

**CSE1303 Part A**  
**Data Structures and Algorithms**  
**Summer Semester 2003**

**Lecture A1 – Welcome & Revision**

Kymerly Fergusson

1

**Overview**

- Important Information.
- Overview of the course.
- Review.

2

**Information – Part A**

- Lecture Notes. Available at University Bookshop & online
  - Practical Classes Notes. } Handed out in lectures
  - Tutorial Exercise Sheets. } & available online
  - Text Book.
- Recommended:
- Kruse R, Tondo C and Leung B. Data Structures and Program Design in C. Prentice Hall, 1997.
  - Standish T A. Data Structures, Algorithms and Software Principles in C. Addison-Wesley, 1995.
  - Y.Langsam , et al. Data Structures using C and C++. Prentice Hall, 1990.
  - K.N. King, C Programming A Modern Approach, Norton, 1996
  - H.M. Deitel and P.J. Deitel, C: How to Program, 2<sup>nd</sup> Edition, Prentice Hall, 1994

3

**Contact Details**

Kymerly Fergusson  
 kef@mail.csse.monash.edu.au  
 Room 131, building 26  
 9905 5222

<http://www.csse.monash.edu.au/~kef>

4

**Consultation Times**

Tuesday 10:00 – 11:00, 2:00 – 3:00  
 Wednesday 10:00 – 11:00, 2:00 – 3:00  
 Friday 1:00 – 3:00

Room 131, Building 26

\* Note: Kym is not in on Mondays and Thursdays.

**Courseware**

<http://www.csse.monash.edu.au/courseware/cse1303>

5

**Overview of CSE1303 Part A**

- Basic data structures.
- How to manipulate them.
- How to implement them.
- Various algorithms and their implementations.
  - Searching
  - Sorting
- Complexity of algorithms

6

## *Data Structures and Algorithms*

- Programs.
- Different problems.
- Operations.
- Size of data.
- Resources available.

7

## *Revision - This lecture*

- Basic C data types
- Boolean
- 1D and multidimensional arrays
- Strings
- Input/Output
- File I/O
- Structures and `typedef`

8

## *Review - Basic Data types in C*

- `int`
- `char`
- `float`
- `double`

9

## *Review - int vs float*

- Integer division in **C** rounds down:  
 $4/5 = 0$
- `float` is a "communicable" type:  
 $4.0/5 + 2 - 1$   
 $= 0.8 + 2 - 1$   
 $= 1.8$

10

## *Review - char*

- Characters are stored as a small integer
- Each character has a unique integer specified in an **ASCII** table.
  - (See appendix D in Deitel and Deitel)
- **ASCII** values range from 0 - 127
  - 0 is the value for the special character `'\0'`
  - 127 is the value for `<DEL>`
  - Other special characters:  
`\n \t \ \ etc`
- There are other extended character sets (extending from 128 - 255)

11

## *Review - Type Modifiers*

- `long`
  - Allows for large numbers to be stored.
  - `long int num;`
  - `double` is a long `float`
- `short`
  - Stores smaller numbers, taking less space
  - `short int x;`
- `const`
  - Makes a variable constant (not able to be changed).
  - `const int p = 5;`
  - Must be initialised at declaration

12

### Review - Boolean

- Has two values, **true** and **false**.
- In C we use integers as Booleans.
- Zero represents **false**.
- Any non-zero integer represents **true**.
- for **gcc** you need to include the library `<stdbool.h>`. (This doesn't work in Borland)
- In Borland programs, use **#define** to define the constants **true** and **false**

13

### Review - Boolean

```
#include <stdbool.h>

bool leapYear(int year)
{
    if ((year % 4 == 0 && year % 100 != 0)
        || (year % 400 == 0) )
    {
        return true;
    }
    else
    {
        return false;
    }
}
```

14

### Review - Boolean

For Borland use `#define orconst int`

```
#define true 1
#define false 0
#define bool int

bool leapYear(int year)
{
    if ((year % 4 == 0 && year % 100 != 0)
        || (year % 400 == 0) )
    {
        return true;
    }
    else
    {
        return false;
    }
}
```

15

### Review - Variables

- Is a logical **name** or **identifier** for a piece of memory for storing values.
- Has a type associated with it.
- Must be declared before being used:
 

```
int x
float sum;
```
- May be initialised at declaration:
 

```
char choice = 'a';
```
- Identifiers may contain letters, digits and underscores, but may not start with a digit.

