Complete Training

C Programming

From Fundamentals to Practical Exercises

Preparation for C-Piscine-C-00

Table of Contents

1. Introduction to C Language

What is C?

C is a programming language created in the 1970s. It's a **compiled** language, which means that your source code is translated into machine language before being executed.

Why Learn C?

• C teaches you how a computer really works
• It forces you to understand memory and pointers
• It's the foundation of many other languages (C++, Java, etc.)
• It's used for operating systems, drivers, embedded systems

> **Important note:** C doesn't forgive mistakes. That's precisely why it's excellent for learning!

2. Structure of a C Program

The Minimal Program

Every C program starts with a main function:

```
int main()
{
    return (0);
}
```

Let's break down this code:

• **int**: indicates that the function returns an integer
• **main**: it's the name of the main function
• **()**: the parentheses contain parameters (none here)
• **{...}**: the braces delimit the function body
• **return (0);**: indicates that the program terminated successfully
• **;**: each statement ends with a semicolon

Compilation and Execution

To transform your code into an executable program:

```
# Compile
cc my_file.c -o my_program

# Execute
./my_program
```

> **Tip:** If your program doesn't compile, read the error messages carefully. They tell you exactly where the problem is.

3. The write() Function

Understanding write()

The **write()** function is the basic system function for writing data. It's the only function you're allowed to use to display text in the exercises.

To use it, you must include unistd.h:

```
#include <unistd.h>
```

Syntax of write()

```
write(file_descriptor, address, number_of_bytes);
```

• **file_descriptor**: where to write (1 = standard output, the screen)
• **address**: the memory address of what we want to write
• **number_of_bytes**: how many bytes to write

Practical Example

```
#include <unistd.h>

int main()
{
    char c;
    c = 'A';
    write(1, &c, 1);
    return (0);
}
```

This program displays the letter 'A'. Let's break it down:

• **&c**: the & symbol gives the address of variable c
• **1**: we write 1 byte (one character is 1 byte)

4. Characters and ASCII

What is ASCII?

ASCII (American Standard Code for Information Interchange) is a system that associates each character with a number between 0 and 127.

Essential ASCII Table

Code	Character	Description
32	space	Space
48-57	0-9	Digits
65-90	A-Z	Uppercase letters
97-122	a-z	Lowercase letters

Key Points

- The character 'A' has ASCII code 65
- The character 'a' has ASCII code 97
- The difference between uppercase and lowercase is 32
- The character '0' (zero) has ASCII code 48

> **Important:** '0' (the character zero) is different from 0 (the number zero)!

Usage Example

```
#include <unistd.h>

int main()
{
    char c;
    c = 'A';
    write(1, &c, 1); // Displays 'A'
    c = c + 1;
    write(1, &c, 1); // Displays 'B'
    return (0);
}
```

5. Variables

What is a Variable?

A variable is a space in the computer's memory where you can store a value. It's like a box with a label.

Variable Types

Type	Description	Example
char	One character (1 byte)	'A', 'z', '5'
int	An integer number	42, -17, 0
void	No value	For functions

Declaration and Initialization

```
int main()
{
    char letter; // Declaration
    int number; // Declaration

    letter = 'X'; // Initialization
    number = 42; // Initialization

    // Or in one line:
    char another_letter = 'Y';
    return (0);
}
```

Important rule: In C, all variables must be declared at the beginning of a block (after an opening brace).

6. Loops

Why Loops?

Loops allow you to repeat actions without rewriting the same code multiple times. It's one of the most powerful concepts in programming.

The while Loop

The while loop repeats instructions as long as a condition is true:

```
while (condition)
{
    // Instructions to repeat
}
```

Example 1: Display a Character 5 Times

```
#include <unistd.h>

int main()
{
    int counter;
    char c;

    counter = 0;
    c = '*';

    while (counter < 5)
    {
        write(1, &c, 1);
        counter = counter + 1;
    }
    return (0);
}
```

This program displays: *****

Example 2: Display the Alphabet

```
#include <unistd.h>

int main()
{
    char letter;

    letter = 'a';
    while (letter <= 'z')
```

```
    {
        write(1, &letter, 1);
        letter = letter + 1;
    }
    return (0);
}
```

This program displays: abcdefghijklmnopqrstuvwxyz

> **Understanding:** At each iteration, we display the current letter, then move to the next one by adding 1 to the ASCII code.

