CS100: Introduction to Programming
First-year undergraduate course for SIST students @ ShanghaiTech
Summer-homework booklet

Abstract

CS100: Introduction to Programming is a mandatory first-year course for all students enrolling at SIST, ShanghaiTech. Programming is an essential, foundational skill that is perhaps more instrumental than any other to your successful university studies. As students arrive with diverging backgrounds in programming, we issue the present summer homework booklet with basic lessons and (graded!) exercises for you to get started before the actual start of the semester. The intention is to standardize prior knowledge in programming across the entire class. The summer homework aims at a very basic level of C programming. **Instructions on how to submit your solution for online grading are provided on the webpage. Note that the grades will already influence your overall outcome for CS100.**

Important message on plagiarism

The single most important point for you to realize before the beginning of your studies at ShanghaiTech is the meaning of "plagiarism":

Plagiarism is the practice of taking someone else's work or ideas and passing them off as one's own. It is the misrepresentation of the work of another as your own. It is academic theft; a serious infraction of a University honor code, and the latter is your responsibility to uphold. Instances of plagiarism or any other cheating will be reported to the university leadership, and will have serious consequences. Avoiding any form of plagiarism is in your own interest. If you plagiarize and it is unveiled at a later stage only, it will not only reflect badly on the university, but also on your image/career opportunities.

Plagiarism is academic misconduct, and we take it very serious at ShanghaiTech. In the past we have had lots of problems related to plagiarism especially with newly arriving students, so it is important to get this right upfront:

You may …
- … discuss with your peers about course material.
- … discuss generally about the programming language, some features, or <u>abstract</u> lines of code. As long as it is <u>not directly related to any homework</u>, but <u>formulated in a general, abstract way</u>, such discussion is acceptable.
- … share test cases with each other.
- … help each other with setting up the development environment etc.

You may not …
- **… read, possess, copy or submit the solution code of anyone else** (including people outside this course or university)!
- **… receive direct help from someone else** (i.e. a direct communication of some lines of code, no matter if it is visual, verbal, or written)!
- **… give direct help to someone else.** Helping one of your peers by letting him read your code or communicating even just part of the solution in written or in verbal form will have equal consequences.
- … gain access to another one's account, no matter if with or without permission.
- … give your account access to another student. It is your responsibility to keep your account safe, always log out, and choose a safe password. Do not just share access to your computer with other students without prior lock-out and disabling of automatic login functionality. Do not just leave your computer on without a lock even if it is just for the sake of a 5-minute break.
- … work in teams. You may meet to discuss generally about the material, but any work on the homeworks is to be done individually and in privacy. Remember, you may not allow anyone to even just read your source code.

With the Internet, "copy", "paste", and "share" are easy operations. Don't think that it is easy to hide and that we will not find you, we have just as easy to use, fully automatic and intelligent tools that will identify any potential cases of plagiarism. And do not think that being the original author will make any difference. Sharing an original solution with others is just as unethical as using someone else's work.

Lesson 1
"Hello World"

The first lesson is just to get you to write, compile, and execute a very simple program. It is a very basic example commonly used to introduce C.

- Open an editor (Notepad under Windows, Xemacs under Linux, Sublime, etc.) and enter the following text:

```
prog.c                    ×
1   /* A program to print "Hello World!" to the console */
2
3   #include <stdio.h>          /* preprocessor instruction*/
4
5   int main() {                /* main program header */
6     printf("Hello World!");   /* print message statement */
7     return 0;                 /* return from program */
8   }
```

- o The preprocessor instructions refer to the instructions to the preprocessor of the compiler. Such instructions always start with `#` and are executed before the actual compilation starts. `#include <filename>` is simply replaced by the content of `filename`, which therefore includes a bunch of useful function definitions for usage in the present program.
  - o `main()` is the header of the program, every program starts with this header. `int` is the value returned to the caller of the program.
  - o The body of the program is enclosed by the braces `{}`
  - o The program contains two statements (terminated with `;`). A statement may be a simple statement or a compound statement.
  - o `printf("Hello World!");` prints `"Hello World!"` to the console.
  - o `return 0;` is the last statement in the program and returns 0 to the caller.
  - o Comments are added to explain to the reader what the program is doing, but these lines are ignored by the compiler.
    - ▪ Multi-line comments: enclosed by `/* … */`
    - ▪ Single-line comments: starting with `// …`