16

### Review - Variables

- Variables can be **local** or **global**
- Global variable declarations are made at the start of the file after any **#define** and **#include**
  - Global variables are available to be used anywhere throughout the program.
- Local variable declarations are made at the start of a function.
  - Local variables are only able to be used inside the function where they are declared.

17

### Review - Variables

```
#include <stdio.h>
#include <stdbool.h>
#define MAXSTRING 100
char name[MAXSTRING];

int main()
{
    bool flag = true;
    while (flag != false)
    {
        if (scanf("%s", name) == 1)
            printf("Hello %s", name);
        else
            flag = false;
    }
}
```

18

### Review - Functions

```
#include <stdio.h>
#include <stdbool.h>

bool leapYear(int year)
{
    if ((year % 4 == 0 && year % 100 != 0)
        || (year % 400 == 0) )
    {
        return true;
    }
    else
    {
        return false;
    }
}
```

**File inclusion header**

**Function definition**

19

### Review - Functions

```
#include <stdio.h>
#include <stdbool.h>

bool leapYear(int year)
{
    if ((year % 4 == 0 && year % 100 != 0)
        || (year % 400 == 0) )
    {
        return true;
    }
    else
    {
        return false;
    }
}
```

**Function name**

**Function parameter**

**Parameter type**

20

### Review - Functions

```
#include <stdio.h>
#include <stdbool.h>

bool leapYear(int year)
{
    if ((year % 4 == 0 && year % 100 != 0)
        || (year % 400 == 0) )
    {
        return true;
    }
    else
    {
        return true;
    }
}
```

**Function return type**

**Must be compatible with the function's return type**

21

### Review - Functions

```
int main()
{
    int year, month, day;

    printf("Enter year, month and day: ");
    scanf("%d %d %d", &year, &month, &day);

    day = dayOfYear(year, month, day);

    printf("\nDay of Year = %d\n", day);
}
```

**Function call**

22

### Review - Arrays (1D)

array1D:

--	--	--	--	--

0
1
N - 1

- All the *elements* are always of the same type.
- An element: `array1D[index]`
- In C, the first element has index 0 (zero).

23

### Review - Multidimensional Arrays

array2D:


- Arrays of arrays.
- All the elements are always of the same type.
- An element: `array2D[row][column]`

24

### Review - Multidimensional Arrays

```
int dayTable[2][13] = {
    {0, 31, 28, 31, 30, 31, 30, 31, 31, 30, 31, 30, 31},
    {0, 31, 29, 31, 30, 31, 30, 31, 31, 30, 31, 30, 31}
};

int dayOfYear(int year, int month, int day)
{
    int i;
    int isLeap = leapYear(year);
    for (i = 1; i < month; i++) {
        day += dayTable[isLeap][i];
    }
    return day;
}
```

2-dimensional array of int

25

### Review - Multidimensional Arrays

```
int dayTable[2][13] = {
    {0, 31, 28, 31, 30, 31, 30, 31, 31, 30, 31, 30, 31},
    {0, 31, 29, 31, 30, 31, 30, 31, 31, 30, 31, 30, 31}
};

int dayOfYear(int year, int month, int day)
{
    int i;
    int isLeap = leapYear(year);
    for (i = 1; i < month; i++) {
        day += dayTable[isLeap][i];
    }
    return day;
}
```

Index goes from 0 to 12

26

### Review - Input/Output

- Input/Output is done via **streams**
- Uses the library `stdio.h`
- Streams that are available in the library are `stdin` (keyboard), `stdout` and `stderr` (screen). These can be redirected.
- Data is written to `stdout` using the `printf()` function.

```
printf("Enter your name: ", name, age, gender, idnumber);
```

