PRCMP PL10
Writting and running programs

November 29, 2023

1 Setting up the work environment

This exercise sheet requires a work environment that allows:

- run programs developed in the Python programming language,
- compile, run and debug programs developed in the C programming language.

1.1 The easy solution

The DEI Linux server has all the necessary tools installed for the problems proposed in this sheet. Just log in and perform the tasks in a remote session.

1.2 Install the tools on your computer

Alternatively, you can install the necessary tools on your computer. The method is slightly different depending on whether your computer has Linux or macOS, but both processes are equally easy. But first, check if you have the tools already installed.

In Ubuntu, run the following commands:

```
$ python3 --version
Python 3.8.10
$ gcc --version
gcc (Ubuntu 9.4.0-1ubuntu1~20.04.2) 9.4.0
Copyright (C) 2019 Free Software Foundation, Inc.
This is free software; see the source for copying conditions.
There is NO warranty; not even for MERCHANTABILITY or FITNESS FOR
A PARTICULAR PURPOSE.
$
```

In macOS, run the following commands:

```
$ python3 --version
Python 3.9.6
$ clang --version
Apple clang version 15.0.0 (clang-1500.0.40.1)
Target: x86_64-apple-darwin23.1.0
Thread model: posix
InstalledDir: /Library/Developer/CommandLineTools/usr/bin
$
```

1.2.1 Guide for Ubuntu

Install Python Installing Python on Ubuntu uses the apt package manager. First you must update the installed packages, and then install Python. Finally, check if `python3` is installed.

```
$ sudo apt update
   ...
$ sudo apt install python3
   ...
$ python3 --version
Python 3.8.10
$
```

Install gcc Installing the GNU C compiler (gcc) on Ubuntu uses the apt package manager. Make sure the installed packages are up to date and then install gcc. Finally, check that `gcc` is installed.

```
$ sudo apt update
   ...
$ sudo apt install build-essential
   ...
$ gcc --version
gcc (Ubuntu 9.4.0-1ubuntu1~20.04.2) 9.4.0
Copyright (C) 2019 Free Software Foundation, Inc.
This is free software; see the source for copying conditions.
There is NO warranty; not even for MERCHANTABILITY or FITNESS FOR
A PARTICULAR PURPOSE.
$
```

1.2.2 Guide for macOS

Install Python Downloading the latest Python version from the official Python website (python.org) is the most common (and recommended) method for installing Python on a Mac.

First, download the installer package from the Python website: https://www.python.org/downloads/ Run the installer walk through the wizard to complete the installation.

Finally, check if `python3` is installed:

```
$ python3 --version
Python 3.9.6
$
```

Install clang To install the C clang compiler, run the following command, not forgetting to check if `clang` was successfully installed, afterwards:

```
$ xcode-select --install
    ...
$ clang --version
Apple clang version 15.0.0 (clang-1500.0.40.1)
Target: x86_64-apple-darwin23.1.0
Thread model: posix
InstalledDir: /Library/Developer/CommandLineTools/usr/bin
$
```

2 Interpreted programs: Python

1. Write your first "Hello world" python script, and save it in the file `hello.py`. Use your favourite text editor and write the following code:

```
#!/usr/bin/python3

# This program prints Hello, world!
print('Hello, world!')

```

   Assign the appropriate permissions and run the script.

2. Let's try a more complex program. This time the program will ask the user to enter two real numbers and then displays the sum of the two.

```
#!/usr/bin/python3

# This program adds two numbers

# Store input numbers
num1 = float(input('Enter first number: '))
num2 = float(input('Enter second number: '))

# Add two numbers
sum = num1 + num2

# Display the sum
print('The sum of {0} and {1} is {2}'.format(num1, num2, sum))

```

   The `input()` function returns a string entered by the user; this string is converted to a real number using the `float()` function. The result is formatted using the `str.format()` method, in which the variable values are placed in the placeholders marked by curly braces.

   Save the program in the file `add2numbers.py` and run it.

3. Let's try a program that determines whether a number given by the user is prime. Transcribe the following source code into the `prime.py` file and run it several times, testing several input values.

   Search the web for documentation about the `if` and `for` statements and the `range()` function.

   Analyse the code and correlate it with the output of your experiments.

```
#!/usr/bin/python3

# Take input from the user
num = int(input("Enter a number: "))

if num == 1:
    print("By definition, 1 is not a prime number.")
elif num > 1:
   # check for factors
   for i in range(2, num):
       if (num % i) == 0:
           print(num, "is not a prime number:")
           print("  -> ", i, "times", num//i, "is", num, ".")
           break
   else:
       print(num, "is a prime number.")
else:
   print(num,"is not a prime number:")
   print("  -> Prime numbers are greater than 1, by definition.")

