

WELCOME

CSE1303 Part A

Data Structures and Algorithms


Summer 2002

KyMBERly Fergusson

Overview

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

Information – Part A

- Lecture Notes. *Available at University Bookshop & online*
- Practical Classes Notes. 
- Tutorial Exercise Sheets. *Handed out in lectures & available online*
- Text Book.
 - *Prescribed:* Kruse R, Tondo C and Leung B. Data Structures and Program Design in C. Prentice Hall, 1997.
 - *Recommended:* 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.

Consultation Times

Thursday 11:30am -12:30pm,

2:30pm – 3:30pm

Friday 11:30am -12:30pm

2:30pm – 3:30pm

Room 131, Building 26

Contact:

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

kef@mail.csse.monash.edu.au

Room 131, building 26

9905 5222

Courseware:

<http://www.csse.monash.edu.au/courseware/cse1303s>

Data Structures and Algorithms

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

Overview of Part A

- Basic data structures.
- How to manipulate them.
- How to implement them.
- Other algorithms and how to implement them.

This Lecture - Review

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

Basic Data types in C

- int
- char
- float
- double

Boolean

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

```
#include <stdbool.h>

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

For Borland use #define or const int

```
#define true 1  
#define false 0
```

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



Recall:

File inclusion header

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

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



Recall:

Function definition

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

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



Recall:

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

Function name

```
bool leapYear(int year)
```

```
{
```

```
    if ((year % 4 == 0 && year % 100 != 0)  
        || (year % 400 == 0) )
```

```
    {
```

```
        return true;
```

```
    }
```

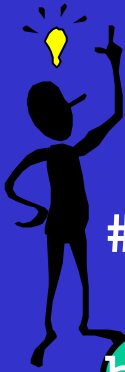
```
    else
```

```
    {
```

```
        return false;
```

```
    }
```

```
}
```



Recall:

Function return type

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

```
bool leapYear(int year)
```

```
{
```

```
    if ((year % 4 == 0 && year % 100 != 0)  
        || (year % 400 == 0) )
```

```
    {
```

```
        return true;
```

```
    }
```

```
    else
```

```
    {
```

```
        return true;
```

```
    }
```

```
}
```

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



Recall:

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

```
bool leapYear(int year)
```

```
{
```

```
    if ((year % 4 == 0 && year % 100 != 0)  
        || (year % 400 == 0) )
```

