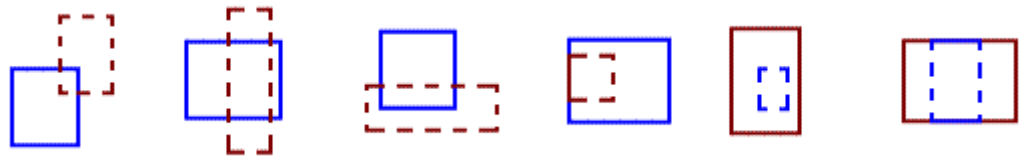
CS1010E Programming Methodology
Semester 1 2016/2017

Week of 7 November – 11 November 2016
Tutorial 11
Past Year Papers

1. Below are some examples of overlapping rectangles.



(a) An axis-parallel rectangle is defined by two points: *min* (bottom-left) and *max* (top-right), where each point is defined by two floating point values x and y. Define structures `Point` and `Rect` to represent points and rectangles respectively.

(b) Write a function `generateRect` that generates a random rectangle such that the rectangle always lies in the interval $[0, 1]$ with respect to both the x-axis and y-axis.

```
Rect generateRect(void);
```

To ensure that the points *min* and *max* are the respective bottom-left and top-right of the rectangle (i.e. $min_x < max_x$ and $min_y < max_y$), first generate a random point for *max*, then use the x and y coordinate values of *max* to generate a random point for *min*.

(c) Write a function `printRect` to print a given rectangle in the following manner.

```
Rectangle (0.141122,0.312640) --> (0.205298,0.776788)
```

```
void printRect(Rect rect);
```

The two coordinates correspond to *min* and *max* points of the rectangle respectively.

(d) Write a function `overlap` that takes in two rectangles `r1` and `r2` as arguments and returns `true` if they overlap, `false` otherwise.

```
bool overlap(Rect r1, Rect r2);
```

(e) Write the `main` function to first generate one rectangle, then repeatedly generates the second rectangle until both rectangles overlap. Print out the two overlapping rectangles. The output should be as follows:

```
Overlapping Rectangles:
Rectangle (0.141122,0.312640) --> (0.205298,0.776788)
Rectangle (0.054034,0.221907) --> (0.532487,0.635151)
```

2. An appointment book is used to schedule appointments and keep track of appointment clashes. An appointment consists of a single word description of the appointment, the start time and a duration. Each appointment starts and ends exactly on the hour. The structure definition of `Appointment` is given below.

```
typedef struct {
   char desc[40] ;   /* description not exceeding 40 characters */
   int start;        /* start time ranging from hour 0 to hour 23 */
   int duration;     /* duration in number of hours */
} Appointment;
```

An array `schedule` is declared to store the appointments for one day and is initialized to be "empty", i.e. all data members of structure elements in the array are nullified.

```
Appointment sched[24] = {{"",0,0}};
```

(a) Write a function `addAppointment` that takes in an existing schedule `sched` and an appointment `appt`. You may assume that appointment `appt` does not clash with existing appointments in the schedule `sched`.

(b) Write a function `isClash` that takes in a schedule `sched` and an appointment `appt`, and determines if there are any existing appointments in `sched` that clashes with `appt`. As an example, suppose there is only one existing appointment for "lecture" at 2pm for a duration of 2 hours in the schedule. Appointments that *clashes* with the above schedule includes

- "Coffee" at 2pm for a duration of 1 hour.
- "Movie" at 12nn for a duration of 3 hours.
- "Nap" at 3pm for a duration of 8 hours.

(c) Write a `main` function to read appointments from the user and prints out all appointments that do not clash in chronological order. A sample input is given below.

```
14 2 lecture
12 1 tutorial
12 2 lunch
-1
```

Each line is an appointment comprising the start time, the duration and the description of the appointment. The last value -1 is a sentinel value. As input appointments may possibly clash, they are included into the schedule on a first-come-first-serve basis. In the example above , "lecture" and "tutorial" is included, but not "lunch" as it clashes with "tutorial". Write a function `printSchedule` that takes in a schedule (`sched`) and prints the schedule in chronological order as shown below.

```
12 1 tutorial
14 2 lecture
```

3. The English Premier League is arguably the most popular football league in the world. You have been drafted in by the Football Association to write a simple program that reads match results and prints the resulting league table. As an example, the following input results in the league table below.

```
Chelsea vs Arsenal, 2-1
Man Utd vs Liverpool, 3-0
Tottenham vs Portsmouth, 2-0
Aston Villa vs Sunderland, 0-1
Middlesbrough vs Derby, 1-0
Blackburn vs Wigan, 3-1
Newcastle vs Fulham, 2-0
Bolton vs Man City, 0-0
Reading vs Birmingham, 2-1
Everton vs West Ham, 1-1
Man Utd vs Bolton, 2-0
Tottenham vs Chelsea, 4-4
Birmingham vs Newcastle, 1-1
Man City vs Tottenham, 2-1
Fulham vs Everton, 1-0
Wigan vs Bolton, 1-0
Derby vs Man Utd, 0-1
Sunderland vs Chelsea, 0-1
Liverpool vs Reading, 2-1
West Ham vs Blackburn, 2-1
Portsmouth vs Aston Villa, 2-0
Arsenal vs Middlesbrough, 1-1
```