```

4. Finally, we will see that an interpreted program can solve even more complex numerical problems. The following program transposes a predefined rectangular matrix. Transcribe the source code into the `transpose.py` file and run it.

   Analyse the code and correlate it with the output of the program.

```
#!/usr/bin/python3

# Program to transpose a matrix using a nested loop

A = [[12,7],
     [4 ,5],
     [3 ,8]]

At = [[0,0,0],
      [0,0,0]]

# Print matrix
print('A =')
for r in A:
   print(r)

# Iterate through rows
for i in range(len(A)):
   # iterate through columns
   for j in range(len(A[0])):
       At[j][i] = A[i][j]

# Print transposed matrix
print('\nAt = ')
for r in At:
   print(r)

```

3 Compiled programs: C

5. Write your first "Hello world" in the C language, and save it in the file `hello.c`. Use your favourite text editor and write the following code:

```
#include <stdio.h>

int main()
{
    printf("Hello, world!\n");

    return 0;
}

```

   This program has to be compiled to obtain an executable file. This executable file must contain machine code that is directly decoded and executed by the processor.

   The command to build an executable depends on the compiler (`gcc` or `clang`) installed on the system:

   - `gcc hello.c -o hello`
   - `clang hello.c -o hello`

   In both cases, you must indicate the file with the source code to be compiled (`hello.c`); the `-o` option allows you to indicate the name of the final executable file (`hello`), otherwise `a.out` would be the default file name.

   Run the program calling it from the command line:

```
$ ./hello
Hello, world!
$
```

6. Write a program that displays the size in bytes of the following C basic types:
   - `char`
   - `short`
   - `int`
   - `long`
   - `long long`
   - `float`
   - `double`
   - `long double`

   Tip: In C, the size of a type is converted into an `unsigned long`.
7. Write a C program that asks the user for an integer. The program must display the entered value and the base address of the variable that contains it.
8. Write a program meeting all of the following requirements.

   R1: Develop the `average()` function that receives two integers and returns their (integer) average. The prototype of the function is:

   ```
   int average(int v1, int v2)
   ```

   R2: The `main()` function has three integer variables:

   ```
   int a, b, avg;
   ```

   R3: The user enters two values that are stored in variables `a` and `b`. The average of the two should be placed in the `avg` variable, using the `average()` function..

   Compile and test the program, checking its correct operation.
9. Extend the previous program to allow you to visualise the base addresses of:
   - `main()` variables `a`, `b` and `c`;
   - `average()` variables `v1` and `v2`
   - functions `main()` and `average()`.

   Assume that the value of variable `a` is passed as an argument to parameter `v1` when the `average()` function is called. Compare the base addresses of these two variables. What can you conclude from this observation?
10. Write a program that creates the array `v[]` with capacity for five values of type `double`. The program should perform the following actions.
    (a) Fill the array with 5 values (choose your preferred method).
    (b) Print the 5 values stored in the array.
    (c) Present the base address and the number of bytes occupied by the array in memory.
    (d) Finally, print the value stored in v[10].

    What is your opinion on the correctness of the last action?
11. Let's observe and analyse the argument passing mechanism when calling a function. Write a program that performs the following actions.

(a) Write the function `foo()` that takes an integer as an argument in the `value` parameter and does not return any value (void).

```
1 void foo(int value)
2 {
3     /* Insert your code here. */
4 }
5
```

First, the function prints the address and contents of the `value` variable.

Then, it assigns the value 5 to `value`.

Finally, it displays the address and contents of `value` again, before returning.

(b) Write the main function, which contains the local integer variable `x`.

Start by assigning the value 16 to `x`.

Print the address and contents of `x`.

Call the function `foo()`, passing the variable `x`.

Print the address and contents of `x`, again, before terminating the program.

The function changed the contents of `value`. Did this change have consequences for the `x` variable? Find a reasoned justification for what you observed.

12. Let's now try something similar, but with an array. Write a program that performs the following actions.

(a) Write the function `bar()` that takes an array of integers and the numbers of elements of that array as arguments and does not return any value (void).

```
1 void bar(int array[], int n_elements)
2 {
3     /* Insert your code here. */
4 }
5
```

First, the function prints the content given by `array`: i.e. `printf( ... , array)`.

Now, for each element of the array, assign the value 5 and then print its base address and its value.

Finally, `bar()` returns.

(b) Write the main function, which contains the local integer array `v[5]`.

First, for each element of the array, assign the value 16 and then print its base address and its value.

Call the function `bar()`, passing the array `v[]` and the number of elements (5).

For each element in the array print its address and contents, again, before terminating the program.

The function changed the contents of its local `array[]`. Did this change have consequences for the `v[]` array? Find a reasoned justification for what you observed.