# A Exercises

# Exercise 1.1 – Hello world

- The goal of this exercise is to verify that computer and compiler setup are functioning correctly.
  - To verify that your setup runs fine, compile and run the “hello world” example

```
#include <iostream>
using namespace std ;

int main() {
   cout << “Hello World” << endl ;
   return 0 ;
}
```

# Exercise 1.2 – Counting chars in a string

- The goal of this exercise is to learn basic pointer skills.
  - We will write a C++ program that analyzes the character content of a string that is given to you by the user of your program. The program should display the number of uppercase and lowercase characters, the number of digits and the number of 'other' characters.
- Approach
  a) First write a small program that reads in a string from the terminal up to a newline character and prints that same string again to the terminal.
     - Use a 'classical string' in this exercise, i.e. an array of characters, not the STL class string.
     - You can read a string from the command using the std::cin.getline(char* str, int len)
  b) Add code that loops over the string and looks at the string one character at a time through a pointer. *You are **not** allowed to use an integer index variable.* Think about the following
     - How do you create a pointer and how do you make it point to the first character of the string?
     - How do you make the pointer point to the next character in the string?
     - How do you know that you're at the end of your string?
  c) Finally add code that determines if each given character is uppercase, lowercase, a digit or otherwise.
     - Think about how character literals work. First write a line of code that determines if a given character is the uppercase A.
     - Now modify this code to determine if the character is any upper case letter A-Z. Before you start this find out what the ASCII table looks like (e.g. Google ASCII table) and exploit the ordering of the ASCII characters.
     - Add similar code for lowercase characters, digits and 'other' characters and make your program print how many of each are in the string.

# Exercise 1.3 – Joining strings

- The goal of this exercise is to learn about string manipulation and dynamic memory allocation
  - Examine the following program

```
#include <iostream>
using namespace std ;

char* join(const char*, const char*) ;
char* joinb(const char*, const char*) ;

int main() {

  cout << join("alpha","bet") << endl ;
  cout << joinb("duck","soup") << endl ;

  return 0 ;
}
```

  - Write the missing `join()` and `joinb()` routines that will concatenate character strings
  - Function `joinb()` should insert a blank between the two strings, function `join()` shouldn't

# Exercise 1.3 – Joining strings

- Approach – writing `join()`
  a) Add an empty body for the function `join()` below the main function in your program code.
  b) Comment out the line using `joinb()` for now so you can test your `join()` code.
  c) What is the return type of `join()`? When you return a pointer in a function, it should point to a memory object you have allocated, so add some code that allocates a string to which the return value pointer can point to.
  d) Explain why you should use `new[]` to allocate this memory.
  e) What length should the allocated string be? Write your code such that exactly the right amount of memory is allocated. Do you need to allocate memory for the terminating NULL character?
     - Hint you can use the Standard Library routine `strlen()`, declared in `<string.h>` to determine the length of a string
  f) Use the Standard Library routine `strcat()` to implement the concatenation part of `join()`. Before you do so, first initialize your return value string to a null string. (Think about what a null string looks like)
  g) Who is responsible for deleting the returned memory? Is there a leak? How would you fix it?
- Approach – writing `joinb()`
  h) Start `joinb()` as a copy of `join()` and modify the concatenation part to insert a space between the two input strings
  i) Do any other parts of the code need to be modified?

# Exercise 1.4 – Effect of memory leaks

- Write a small program that deliberately leaks memory
- Watch what happens to the process memory allocation using the UNIX ‘top’ command in a separate window
  - What happens when memory runs out?

# Exercise 1.5 – Base 32 printing of integers

- The goal of this exercise is to learn how to use the bit-wise operators
  - Integers can be represented in several bases. Base-10 is the default (digits 0-9) but also Base-16 or hexadecimal (digits 0-9,A-F) is common in computing
  - What are the advantages of base-16 over base-10 in computing?
- Approach: write a program that reads in an unsigned integer and print it in a base-32 implementation
  - You can represent base-32 number with decimals 0-9 followed by characters A-V (just like hex numbers, which only go up to F)

a) Write a small program that reads in an unsigned integer.

b) Determine how many bits there are in an integer (use `sizeof`) and calculate how many base-32 digits you need to represent an integer

c) Allocate a temporary storage array of short ints to hold each digit.

d) Extract the first base-32 digits by isolating the 5 rightmost bits of the input integer. You can do this by masking out the remaining 27 bits using the bit-wise AND operator with a well chosen 'masking' integer. Store the digits value in the temporary array.

e) Process the remaining 5-bit digit by shifting all bits in the input integer to the right and repeating the above procedure until you are at the end.

f) Now that you have parsed the input into 5-bit digits, print them in base-32 representation. First make a `char[]` lookup table that converts a small integer (<32) used as index into that array into a Base-32 char (0-9,A-V).

g) Think about the order in which to print the digits. We started parsing at the least significant digit, but when you print you should start with the most significant digit. Write your print loop such that it takes that into account.