- Save the file in some directory (e.g. …/CS100/summer_homeworks/prog01.c)
- Compile your program
  - o Open a terminal
  - o Go to the directory of your summer_homeworks
  - o Compile the program, e.g.:
    - ▪ `gcc prog01.c`
    - ▪ `"gcc"` here represents a call to the GNU-C-Compiler
    - ▪ If you want to give a custom name to your executable, run `gcc prog01.c –o prog`
- Run the program: `./prog`

The examples and exercises in this booklet all involve simple single-file code examples, which can be compiled using (almost) any regular C compiler. For further details on how to open the terminal and on how to install and use a basic compiler on your system, please refer to the document CS100_build_environment_installation.pdf from the course webpage (http://mpl.sist.shanghaitech.edu.cn/CS100)

Lesson 2
Variables, expressions, and statements

A C program is composed of a series of statements, each one inducing one or several operations to be executed. In its simplest form, a statement is a line of code terminated by a semi-colon. There are a number of different types of statements:

- **Declaration statements:** A declaration is simply a statement through which we describe an identifier, such as the name of a variable. Through such declarations, the compiler will know which local variables will have to be allocated on the memory stack, and what identifier is used to refer to those variables within their scope. In its simplest form, a declaration statement takes the form:

type declarator

  where type denotes the type of the variable and declarator the name of that variable. A few primitive variable types sufficient for the solution of the preparatory summer homeworks are listed as follows:

`char`	8-bit (1-byte) variable representing a character (interchangeably used to represent 8-bit integer numbers)
`short, int, long`	Integer numbers for which the (physical) binary representation has a bit-width of at least 16, 16, and 32 bits, respectively (depending on computer and implementation)
`float, double`	Special representations for floating point numbers. Physical bit-width is 32 and 64 bits on most systems.

  You are invited to do some research by yourself to understand how for example the ASCII encoding scheme is used to define correspondences between numbers and characters such as the letters of the alphabet or other special characters. It is also useful to understand the binary number system, or floating-point number representations:

  https://en.wikipedia.org/wiki/C_data_types
  https://en.wikipedia.org/wiki/ASCII
  https://en.wikipedia.org/wiki/Single-precision_floating-point_format
  https://en.wikipedia.org/wiki/Double-precision_floating-point_format
  https://en.wikipedia.org/wiki/Binary_number

- **Expression statements:** Expression statements can be either just a single entity such as a variable or a combination of entities interconnected by operators. The following are simple examples of expressions:

```
a
a + b
```

```
x = y
t = u + v
x <= y
++j
```

It is useful to read up operator preference to understand in which order the different operations will be executed in case there is more than one. Also note that declaration and expression statements can be combined, as in

```
int a = b + c;
```

- **Compound statements:** Compound statements are in fact just a combination of several statements grouped by braces. The sub-statements can again be declaration statements, expression statements, or compound statements. An example would be:

```
{
  pi = 3.141593;
  circumference = 2. * pi * radius;
  area = pi * radius * radius;
}
```

- **Control statements:** Control statements are a bit different, as they use special keywords reserved for the compiler and will dynamically influence the program flow based on some conditions. The most basic control statement is the if-statement. It consists of an `"if"` keyword followed by a condition in parentheses. The condition is an expression that evaluates to either 0 or not (representing an unfulfilled or fulfilled condition, respectively). The closing parenthesis is followed by either a single expression or a compound statement to be conditionally executed. The if-statement may be followed by an else-statement, thus resulting in an if-else-statement. The else-statement simply consists of the keyword `"else"`, again followed by either a single expression or a compound statement, and defines alternative operations to be executed in case the condition is unfulfilled. The following indicates an example of an if-else-statement:

```
if( condition ) {
  statements
}
else {
  statements
}
```