	P	W	D	L	F	A	Pts
Man Utd	3	3	0	0	6	0	9
Chelsea	3	2	1	0	7	5	7
Newcastle	2	1	1	0	3	1	4
Tottenham	3	1	1	1	7	6	4
West Ham	2	1	1	0	3	2	4
Man City	2	1	1	0	2	1	4
Middlesbrough	2	1	1	0	2	1	4
Blackburn	2	1	0	1	4	3	3
Reading	2	1	0	1	3	3	3
Portsmouth	2	1	0	1	2	2	3
Sunderland	2	1	0	1	1	1	3
Wigan	2	1	0	1	2	3	3
Fulham	2	1	0	1	1	2	3
Liverpool	2	1	0	1	2	4	3
Arsenal	2	0	1	1	2	3	1
Birmingham	2	0	1	1	2	3	1
Everton	2	0	1	1	1	2	1
Bolton	3	0	1	2	0	3	1
Derby	2	0	0	2	0	2	0
Aston Villa	2	0	0	2	0	3	0

In the league table, each team name is followed by the number of games **played** (P), the number of **wins** (W), the number of **draws** (D), the number of **losses** (L), the number of goals scored **for** or by the team (F), the number of goals scored **against** the team (A), and the total number of **points** (Pts).

There are a maximum of 20 teams in the league. Team names have a maximum of 20 characters. Team names consist of only letters, numbers, and spaces, and may contain one or more words. The first letter for each word in a team name is always capitalized. Each team may score between zero to 999 goals inclusive per match.

(a) An example of a line of input is:

```
Team A vs Team B, 3-1
```

We wish to separate the line into two parts. Part one should contain the string "Team A vs Team B" (without the quotes), and part two should contain the string "3–1" (without the quotes). Define the `split` function that splits the line of text into two parts.

```
void split(char *s, char *s1, char *s2);
```

- `s` contains the line of text to be split. You may modify this string.
- `s1` is to contain part one of the text.
- `s2` is to contain part two of the text.

(b) Now we want to further split part two of the text which contains the results. An example of this string is "3–1" (without the quotes), which says that the first team scored three goals and the second team scored one goal. We want to extract out the number of goals scored as integers. Define the `getScore` function.

```
void getScore(char *s, int *g1, int *g2);
```

- `s` contains part two of the line of text to be split. You may modify this string.
- `g1` is a pointer to an integer that contains the number of goals scored by the first team.
- `g2` is a pointer to an integer that contains the number of goals scored by the second team.

(c) Now we want to further split part one of the text which contains the team names. An example of this string is "Team A vs Team B" (without the quotes), where the name of the first team is "Team A" and the name of the second team is "Team B" (all without the quotes). The string "vs" means versus. We want to extract out the team names from this string. Define the `getTeamNames` function.

```
void getTeamNames(char *s, char *t1, char *t2);
```

- `s` contains part one of the line of text to be split. You may modify this string.
- `t1` is to contain the name of the first team.
- `t2` is to contain the name of the second team.

(d) Each team consists of its team name and seven other attributes as shown in the league table. Define a suitable C structure called "`Team`".

(e) In the `main` function, we will have an array of `Team` structures that will eventually contain all the teams. As we read the results from the text file, we search the array to see if this team already exists in the array. Use a simple linear search on an unsorted array. If this team is found in the array, simply return the index of the team in the array. If this team is not found in the array, add the new team. Initialize all the new team's attributes (except the team name) to zero. The function then returns the index of the new team in the array.

Define the following `search` function.

```
int search(Team teams[], int *nmTeams, char *teamName);
```

- `teams` references the array of `Team` structures to be searched and modified if necessary.
- `numTeams` is a pointer to an integer that contains the number of teams. If the team name is not found in the array, this is increased by one.
- `teamName` contains the team name to search.

(f) Define the following `update` function that updates the attributes of two teams based on the number of goals scored by each team in a single match.

```
void update(Team *t1, Team *t2, int g1, int g2);
```

- `t1` is a pointer to a structure containing the first team.
- `t2` is a pointer to a structure containing the second team.
- `g1` is the number of goals scored by the first team.
- `g2` is the number of goals scored by the second team.

Suppose the result of "Team A vs Team B" is "3–1", which means Team A scored three goals and Team B scored one goal. The team which scores more goals wins. The winning team gets three points and the losing team gets zero points. If both teams score the same number of goals, it is a draw result and both teams get one point each. Update the relevant attributes for each team. Remember to also update the number of games played and the (F) and (A) attributes.

(g) Define the `main` function to perform the following tasks:

(1) Read the input line by line. Each line consists of the results of a single match. You may assume each line in the text file consists of at most 100 characters.

(2) For each line in the file:

(i) Extract the team names and number of goals scored per team.

(ii) Search for both teams in the team array. If it does not exist, add it.

(iii) Update the attributes for both teams in the team array.

Remember to declare all your necessary variables, for example the array of `Team` structures, the string to store the line of text, etc.

Adapt the following `printTable` function to print the league table. It is not necessary to sort the league table at this time.

```
void printTable(Team teams[], int n) {
   int i;

   printf("%20s P   W   D   L   F   A   Pts\n", "");
   for (i = 0; i < n; i++) {
      printf("%-20s %-3d %-3d %-3d %-3d %-3d %-3d %-3d\n",
         "Team", 0, 0, 0, 0, 0, 0, 0);
   }

   return;
}
```

(h) *Extra question not in the exam...*

Define the `sort` function that sorts the league table in order of the best team first based on the points. If two or more teams are equal on points, the winner is then determined by goal difference (F)−(A) and then, if necessary, by the number of goals scored. If teams are equal on points, goal difference and the same number of goals scored, then sort the teams in lexicographical (dictionary) order.

```
void sort(Team teams[], int numTeams);
```

- `teams` references the array of `Team` structures to be sorted.
- `numTeams` contains the number of teams.

Now print out the league table after sorting and check against the one given at the start of the question.