Format control string      Variables containing data to be printed

- Data is read in from `stdin` using the `scanf()` function.

```
scanf("%s %d %c %d", name, &age, &gender, &idnumber);
```

Conversion specifiers      Pointers to variables where input will be stored

27

### Review - Input/Output

- `scanf()` returns the number of values successfully read and converted or returns a special value `EOF` when input ends.
- Note for when reading a single character (`%c`): if there is a `\n` character left in the buffer from reading another value (`%d`) then that `\n` will be read into your character variable.
- Conversion specifiers:
  - `i` or `d`: display a signed decimal integer
  - `f`: display a floating point value
  - `e` or `E`: display a floating point value in exponential notation
  - `g` or `G`: display a floating point value in either `f` form or `e` form
  - `l`: placed before any float conversion specifier to indicate that a `long int` is displayed (`%ld`)

28

```
#include <stdio.h>

int main()
{
    int day;
    int month;
    int year;
    char name[30];

    printf("Enter your name: \n");
    scanf("%s", name);

    /* skipping spaces */
    printf("Hi %. Enter birthdate as: dd mm yyyy\n", name);
    scanf("%d %d %d", &day, &month, &year);

    /* alternative */
    printf("Hi %. Enter birthdate as: dd-mm-yyyy\n", name);
    scanf("%d-%d-%d", &day, &month, &year);

    return 0;
}
```

Note: no ampersand for strings

Conversion specifier

Literal characters

29

### Review - Strings

- Strings**: array of characters + `'\0'` (Max length including '\0')
- Example: `char name[30];`
- Unlike other arrays, strings must have an end-of-string character: `'\0'`
- String functions you will use from the `string.h` library include:
  - Length - `strlen(string1)`
  - Assignment - `strcpy(dest, source)` (Copies string2 onto the end of the destination string)
  - Concatenation - `strcat(dest, string2)`
  - Comparison - `strcmp(string1, string2)`

Returns: positive if string1 sorts after string2, 0 if they are the same string, negative if string1 sorts before string2

30

### Review - Strings

- An array of characters compared to a string:

C	A	T	S
C	A	T	\0

- The length of the string as returned by `strlen()`, is 1 element shorter than the array – it does not count the `\0` (end of string character)
- Strings are enclosed in double quotes ("hello") and contain the `\0` character at the end
- Single characters use single quotes ('h')

31

```
#include <stdio.h>
#include <string.h>

#define MAXLENGTH 100

int main()
{
    char string1[MAXLENGTH];
    char string2[MAXLENGTH];

    strcpy(string1, "Hello World!");
    strcpy(string2, string1);

    length = strlen(string1);
    printf("length of string1 = %d\n", length);

    strcpy(string1, "Apple");
    strcpy(string2, "Orange");
}
```

string1 needs to fit the number of characters of the second string, +1 for the '\0' character

string2 needs to be the same length as string 1

32

```
if (strcmp(string1, string2) < 0)
{
    printf("%s %s\n", string1, string2);
}
else if (strcmp(string1, string2) == 0)
{
    printf("The strings are the same.\n");
}
else
{
    printf("%s %s\n", string2, string1);
}

strcat(string1, " juice");
printf("%s\n", string1);

return 0;
}
```

Prints the order which the two strings sort, alphabetically

Note: To scan within a string use: `scanf(string, "%d", int);`

33

### Review -File Handling in C

- File Input/Output is also done via *streams*
- Files need to be *opened* before use.
  - Associate a "file handler" to each file
  - Modes: read, write, or append
- File input/output functions use the file handler (*not* the filename).
- Some examples of file handlers you know: **stdin, stdout, stderr**

34

### Review -File Handling in C

- Need to check the file opened successfully.
- Need to *close* the file after use.
- Basic file handling functions: **fopen()**, **fclose()**
- File I/O functions: **fscanf()**, **fprintf()**, **fgets()**.