Lesson 3 (Exercise 1)
Fibonacci number generator

The Fibonacci sequence of numbers is as follows:

0, 1, 1, 2, 3, 5, 8, 13, 21, 34, …

It is easily obtained by starting the sequence with the numbers 0 and 1, and then setting each subsequent number as the sum of the two previous ones. The sequence has an interesting application, as a proper arrangement of a sequence of squares of which the side lengths are given by the numbers in the Fibonacci sequence may serve as an approximation of the golden spiral.
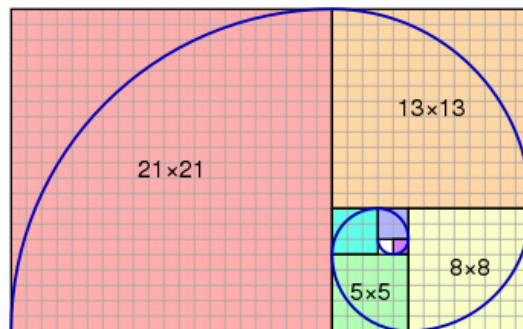
[source: https://en.wikipedia.org/wiki/Fibonacci_number]

The task of this problem is straightforward: Implement a Fibonacci number-generator program. It should simply print the first 90 numbers of the Fibonacci number sequence, as follows:

```
The next number is 0
The next number is 1
The next number is 1
…
```

You will need to use a special control statement, which is a for-loop. It has the syntax

```
for( int i = 0; i < someNumber; i++ ) {
    statements …
}
```

`"i"` is a variable used as a loop index, and the compound expression in parentheses right after the keyword `"for"` contains three expression statements. The first one initializes (and possibly even declares) the loop index variable. The second expression is used as a condition to decide whether or not more loops need to be executed (the condition will be checked at the beginning of every loop, including the very first one). The third expression will be evaluated at the end of each loop and is used to update the loop index variable. Note that the loop index variable may of course be used in the compound statement following the for-loop header.

In order to print custom numbers to the console, you may insert special format specifiers in the string literal to be printed using `printf`. For example, if you want to print two integers to the console, you can use

```
printf("%d %d\n", var1, var2);
```

%d here are special format specifiers that will be replaced sequentially by the values passed starting from the second parameter to the `printf` function. Every format specifier is of the form %x, where x is replaced according to the actual format of the data. Examples:

```
c: character
s: string
d: integer (int)
u: unsigned integer
f: floating point(float)
ld: integer (long)
…
```

"\n" is the EOL (End Of Line) character and will cause a line break in the console. Ask yourself how big the number becomes, and what bit-width is required!? The required output can be downloaded from the webpage.

Please follow the instructions on the webpage to login to our online grading system and submit your solution there. Please follow the exact instructions on how to register such that we are able to unambiguously identify you. This is very important as the summer-homeworks will be graded and already make up for a small part of your overall grade.

Lesson 4 (Exercise 2)
Mini calculator

In order to make programs that are more interesting than just a simple "Hello world" example that does the same thing everytime, we need control statements that depend on conditions that may possibly change each time we run the program. Two such control statements have already been introduced (for-loops and if-else-statements). The conditions may for example depend on a variable user input to the program. The simplest way to provide such a variable user input is through the console. Similar to the function `printf`, the header `<stdio.h>` also provides a function `scanf` to read user input from the console. The function simply reads characters input through the console, and in particular tries to read in a sequence of characters that matches a pre-defined format. The function furthermore lets us indicate variables to which the input will be copied.