Comparison Operators

Operators you can use in conditions:

Operator	Meaning	Example
==	equal to	a == b
!=	not equal to	a != b
<	less than	a < b
>	greater than	a > b
<=	less than or equal	a <= b
>=	greater than or equal	a >= b

Warning: Don't forget to increment your counter, or you'll create an infinite loop!

7. Conditions

The if / else Structure

Conditions allow your program to make decisions:

```
if (condition)
{
    // If the condition is true
}
else
{
    // Otherwise
}
```

Example 1: Test a Number

```
#include <unistd.h>

int main()
{
    int number;
    char c;

    number = 5;

    if (number > 0)
    {
        c = 'P';
        write(1, &c, 1);
    }
    else
    {
        c = 'N';
        write(1, &c, 1);
    }
    return (0);
}
```

This program displays: P (because 5 is positive)

Example 2: Handle Zero

```
if (number > 0)
{
    c = 'P';
    write(1, &c, 1);
}
else if (number == 0)
{
```

```
    c = 'Z';
    write(1, &c, 1);
}
else
{
    c = 'N';
    write(1, &c, 1);
}
```

Important: Note the use of == to test equality, not = which is used for assignment!

8. Functions

What is a Function?

A function is a reusable block of code that performs a specific task. It's like a mini-machine in your program.

Anatomy of a Function

```
return_type function_name(param_type param)
{
    // Function body
    return (value);
}
```

• **return_type**: the type of value the function returns (void if none)
• **function_name**: the name you give to your function
• **param_type param**: the parameters the function receives
• **return**: returns a value (optional if void)

Example: ft_putchar

```
#include <unistd.h>

void ft_putchar(char c)
{
    write(1, &c, 1);
}
```

Let's break it down:

• **void**: the function returns nothing
• **ft_putchar**: the function name
• **char c**: it takes a character as a parameter
• It displays this character with write

Using a Function

```
#include <unistd.h>

void ft_putchar(char c)
{
    write(1, &c, 1);
}

int main()
{
```

```
    ft_putchar('H');
    ft_putchar('i');
    ft_putchar('!');
    return (0);
}
```

This program displays: Hi!

Function Prototype

A prototype declares a function before defining it. This allows the compiler to verify that you're using it correctly:

```
void ft_putchar(char c);
```

In the exercises, you'll always be asked to respect the given prototype. This is crucial for your code to be compatible with tests.

> **Tip:** Always create your functions so that they do one thing, and do it well!

9. Guided Practical Exercises

Guided Exercise 1: Display 10 Stars

Objective: Create a program that displays 10 stars

Step 1: Create the basic structure

```
#include <unistd.h>

int main()
{
    // Your code here
    return (0);
}
```

Step 2: Declare the necessary variables

```
int main()
{
    int counter; // To count the stars
    char star; // The character to display
    return (0);
}
```

Step 3: Initialize the variables

```
int main()
{
    int counter;
    char star;

    counter = 0;
    star = '*';
    return (0);
}
```

Step 4: Create the loop

```
int main()
{
    int counter;
    char star;

    counter = 0;
    star = '*';

    while (counter < 10)
    {
        write(1, &star, 1);
        counter = counter + 1;
```

```
    }
    return (0);
}
```

Result: **********

Guided Exercise 2: Display Numbers from 0 to 9

Reflection:

• What variables do I need? A character to display
• What is my starting value? The character '0' (ASCII code 48)
• What is my stop condition? When I reach the character '9'
• How to move to the next number? Add 1 to the character

Solution:

```
#include <unistd.h>

int main()
{
    char digit;

    digit = '0';
    while (digit <= '9')
    {
        write(1, &digit, 1);
        digit = digit + 1;
    }
    return (0);
}
```

Result: 0123456789

> **Understanding:** We use '0' and '9' (with quotes) because they are ASCII characters, not integer numbers.

