CS244, STELLENBOSCH UNIVERSITY

TUT 1: INTRODUCTION TO C

22 JULY 2022

> On two occasions, I have been asked [by members of Parliament], "Pray, Mr Babbage, if you put into the machine wrong figures, will the right answers come out?" I am not able to rightly apprehend the kind of confusion of ideas that could provoke such a question.
>
> —Charles Babbage

INSTRUCTIONS

Please read items 1 to 4 carefully *before* starting the tutorial or asking questions.

1. If you work from the PDF version of this tutorial, do not copy and paste code from the document into vim, but type it over. Copying and pasting is a passive way of learning a language; typing, possibly making mistakes and having to correct them, is active way of engaging with a language. Complete solutions for the first six questions will be posted. However, they are worth very little if you do not try the questions on your own first.

2. Once we start the compiler project, we will use `make` to drive compilation. For now, however, you must invoke the C compiler from the command line. For example, if you want to compile the source file `hello.c` to the executable file `hello`, execute the following on the command line:

   ```
   $ cc -Wall -ansi -pedantic -o hello hello.c
   ```

   The dollar character is the command prompt, so you don't have to type it. Note that the `-o` switch precedes the name of the object file to which the source file will be compiled. The name of the object file does not have to be related to the name of the source file. If you do not specify an object file name with `-o`, by default the object file will be written to `a.out`. For particulars on the other options, refer to the `gcc` manual page by invoking `man gcc` at the command prompt.

   Be very careful not to overwrite your source file, for example, by accidentally typing the source file name after the `-o` option.

3. Unless your program compiles perfectly, the C compiler will report error and warning messages. An error message implies that compilation failed; a warning message means that the compiler is not entirely happy about something, but produced an executable nevertheless. Ignore the warnings to your own peril. For the project, if your submission compiles with warnings, you will be penalised. In other words, get rid of all warnings before moving on.

4. In Eclipse, the default behaviour when building a project is to compile the source files, and then, if compilation is successful, to execute the `main` method specified in the project configuration. Now, you must run the executable – the object file, automatically linked to the system libraries – yourself, after compiling it without error. To run an executable, simply type its *complete path* on the command line and hit ⟨Enter⟩. In any directory, the single dot refers to the current directory. Therefore, to run an executable called `hello` in the current directory, it suffices to run the following at the command line:

   ```
   $ ./hello
   ```

   The reason we are able to run system utilities without specifying a complete path is that they are typically stored in a sequence of directories called the **path**, which is searched when an executable name is entered in the shell. To see what the path contains, echo the `PATH` environment variable as follows.

```c
#include <stdio.h>

int main(int argc, char *argv[])
{
    printf("Hello, world!\n");
}
```

Listing 1: Displaying "Hello, world!"

```sh
#!/usr/bin/env bash
./hello
EXITCODE=$?
if [ "${EXITCODE}" -eq "0" ]; then
    echo "Command returned successfully"
else
    echo "Command terminated with failure code: ${EXITCODE}"
fi
```

Listing 2: Evaluating exit codes in a shell script.

```
$ echo $PATH
/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin:/usr/games
```

The directories are separated by colons and are searched in the order they appear in the variable.

5. Get to know the standard C libraries as soon as possible. Refer to http://www.csse.uwa.edu.au/programming/ansic-library.html or to K&R [2], even though you probably will not understand all the detail. You should try to read through the sections on input and output `<stdio.h>`, character class tests `<ctype.h>`, string functions `<string.h>`, mathematical functions `<math.h>`, utility functions `<stdlib.h>`, variable argument lists `<stdarg.h>`, and implementation-defined limits `<limits.h>` and `<float.h>`. Detailed descriptions of the functions are also available in §3 of the manual pages. For example, to read up on `printf`, execute the following at the command prompt:

   ```
   $ man 3 printf
   ```

EXERCISES

1. Begin by typing in and compiling the "Hello, world!" program (Listing 1) [2, p. 7]. If you used the switches specified above, you should get a warning: "Control reaches end of non-void function." You may have noticed that we specified an `int` return value for the `main` function. So, the compiler is complaining that there is no statement returning this expected `int` value from `main`.

   To remove the warning, add an inclusion directive for `<stdlib.h>`, and insert

   ```
   return EXIT_SUCCESS;
   ```

   as the last statement in `main`. As the constant name suggests, this constant indicates that a function has completed successfully. There is also an `EXIT_FAILURE` constant available.

   You may have wondered where the return value ends up. Such an exit code is returned to the parent process. If a command is executed from the command line, the exit code is available in the shell after the command terminates. For an idea of how this works, copy the code in Listing 2 to a file named `exitcode.sh`, make it executable (`chmod 700 exitcode.sh`), and finally, run this script file.