For example, the following statements will aim at reading in a sequence of a floating point number, a character, and another floating point number from the console, and copy the result to three variables that have been declared upfront:

```
float var1;
float var2;
char op;
scanf("%f %c %f", &var1, &op, &var2);
getchar();
```

"%f %c %f" declares the format of the entire sequence of the characters to be read from the console, and "%x" is a special format specifier where x is to be replaced by either f for floating point numbers, d for integer numbers, or c for a single character (to again name just a few, the idea is similar than for the `printf` function). "getchar()" is added to empty the input queue and remove the EOL (End of Line) character added by hitting "Enter" on entering the sequence. Note the added character "&" in front of each variable. This is the "address-of"-operator that makes sure that we pass the address of the variables (i.e. pointers) rather than a copy of the variable itself. It ensures that we really modify the original variables. You will learn more on this topic during the class.

To create a condition based on the value of a certain character, you may for example write:

```
if( op == '+' ) {
   ...
} else {
   ...
}
```

Note that the operator == here is not an assignment operator, it is a comparison operator that produces either 0 or 1 depending on whether or not the left and right hand side expressions are identical.

Ok, you should now have all necessary tools together to solve the second exercise. Your task is to write a simple C program to evaluate arithmetic expressions. It should read in simple arithmetic expressions composed of two numbers and one of the four basic arithmetic operations, evaluate the result, and print it to the console. An example session will look as follows:

```
Laurents-MacBook-Pro:summer_homework laurent$ ./ex1
Enter the arithmetic expression to be evaluated (e.g. 5 + 6):
5 + 6
11.000000
Did you want to enter another expression? (Enter 'y' or 'n'): y
Enter the arithmetic expression to be evaluated (e.g. 5 + 6):
3.67 - 5.12
-1.450000
Did you want to enter another expression? (Enter 'y' or 'n'): y
Enter the arithmetic expression to be evaluated (e.g. 5 + 6):
12 / 4.234
2.834199
Did you want to enter another expression? (Enter 'y' or 'n'): y
Enter the arithmetic expression to be evaluated (e.g. 5 + 6):
3 * 9
27.000000
Did you want to enter another expression? (Enter 'y' or 'n'): n
Laurents-MacBook-Pro:summer_homework laurent$
```

Red text represents the user input. Note that rather than entering examples by hand, you may forward the content of a text-file to the console input. The output may also be redirected into a file. This is achieved as follows:

```
./myProgram < input.txt > output.txt
```

This is useful as you may use our examples provided on the webpage and check whether or not the produced output file exactly matches the provided example output. The online judge will also make such a direct comparison of the produced and the expected output, and it needs to match 100% (Note however that—in contrast to the text in the console—the text in the output file does not contain a copy of the user input including the line breaks added by the user). Use the provided test cases to figure out the problem if your test cases do not pass (note that differences as small as a missing or an added end-of-line command towards the end of the file may be sufficient to make the test cases fail).

Please follow the instructions on the webpage to login to our online grading system and submit your solution there. Please follow the exact instructions on how to register such that we are able to unambiguously identify you. This is very important as the summer-homeworks will be graded and already make up for a small part of your overall grade.

Lesson 5 (Exercise 3)
Angle between the hour and the minute hand of a clock

Older mechanical clocks do not only move the second or minute hands continuously at every second, but even the hour hand. The task of this exercise is straightforward. Given a time (in 24-hour format and with minute precision, as in 00:12, 13:04, etc.), write a C program that calculates the angle between the hour and the minute hand (in degrees). Examples input/output sessions with the program are given as follows:

```
What time is it? 10:10
The angle between the hands is 115.0 degrees.

What time is it? 21:45
The angle between the hands is 22.5 degrees.
```

The red text represents the user input. Further notes on correctness:

- The angle refers to the one that is less than 180 degrees in absolute terms (and should be reported without sign).
- Knowing that the time is in 24-hour format, you may have to deal with the hours upfront.
- Please format the output with one decimal digit.

It may be useful to familiarize yourself with the math library and some of its functions. Add `#include <math.h>` to the top of your program to include the math functions, and—depending on your system and compiler—you may also have to add the option –lm to your compiler command to activate the math functions. Some useful commands are:

```
int remainder = var1 % var2; //Takes var1 modulo var2,
                             //i.e. the remainder of
                             //the division of var1 by var2
fabs(var1);                  //Takes the absolute value of
                             //a floating point variable
```

Again, consider the input/output examples on the webpage to check the correctness of your code, and follow the instructions on the webpage to login to our online grading system and submit your solution there. Please follow the exact instructions on how to register such that we are able to unambiguously identify you. This is very important as the summer-homeworks will be graded and already make up for a small part of your overall grade.