Guided Exercise 3: Function to Test a Number

Objective: Create a function that displays 'P' if a number is positive or zero, 'N' if it's negative

Step 1: Write the prototype

```
void test_number(int n);
```

Step 2: Write the function structure

```
void test_number(int n)
{
    // Your code here
}
```

Step 3: Add the logic

```
#include <unistd.h>

void test_number(int n)
{
    char c;

    if (n >= 0)
    {
        c = 'P';
    }
    else
    {
        c = 'N';
    }
    write(1, &c, 1);
}
```

Test the function:

```
int main()
{
    test_number(5); // Displays 'P'
    test_number(-3); // Displays 'N'
    test_number(0); // Displays 'P'
    return (0);
}
```

10. Solution to Subject Exercises

Exercise 0: ft_putchar

Analysis:

• Given prototype: void ft_putchar(char c)
• The function must display a character
• We must use write(1, &c, 1)

Solution:

```
#include <unistd.h>

void ft_putchar(char c)
{
    write(1, &c, 1);
}
```

Line-by-line explanation:

1. **#include <unistd.h>**: necessary to use write
2. **void**: the function returns nothing
3. **ft_putchar**: the name imposed by the subject
4. **(char c)**: it receives a character as a parameter
5. **write(1, &c, 1)**: writes 1 character to standard output

> **Key points:** This is the simplest but most important function! It will be used in all other exercises.

Exercise 1: ft_print_alphabet

Analysis:

• Given prototype: void ft_print_alphabet(void)
• Display all lowercase letters from 'a' to 'z'
• On a single line
• In ascending order

Strategy:

1. Create a char variable initialized to 'a'
2. While this variable is less than or equal to 'z'
3. Display it with write
4. Move to the next letter by adding 1

Solution:

```
#include <unistd.h>

void ft_print_alphabet(void)
{
    char letter;

    letter = 'a';
    while (letter <= 'z')
    {
        write(1, &letter, 1);
        letter = letter + 1;
    }
}
```

Expected result: abcdefghijklmnopqrstuvwxyz

> **Why it works:** In ASCII, lowercase letters are consecutive from 'a' (97) to 'z' (122). By adding 1, we naturally move from one letter to the next.

Exercise 2: ft_print_reverse_alphabet

Analysis:

• Given prototype: void ft_print_reverse_alphabet(void)
• Display all lowercase letters from 'z' to 'a'
• On a single line
• In descending order

Strategy:

1. Create a char variable initialized to 'z'
2. While this variable is greater than or equal to 'a'
3. Display it with write
4. Move to the previous letter by subtracting 1

Solution:

```
#include <unistd.h>

void ft_print_reverse_alphabet(void)
{
    char letter;

    letter = 'z';
    while (letter >= 'a')
    {
        write(1, &letter, 1);
        letter = letter - 1;
    }
}
```

Expected result: zyxwvutsrqponmlkjihgfedcba

> **Difference from exercise 1:** We start at 'z', use >= instead of <=, and subtract 1 instead of adding 1.

Exercise 3: ft_print_numbers

Analysis:

• Given prototype: void ft_print_numbers(void)
• Display all digits from 0 to 9
• On a single line
• In ascending order

Trap to avoid:

Don't confuse the **number** 0 and the **character** '0'!
• The number 0 has the value 0
• The character '0' has ASCII code 48

Solution:

```
#include <unistd.h>

void ft_print_numbers(void)
{
    char digit;

    digit = '0';
    while (digit <= '9')
    {
        write(1, &digit, 1);
        digit = digit + 1;
    }
}
```

Expected result: 0123456789

> **Why '0' and not 0:** We must display characters, not numbers. '0' is the character zero (ASCII 48), while 0 is the number zero which cannot be displayed directly.