```
    {
```

```
        return true;
```

```
    }
```

```
    else
```

```
    {
```

```
        return false;
```

```
    }
```

```
}
```

Function parameter

Parameter type



Recall:

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

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

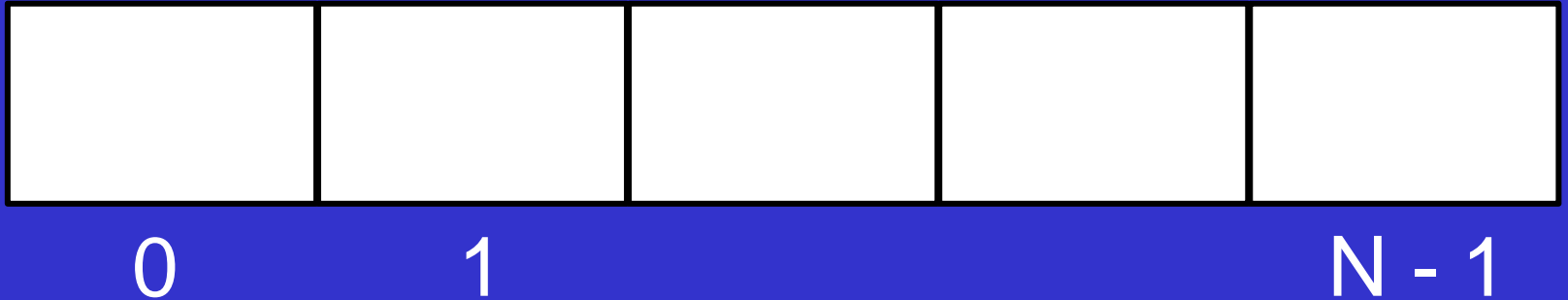
    day = dayOfYear(year, month, day);

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

Function call

Review - Arrays (1D)

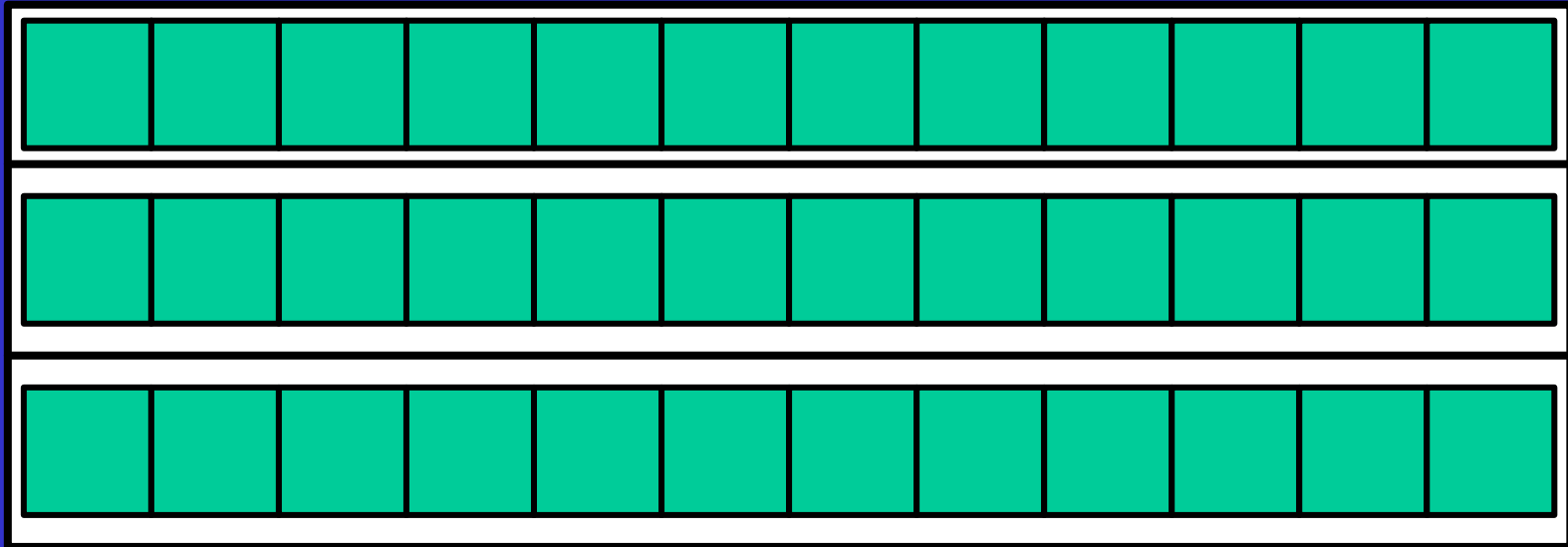
array1D:



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

Review - Multidimensional Arrays

array2D:



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

```
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;
}
```



Recall:

```
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}  
};
```

Global variable

```
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;  
}
```

Local variable



Recall:

```
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



Recall:

```
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}  
};
```

Index goes from 0 to 12

```
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;  
}
```

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("%s\n%f\n%c\n%d\n", 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 %f %c %d", name, &age, &gender, &idNumber);
```

Conversion specifiers

Pointers to variables where input will be stored

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 double is displayed

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

```
int main()
```

```
{
```

```
    int day;
```

```
    int month;
```

```
    int year;
```

```
    char name[30];
```

```
    printf("Enter your name:\n")>
```

```
    scanf("%s", name);
```

Note: no ampersand for strings

```
    /* skipping spaces */
```

```
    printf("Hi %s. Enter birthdate as: dd mm yyyy\n", name);
```

```
    scanf("%d %d %d", &day, &month, &year);
```

Conversion specifier

```
    /* alternative */
```

```
    printf("Hi %s. Enter birthdate as: dd-mm-yyyy\n", name);
```

```
    scanf("%d-%d-%d", &day, &month, &year);
```

Literal characters

```
    return 0;
```

```
}
```

Review - Strings

- **Strings** : array of characters

*Max length including
'\0'*

- **Example:** `char name[30];`

- Unlike other arrays, strings require an end-of-string character : `'\0'`

- String functions you will use from the `string.h` library include:

Length - `strlen(string1)`

Assignment - `strcpy(dest, source)`

Concatenation - `strcat(dest, string2)`

Comparison - `strcmp(string1, string2)`

*Copies string2 onto
the end of the
destination string*

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

```
#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

```
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:
sscanf(string1, "%d", int1);*

Review -File Handling in C

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

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

Associate a file handler for every file to be used.