2. Listing 3 [2, p. 9] gives a program that prints a table of Fahrenheit temperatures and their Celsius equivalents.

```c
#include <stdio.h>
#include <stdlib.h>

int main(int argc, char *argv[])
{
    int fahr, celsius;
    int lower, upper, step;

    lower = 0;    /* lower limit of temperature table */
    upper = 200;  /* upper limit */
    step = 20;    /* step size */

    fahr = lower;
    while (fahr <= upper) {
        celsius = 5 * (fahr - 32) / 9;
        printf("%d\t%d\n", fahr, celsius);
        fahr += step;
    }

    return EXIT_SUCCESS;
}
```

Listing 3: Print a Fahrenheit–Celsius table.

```c
#include <stdio.h>
#include <stdlib.h>

int main(int argc, char *argv[])
{
    int i;
    for (i = 0; i < argc; i++)
        printf("%d: %s\n", i, argv[i]);
    return EXIT_SUCCESS;
}
```

Listing 4: Accessing command line arguments.

(a) Modify the program so that it uses a `for` loop instead of a `while` loop.

(b) Read up on the output conversions that apply to the format string of the `printf` function. Then modify the program so that the last digits in each column line up.

(c) Modify the program so that the variables `fahr` and `celsius` are of type `float`. Line up the second column of the table by the decimal points in the numbers.

(d) Use preprocessor definitions to define the magic numbers for the upper and lower limits, as well as the step size.

(e) Modify the program so that the upper and lower limits, and the step size is read in from the command line. Listing 4 gives an example of how to access the command line arguments to a program. Note that, in contrast to Java, the complete command line as entered in the shell, including the program path as it was called, is passed to the program. The `atoi` and `atof` library functions should be used to convert arguments to `int`s and `float`s, respectively.

3. Listing 5 [2, p. 17] gives a program that copies characters from the standard input to the standard output. The type `char` is specifically meant for character data, but actually, any integer type can be

```c
#include <stdio.h>
#include <stdlib.h>

int main(int argc, char *argv[])
{
    int c;
    while ((c = getchar()) != EOF)
        putchar(c);
    return EXIT_SUCCESS;
}
```

Listing 5: Copying input to output.

used. Here, we use `int`, since we want a data type large enough to handle an extra sentinel value `EOF` (from `<stdio.h>`) for the end of file.

Pay close attention to the `while` clause. In C, any assignment, such as

```
c = getchar()
```

is an expression and has a value, which is the value of the left-hand side after the assignment.

(a) Test your program without any command line arguments. After the program has been invoked, start typing, and as soon as you press ⟨Enter⟩, what you have typed will be echoed (again) on screen. ⟨Ctrl⟩-D signals the end of file (alternatively, end of input).

(b) You should also test your program with standard input and output redirection. For example, assuming you have compiled your program to `copy`, the first of the following commands copies the file `input.txt` to the screen; the second copies the file `input.txt` to the file `output.txt`.

```
$ ./copy <input.txt
$ ./copy <input.txt >output.txt
```

How can you use your program to create a new text file and add some text it?

(c) Modify the program so that, instead of copying standard input to standard output, it reads from standard input and counts the number of characters, words, and lines. Lines are ended by character `0x0A` (hexadecimal), normally specified as `'\n'`. Words are separated by white space, made up of spaces, end-of-line characters, and tabs.

(d) Extra work: Figure out how to open files, so that you don't have to use redirection. Use the functions `fopen`, `fclose`, and `fgetc`.

4. Write a program that reads lines from standard input, and once the end of input is reached, prints the longest. The following should be useful.

   - Use the preprocessor to define the magic number `MAXLINE`, the maximum input line size. For starters, set this value so that your character buffers are each 1 KiB.
   - Write a function with the following prototype that reads a line from standard input into `s` and returns the length of this line.

     ```
     int get_line(char s[]);
     ```

   - Write a function with the following prototype that copies one character array to another.

     ```
     void copy(char to[], char from[]);
     ```

5. For input, the counterpart of `printf` is `scanf`. Unfortunately, `scanf` requires an understanding of pointers, which we cover next week. For now, suffice it say that any scalar variable which is used as an argument must be prefixed by the `&` operator; arrays, for example, when scanning strings, are treated as normal, which is to say, without the `&` operator.

(a) Write a program that prompts the user for his/her name. Then display a message to the user, including the name that was read.