Exercise 4: ft_is_negative

Analysis:

• Given prototype: void ft_is_negative(int n)
• If n is negative, display 'N'
• If n is positive or zero, display 'P'

Reflection:

Question: What does 'negative' mean?
Answer: A number less than 0

Question: What about zero?
Answer: The subject says 'positive or zero', so 0 must display 'P'

Strategy:

1. Declare a char variable for the character to display
2. Test if n is negative (n < 0)
3. If yes, put 'N' in the variable
4. Otherwise, put 'P' in the variable
5. Display the variable with write

Solution:

```
#include <unistd.h>

void ft_is_negative(int n)
{
    char c;

    if (n < 0)
    {
        c = 'N';
    }
    else
    {
        c = 'P';
    }
    write(1, &c, 1);
}
```

Test the function:

```
ft_is_negative(5); // Displays 'P'
ft_is_negative(-3); // Displays 'N'
ft_is_negative(0); // Displays 'P'
```

Important point: The condition uses < and not <=, because zero must be considered positive according to the subject.

Tips and Methodology

Methodical Approach to Solve an Exercise

1. Read and Understand

• Read the statement several times
• Identify exactly what is being asked
• Note the imposed prototype
• Identify the allowed functions

2. Break Down the Problem

• What are the necessary steps?
• Do I need a loop? A condition?
• What variables are necessary?
• What is the simplest case? The most complex?

3. Write the Code

• Start with the basic structure
• Add the variables
• Write the main logic
• Test mentally with simple values

4. Test

• Compile your code
• Test with different inputs
• Check edge cases (0, negative, etc.)
• Make sure there are no segmentation faults

Common Mistakes to Avoid

1. Forgetting the semicolon
Each statement must end with ;

2. Confusing = and ==
= is for assignment, == is for comparison

3. Forgetting to increment in a loop
Result: infinite loop!

4. Confusing '0' and 0

'0' is a character (ASCII 48), 0 is a number

5. Not including unistd.h
Without it, write() won't work

6. Wrong loop condition
Check carefully <, <=, >, >=

Understanding Error Messages

Error: expected ';' before ...
→ You forgot a semicolon

Error: 'write' undeclared
→ You forgot #include <unistd.h>

Warning: unused variable
→ You declared a variable but don't use it

Segmentation fault
→ Memory access problem (often an infinite loop or incorrect pointer)

Best Practices

• **Indent your code**: 1 tab or 4 spaces per level
• **Name your variables clearly**: 'counter' rather than 'i'
• **Compile often**: don't code 100 lines before compiling
• **Test each function separately**: make sure it works alone
• **Respect the Norm**: norminette is your friend, not your enemy
• **Ask for help**: discuss with your peers, it's encouraged!

> **Important reminder:** Peer learning is at the heart of 42's pedagogy. Never hesitate to ask for help or to explain your code to someone else. Teaching is the best way to learn!

Useful Commands

Compile a program:

```
cc -Wall -Wextra -Werror file.c -o program
```

Execute a program:

```
./program
```

Check the norm:

```
norminette -R CheckForbiddenSourceHeader file.c
```

Display ASCII table:

```
man ascii
```

Additional Resources

• **man write**: documentation for the write function
• **man ascii**: complete ASCII table
• **man cc**: compiler documentation
• **Your peers**: your best resource!
• **Piscine Slack**: for general questions

Conclusion

You now have all the necessary knowledge to succeed in the C-Piscine-C-00 exercises. Here's a summary of what you've learned:

✓ The structure of a C program
✓ Using the write() function
✓ Characters and the ASCII system
✓ Variables (char, int)
✓ While loops
✓ If/else conditions
✓ Creating and using functions
✓ Methodical problem solving

Remember:

• Programming is learned by practice, not by reading
• Each error is an opportunity to learn
• Failure is part of the learning process
• Collaboration with your peers is encouraged
• Patience and perseverance are your best allies

> **Final message:** You are now ready to face the C Piscine. Don't get discouraged if things are difficult at first — it's normal and it's by design! Every line of code you write brings you closer to mastering C. Good luck!

By Odin, by Thor! Use your brain!!!