Appendix

To include the datatype `bool` in your code, include the header `<stdbool.h>`.
`#define Max 10` **Max is called a defined symbolic constant**

int printf(const char *format, <variables>)

- **format**: the string that contains the text to be written. It can optionally contain embedded format tags that are replaced by the values of <variables>
- If successful, the total number of characters written is returned. On failure, a negative number is returned.

int scanf(const char *format, ...)

- **format** : the C string that contains embedded tags on the type and format of data to read.
- On success, the function returns the number of items matching the format which were successfully read.

```
switch (expression ){
  case value1 :
    statement-list1
  case value2 :
    statement-list2
  ….
  default:
default_statements
}
```

```
if ( expression 1) {
   statement-list1
} //
end if (expression 2){
   statement-list2
}
else {
  statement-list3
} //
```

```
while( loop repetition condition )
{
statements;      /* loop body */
}
```

```
for ( initialization; repetition condition; update )
{
statements;      /* loop body */
}
```

```
do
{
          statement_list;
}
while (loop repetition condition);
```

&&: Logical AND
|| : logical OR
! : logical NOT

```
typedef struct {
  datatype1 variableName1;
  datatype2 variableName2;
…..
} <struct_Type_Name> ;
```
Example: <struct_Type_Name> is Student_t;

```
typedef union {
  datatype1 variableName1;
  datatype2 variableName2;
…..
} <union_Type_Name> ;
```
Example: <union_Type_Name> is Student_t;

```
typedef enum {
  symbol1,
  symbol2,
……
…symbolX
} <enum_Type_Name> ;
```

You can use the following function to **clear the input data buffer:**

```
void skip_line(){
   #define LINE_SIZE 100
   char line [LINE_SIZE];
   scanf("%[^\n]s", line);
}
```

- **Dynamic Memory allocation:**
  void *malloc(size_t size) , where size is the size of the requested memory block, in **bytes.**
  To use it, include the header `<stdlib.h>`

String Manipulation:

char name[31];

- Read a string : **scanf(**"%s", name);
- Write a string: **printf**("%s", name);
- You need to include **<string.h> header file** to use the following functions:

Function name	Description
`strcat(s1, s2)`	Appends `s2` to `s1` (also `strncat`)
`strcmp(s1, s2)`	Return 0 if (`s1` is same as `s2`), -1 if (`s1 < s2`), 1 if (`s1 > s2`)
`strcpy(s1, s2)`	copy `s2` to `s1`
`strncpy(s1, s2, n)`	copy n characters from s2 to s1 (substrings)
`strlen( s1 )`	return the length of `s1` up to (but not including) the null character
….	….

Open a file:

FILE * fs**= fopen(const char** filename[], **const char*** mode**);**

where

	Text file	Binary file
Mode Access	`mode` parameter	`mode` parameter
Read Only	"r"	"rb"
Write only	"w"	"wb"
Append	"a"	"ab"

File read operation: where FILE *fs already initialized through the function **fopen,** see above

Text file	Binary file
fscanf(fs, " conversion character", variable address); **Example:** **fscanf**(fs, "%d",&i); **fscanf(**fs, "%s",arr); ▪ If successful, the **fscanf()** function returns the number of receiving arguments successfully assigned. ○ Example: while(fscanf(fs, "%f", &i)==1) {…}	The **fread** function reads a given number of items (numOfItems), all of the same size (sizeOfItem), from a memory buffer position (buff) into a binary file stream (fs): **int fread(** **void** *buffer, /*buffer */ **int** sizeOfItem, /*size of items */ **int** numOfItems, /*number of items */ **FILE** *fs); /* file stream ID */ **fread** returns the actual number of items read from the file: • **Example:** **char** buff[30]; **fread(**buff**, sizeof(char),**30,fs); • **char** in the above is **the data type** of the **buff elements** and **30** is the number of elements to transfer. • If **"buff" is not a pointer or array,** prefix the variable name with **&** • **on successful read, `fread` returns 30!**

File write operation:

Text file	Binary file
fprintf(fs, " conversion character",variable address) **Example:** **fprintf**(fs, "%d",i); • If successful, the fprintf() function returns number of characters written. • If it fails, it returns (for simplification!) EOF	The **fwrite** function writes a given number of items (numOfItems), all of the same size (sizeOfItem), from a memory buffer position (buff) into a binary file stream (fs): **int fwrite**(**void ***buffer, /*buffer */ **int** sizeOfItem, /*size of items */ **int** numOfItems, /*number of items */ **FILE** *fs); /* file stream ID */ **fwrite** returns the actual number of items written into the file: • **Example:** **char** buff[30]; **fwrite(**buff**, sizeof(char),**30,fs); • **char** in the above is **the data type** of **buff elements** and **30** is the number of elements to transfer. • If **"buff" is not a pointer or array,** prefix the variable name with **&** • **On successful write, fwrite returns 30!**

For a file access, with a file stream reference **fs**:
The function **feof(**fs**) returns true** if **the end of file has been reached; false otherwise.**
The function **ferror(**fs**) returns true if a hardware failure occurs during the processing of the file; false otherwise**

Closing a file: fclose(fs)

Basic C++ Program structure:

```
#include<iostream>
using namespace std;
...
int main(){
...
}
```

An example of a C++ template function (reverse) prototype :

```
template < typename R>
R reverse ( R x );
```

C++ manipulators are used to format the output:

- **setprecision(n)**: sets the number of decimal places to n
- **showpoint:** shows the decimal point even when the decimal part is 0
- **setw(n):** sets the width of the output field to n positions
- **fixed:** print the real-value in fixed point notation (representation)
- **left:** sets left justified output in the output field
- **right:** sets right justified output in the output field
  For the above, the header **iomanip** should be included.