(b) Write a program that prompts the user for a date in the form `yyyy/mm/dd`. If the date entered is valid, calculate the day of the week with Zeller's Congruence, in modified form,

$$h = \left(q + \left\lfloor \frac{26(m+1)}{10} \right\rfloor + y + \left\lfloor \frac{y}{4} \right\rfloor + 6 \left\lfloor \frac{y}{100} \right\rfloor + \left\lfloor \frac{y}{400} \right\rfloor \right) \bmod 7, \tag{1}$$

where q is the day of the month, m is the ordinal number of the month, and y is the year. The ordinal number is what you would write a numeric date, that is, 3 for March, 4 for April, and so on. January and February are treated differently: They are months 13 and 14, respectively, *of the previous year.* Also, the day h is 0 for Saturday, 1 for Sunday, and so on. Display the answer as a string, for example, "Saturday".

Before calculating the day of the week, ensure that the day and month specified are in the proper ranges, providing for leap years. (In the Gregorian calendar, a year is a leap year if it is either divisible by four but not by 100, or if it is divisible by 400.) If the date is invalid for some reason, display an informative error message.

Hints: Organise the various parts of your program into functions. Store data like the number of days in each month in arrays. Also, Eq. (1) uses the floor function in four places, but you don't have to do anything special since C performs integer division when both the dividend and divisor are integral.

6. In reverse Polish notation – used, for example, in the PostScript page description language (PDL) – each operator follows its operands. An infix expression like

```
(1 - 2) * (4 + 5)
```

is entered as

```
1 2 - 4 5 + *
```

The reverse Polish notation is particularly amenable to computation with a stack. (PostScript, by the way, is a stack-based language.) Parentheses are unnecessary; the notation is unambiguous as long as we know how many operands each operator expects. The structure of a program is thus a loop that performs the proper operation on each operator and operand as it appears:

```
while next operator or operand is not end-of-file indicator do
    if number then
        push it
    else if operator then
        pop operands
        do operation
        push result
    else if newline then
        pop and print top of stack
    else
        error
    end if
end while
```

Write a C program that implements a calculator which uses the algorithm above to perform calculations on floating-point input expressions in the reverse Polish notation. Pay attention to the following:

- Separate your source code into three separate files:
  (a) `calc.c` contains the `main` function that implements the algorithm as given above;
  (b) `stack.c` contains an implementation of a stack; and

(c) `scanner.c` contains an implementation of a simple scanner.

For the stack and the scanner, provide header files so that these may be included in `calc.c`.

- Your stack should implement the following two functions:

```
void push(double f);
double pop(void);
```

Store the stack data in an array of type `double`; define a magic number for the stack size.

- Your scanner must implement the following function:

```
int getop(char s[]);
```

This function reads characters from the standard input, and returns an `int` indicating the type of token that was found: a number, one of the four operators (`+`, `-`, `*`, `/`), or the end-of-line character. If the token is a number, the string representation scanned from the standard input must be copied into `s`. White space must be ignored.

Sometimes it is the case that `getop` cannot determine that it has read enough until it has read too much. If you don't find another way around this problem, write two functions

```
int getch(void);
void ungetch(int c);
```

that gets a (possibly pushed back) character and pushes back a character, respectively. They don't actually attempt to push back anything, but instead, keep a buffer of characters available.

7. Implement a program that converts an infix expression to reverse Polish notation. It must be able to handle the four operators of the previous question, as well as left and right parentheses for grouping.

   Hints: Use your stack implementation from the previous question to store operands. Also, define a function that returns a level of precedence for of the operators and parentheses.

8. Combine ideas from the programs of the previous two questions, and write a calculator program that reads infix expressions and evaluates them.

9. Now might also be a good idea to revisit some sorting algorithms from Computer Science 214. For each of the following algorithms, write a driver program that reads a maximum of 10 000 integers from the standard input stream, sorts them, and then writes the sorted sequence to the standard output stream, one integer per line:

   - selection sort;
   - insertion sort;
   - quicksort; and
   - mergesort.

10. If you need more of challenge, consider this: Although mergesort has $\Theta(n \log n)$ performance, we can perhaps do better when the input is already somewhat sorted. The key is to understand that although insertion sort has $O(n^2)$ performance, it is also **adaptive**; in particular, when the input data is sorted or nearly sorted, insertion sort takes $O(n)$ time. So, what about this idea?—When a top-down mergesort reaches a given threshold with respect to the length of the subarrays (to sort recursively), it could switch to insertion sort as base case. Since mergesort has the added complexity of $O(n)$ auxiliary space, and insertion sort is both adaptive and in-place, doing so should speed things up for nearly-sorted data. Make such an implementation, first determining a suitable switchover threshold.

References

[1] Kernighan, Brian W. and Rob Pike. 1999. *The Practice of Programming.* Reading, Mass.: Addison Wesley.

[2] Kernighan, Brian W. and Dennis M. Ritchie. 1988. *The C Programming Language.* Second edition. Englewood Cliffs, NJ: Prentice Hall.