35

```
#include <stdio.h>
#define MAXLEN 100

int main()
{
    FILE *inputfile = NULL;
    FILE *outputfile = NULL;
    char name[MAXLEN];
    int count;
    float mark;

    inputfile = fopen("Names.txt", "r");
    outputfile = fopen("marks.dat", "w");

    if (inputfile == NULL)
    {
        printf("Unable to open input file.\n");
        return 1;
    }
    if (outputfile == NULL)
    {
        printf("Unable to open output file.\n");
        return 1;
    }
}
```

Associate a file handler for every file to be used.

Mode  
r : read  
w : write  
a : append

fopen () returns NULL if an error occurs

36

```

count = 0;
while ( fscanf(inputfile, "%s", name) == 1 )
{
    count++;

    printf("Enter mark for %s: \n", name);
    scanf("%f", &mark);

    if ( fprintf(outputfile, "%s %f\n", name, mark) <= 0 )
    {
        printf("Error writing to output file.\n");
        return 1;
    }
}

printf("\n");
printf("Number of names read: %d\n", count);

fclose(inputfile);
fclose(outputfile);

return 0;
    
```

*fscanf() returns the number of values read and converted*

*fprintf() returns the number of successfully written or negative if an error occurs*

37

```

#include <stdio.h>
#define MAXLEN 100

int main()
{
    FILE *inputfile = NULL;
    char line[MAXLEN];
    int count = 0;

    inputfile = fopen("Names.txt", "r");
    if (inputfile == NULL)
    {
        printf("Unable to open input file.\n");
        return 1;
    }

    while(fgets(line, MAXLEN, inputfile) != NULL)
    {
        count++;
    }

    printf("Number of lines: %d", count);
    fclose(inputfile);
    return 0;
    
```

To read in a line, use fgets(). fgets() returns NULL if end of file is reached.

*fgets(string, length, filehandle)*

What would happen if you tried to count the number of lines again, once the end of the file has been reached?

38

### Review - struct

structname:

- Members may have different types.
- structname.membername
- structs are also known as "records," and members as "fields"

39

### Review - typedef

- Gives a new name to a type that has already been defined.
- E.g.
 

```
typedef struct StudentRec Student;
```
- Saves typing `struct StudentRec` everywhere.
- Or you can use it:
 

```
typedef struct
{
    int ID;
    float mark;
} Student;
```

40

**Recall:**

```

#include <stdio.h>
#define MAXNAME 80

struct StudentRec
{
    char name[MAXNAME];
    int mark;
};

typedef struct StudentRec Student;
    
```

Macro substitution

Structure declaration

Data type

New type name

41

**Recall:**

```

#include <stdio.h>
#define MAXNAME 80

struct StudentRec
{
    char name[MAXNAME];
    int mark;
};

typedef struct StudentRec Student;
    
```


Structure name / tag

Members

Don't forget this!

42

**Recall:**



```

Student readStudent(void)
{
    Student next;
    scanf("%s %d", next.name, &next.mark);
    return next;
}

"Package"
void printStudent(Student student)
{
    printf("%s %d\n", student.name, student.mark);
}
    
```

Type

An instance of the struct

A member of a struct instance

43

```

#define MAXCLASS 100

main()
{
    Student class[MAXCLASS];
    int n, i, best = 0;

    printf("Enter number of students: ");
    scanf("%d", &n);

    for (i = 0; i < n; i++) {
        class[i] = readStudent();
        if (class[best].mark < class[i].mark) {
            best = i;
        }
    }


    printf("Best student is: ");
    printStudent(class[best]);
}
    
```

Array of instances of structs

Assignment

Member of an array element

**Recall:**



44

**Revision**

- Basic Data Types and booleans
- I/O and File I/O
- Arrays and Structs
- Strings
- Typedef

**Revision: Reading**

- Kruse - Appendix C
- Deitel & Deitel - Chapters 1, 2, 3, 4, 5, 6, 8, 9, 11, 12, 13
- King - Chapters 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 13, 16, 22,
- Langsam - Chapters 1.1 – 1.3 inclusive

**Preparation**

Next lecture: **Pointers**

- Read Chapter 7 in Deitel and Deitel
- Read Appendix C.6 in Kruse et al.

45