basicstyle=

Computer Systems, B1-2 2024

Introduction to C

David Marchant
Based on slides by Troels Henriksen and Michael Kirkedal Thomsen

2024-09-02

Hello

- David on the Discord, david.marchant@di.ku.dk, 772-01-0-s06
- Mit dankse er ikke godt
- I will make an appearance at exercise classes and assignment cafes
- I run linux, not mac, not windows
- Questions and comments are always welcome

Outline

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

Outline

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

Programming is a craft

I will sometimes do live coding. Feel free to ask questions and make comments anytime.

Recommended compiler flags

Compiler call

```
gcc -Wall -Wextra -pedantic -std=c18 -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=c18 Chooses the latest C standard

Tool that makes it easy to build your program.

```
CC=gcc
CFLAGS=-std=c18 -Wall -Werror -Wextra -pedantic

.PHONY: clean all

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

This will be detailed more over time.

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

```c
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.

- Simple main

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

- Main with arguments

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

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

```c
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

```c
#include <stdlib.h>

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

```c
#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

```c
#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;
  }
}
```

```c
#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

```c
#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

```c
#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

```c
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

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

```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?

```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

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

```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);
}
```

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

```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.**

```c
#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 = printf("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

```c
#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

```c
#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

```c
#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)

```c
#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

```c
#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

```c
#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

```c
#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

```c
#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);
    }
  }
}
```

Questions?