Lesson 6 (Exercise 4)
Calculating 5000 digits of π

Your task in this exercise is to calculate 5000 digits of π. In particular, you will implement the Spigot algorithm as originally proposed by Stanley Rabinowitz and Stan Wagon. The algorithm has a few nice properties, which is that it has limited memory footprint (the memory footprint is essentially just a consequence of the required number of digits), the digits are computed one by one, and each iteration involves only integer computations. The following explanation is taken from the below link, please check it out for more visual descriptions of what the algorithm is supposed to do (Essentially the first page of the document is enough), and there is also an example Pascal code towards the end of the document.

http://stanleyrabinowitz.com/bibliography/spigot.pdf

Those who are interested in the origin of the algorithm (basis conversion) are encouraged to do further research online. Here we only explain how the algorithm works:

- The algorithm starts by defining an array *A* filled with the number *2*. The length of *A* is a consequence of the required number of digits *N*. In particular, if *N=5000*, the length is to be set to *len = floor(10 * N/3) + 1*.

- We assign a fractional coefficient to each column *i* (the first column being denoted by *i=1*). The numerator num_i is equal to *(i-1)* and the denominator den_i is equal to *(2 * i – 1)*.

- We now process each column starting from the right-most one and towards the left. For each column *i*, multiply the corresponding entry in *A* by *10* and add the "carry" from the previous column *i+1*. How to calculate the "carry" for each column is indicated below, and simply use *0* for the starting, right-most column.

- For the resulting value in the currently treated column *i* of *A*, calculate the quotient *q* and remainder *r* after division by den_i. Replace the value in *A* by the remainder *r*, and define the carry of the column as *q * num_i* (to be used for the next column, i.e. column *i-1*).

- A special case is given by column *1* (i.e. the left-most one). For this column, we divide the number by *10* rather than *1*. The quotient is the next digit of pi, and the remainder is retained as the first element in *A*.

There is only one tricky part in the algorithm, which is that the quotient may be 10 on some occasions. If this happens, the previous digit should be incremented by 1. If the previous digit is 9, the second-last digit should be incremented instead, etc.

To solve this problem, the smallest set of preceding digits that includes exactly one non-9 digit needs to be first buffered before being printed. Printing then happens conditionally and according to the following examples:

Buffer: [4], New digit: 3 → Print 4 and set buffer to [3]
Buffer: [4], New digit: 9 → Print nothing and set buffer to [4,9]
Buffer: [4], New digit: 10 → Print 5 and set buffer to [0]
Buffer: [4,9]. New digit: 2 → Print 49 and set buffer to [2]
Buffer: [4,9]. New digit: 9 → Print nothing and set buffer to [4,9,9]
Buffer: [4,9]. New digit: 10 → Print 50 and set buffer to [0]
Buffer: [4,9,9]. New digit: 0 → Print 499 and set buffer to [0]
Buffer: [4,9,9]. New digit: 9 → Print nothing and set buffer to [4,9,9,9]
Buffer: [4,9,9]. New digit: 10 → Print 500 and set buffer to [0]
…

Note that the buffer is always of the form of a single digit that is different from 9 followed by a certain number of 9's (possibly none). It is best to store the buffer as such rather than as a dynamically resized array.

The last exercise requires you to work with C-arrays. An array of a certain type and length can simply be defined by the declaration statement:

```
type arrayName[length];
```

and elements within this array can be referred to by using

```
type copy = arrayName[indexVariable];
```

or

```
arrayName[indexVariable] = newValue;
```

Note that length in the declaration statement needs to be a constant, not a variable. Though modern definitions of C permit variable sized arrays, the strict standard in fact does not preview this. Good luck!