```
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;
}
```

Mode
r : read
w : write
a : append

fopen() returns
NULL if an error
occurs

```
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

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

```
#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++;
    }

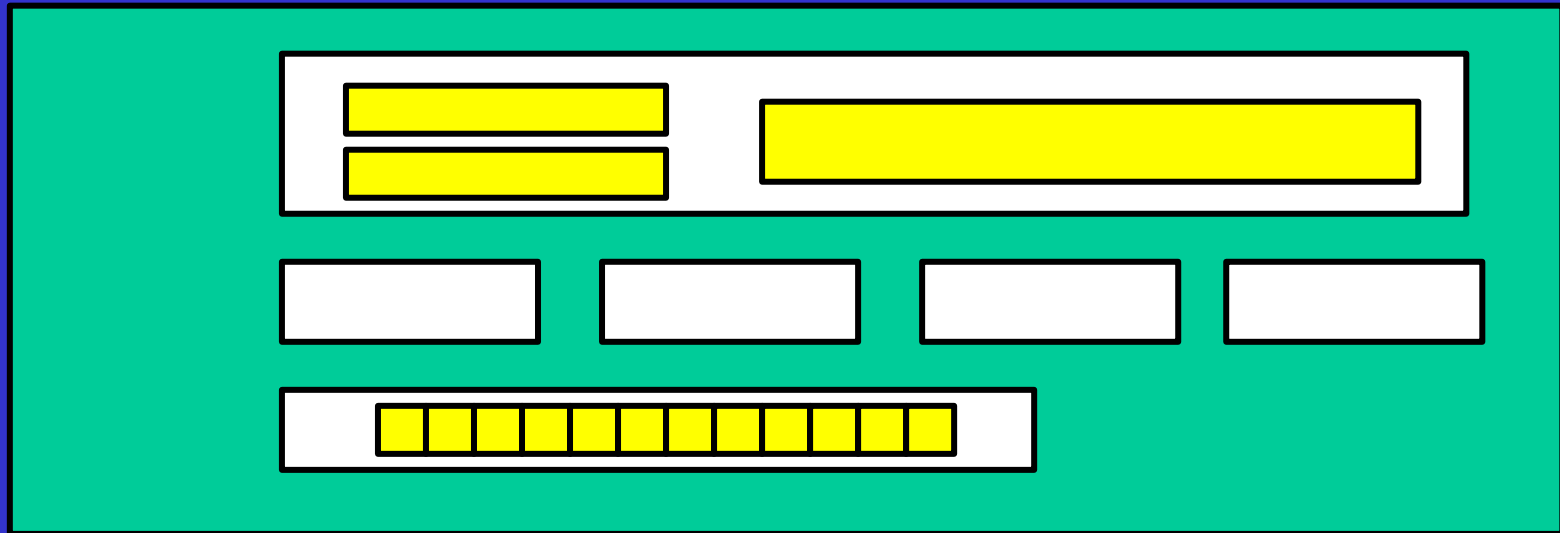
    printf("Number of lines: %d", count);
    fclose(inputfile);
    return 0;
}
```

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?

Review - struct

structname:



- Members may have different types.

structname.membername

- structs are also known as “records,” and members as “fields”

Review - typedef

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

```
typedef struct StudentRec Student;
```

- Saves typing **struct whatever** everywhere.

Example :

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

```
#define MAXNAME 80
```

```
struct StudentRec
```

```
{
```

```
    char    name[MAXNAME];
```

```
    int     mark;
```

```
};
```

```
typedef struct StudentRec Student;
```



Recall:

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

```
#define MAXNAME 80
```

Macro substitution

```
struct StudentRec
```

```
{
```

```
    char    name[MAXNAME];
```

```
    int     mark;
```

```
};
```

```
typedef struct StudentRec Student;
```



Recall:

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

```
#define MAXNAME 80
```

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

```
typedef struct StudentRec Student;
```

Structure declaration



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!



Recall:

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

```
#define MAXNAME 80
```

```
struct StudentRec
```

```
{  
    char   name[MAXNAME];  
    int    mark;  
};
```

```
typedef struct StudentRec Student;
```

Data type

New type name

```
Student readStudent(void)
```

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

```
void printStudent(Student student)
```

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



Recall:

```
Student readStudent(void)
```

```
{
```

```
    Student next;
```

An instance of the struct

```
    scanf("%s %d", next.name, &next.mark);
```

```
    return next;
```

A member of a struct variable

```
}
```

"Package"

```
void printStudent(Student student)
```

```
{
```

```
    printf("%s %d\n", student.name, student.mark);
```

```
}
```

```
#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]);
```

```
}
```

```
#define MAXCLASS 100
```

```
main()
```

Array of instances of structs

```
{
```

```
Student class[MAXCLASS];
```

```
int n, i, best = 0;
```

Recall:



```
printf("Enter number of students: ");
```

```
scanf("%d", &n);
```

Assignment

```
for (i = 0; i < n; i++) {
```

```
class[i] = readStudent();
```

```
if (class[best].mark < class[i].mark) {
```

```
best = i;
```

```
}
```

Member of an array element

```
}
```

```
printf("Best student is: ");
```

```
printStudent(class[best]);
```

```
}
```

Revision

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

Preparation

Next lecture: Pointers

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