... );
```

- c Character
- d or i Signed decimal integer
- e Scientific notation (mantissa/exponent) using e character
- E Scientific notation (mantissa/exponent) using E character
- f Decimal floating point
- g Uses the shorter of %e or %f
- G Uses the shorter of %E or %f
- s String of characters
- u Unsigned decimal integer
- x Unsigned hexadecimal integer
- X Unsigned hexadecimal integer (capital letters)
- p Pointer address
- n Nothing printed
- % Character

Reading from stdin

```
#include <stdio.h>

int main(int argc, char* argv[]) {
    char input[20];
    printf ("Give_me_a_string\n");
    scanf("%s", input);
    printf ("You_wrote:_%s\n", input);
}
```

- Reading value from stdin

Reading from stdin

```
#include <stdio.h>

int main(int argc, char* argv[]) {
    char input[20];
    printf ("Give_me_a_string\n");
    scanf("%s", input);
    printf ("You_wrote:_%s\n", input);
}
```

- Reading value from stdin
- Problem with writing outside buffer

Reading a string

```
#include <stdio.h>

int main(int argc, char* argv[]) {
    char input[20];
    printf ("Give_me_a_string\n");
    fgets(input, sizeof(input), stdin);
    printf ("You_wrote:_%s\n", input);
}
```

- Use fgets
- Can limit the number of values read
- Made for reading from files
- stdin is just a file (somewhere)

Reading values

```
#include <stdio.h>

int main(int argc, char* argv[]) {
    int input;
    printf ("Give_me_a_integer\n");
    scanf("%d", input);
    printf ("You_wrote:_%d\n", input);
}
```

- Reading value from stdin

Reading values

```
#include <stdio.h>

int main(int argc, char* argv[]) {
    int input;
    printf ("Give_me_a_integer\n");
    scanf("%d", input);
    printf ("You_wrote:_%d\n", input);
}
```

- Reading value from stdin
- Does not work

Reading values – correct

```
#include <stdio.h>

int main(int argc, char* argv[]) {
    int input;
    printf ("Give_me_a_integer\n");
    scanf ("%d", &input);
    printf ("You_wrote:_%d\n", input);
}
```

- Reading value from stdin

Reading values – correct

```
#include <stdio.h>

int main(int argc, char* argv[]) {
    int input;
    printf ("Give_me_a_integer\n");
    scanf ("%d", &input);
    printf ("You_wrote:_%d\n", input);
}
```

- Reading value from stdin
- Does not work

Getting values from arguments

```
#include <stdio.h>

int main(int argc, char *argv[]) {
    int input;
    if(argc == 2) {
        int res = sscanf(argv[1], "%d", &input);
        if (res == 0) {
            printf("Bad_value\n");
        } else {
            printf("Input_was:_%d\n", input);
        }
    }
}
```

You now know enough to get in trouble

Questions?