# Style rules for writing C

These are the standard rules used to evaluate the code submitted with coursework.

1. All program files should start with a multi-line comment. The first and last line of the comment padded with asterisks to 72 characters. The second line of the comment should be the current file name. The next line, or two, should describe the purpose of the code in the file. The following line gives your name or id no, followed by a line giving the date the code was last modified or version number.

```
/**********************************************************************
 * w11p02agentDB.c
 * maintains list of agents in text mode
 * 882266
 * Version a
 * ********************************************************************/
```

2. For blocks of code, use detached curly brackets with 4 space indents. That is, the open curly bracket goes immediately under the first character of the block header line. The close curly bracket should be placed on a line by itself vertically aligned with the open curly bracket and all statements between the curly brackets should be indented by 4 spaces.

```
/* Empty the keyboard buffer */
void emptyBuffer(void)
{
    while(getchar() != '\n')
    {
        ;
    }
}
```

3. All statements between the curly brackets should be indented by 4 spaces (not tab characters).
4. Put one space around all operators (arithmetic, relational, assignment) and curly brackets.

```
    int century = year / 100;
    year %= 100;
    return dayOfWeek ((26 * month - 2) / 10 + _day + year + year / 4
                     + century / 4 + 5 * century) % 7;
```

5. Have one space after a comma but not before.

```c
char month[] = { "", "January", "February", "March", "April", "May",
                 "June", "July", "August", "September", "October",
                 "November", "December" };
```

6. Don’t have spaces around round brackets (parenthesis).

```c
if((monthNo == 2) && isLeapYear())
{
    return 29;
}
```

7. Attach pointer symbols to their variable name if given; otherwise attach the symbol to the data type.
8. With conditional statements (if, for, while) always use curly brackets for the statement body, even for one line blocks.

```c
if((monthNo == 2) && isLeapYear())
{
    return 29;
}
```

9. Break all lines that exceed 72 characters. When doing this, break after a comma, a closed round bracket ‘)’, a semicolon or space. The continuation of a split lines should have a further indent, (not always of 4 spaces).

```c
while((scanf("%d/%d/%dERROR - invalid date.\n", &day, &month, &year)
            != 3)
          || year < 1800 || year > 2099
          || month < 1 || month > 12
          || day < 1 || day > 31)
{
    printf("ERROR - invalid date.\nEnter date (dd/mm/yyyy): ");
}
```

10. Do not have blank lines within a function. Functions should be short and single purpose.
11. Variable and function names should be meaningful and written in lower case. With multiple word names, capitalise the first letter of the second and later words. Do not use underscores to separate words.

12. Constant names should be written in uppercase and use underscores to separate words. Use a constant instead of a magic number, for all numbers other than 0, 1, -1 or 2.
13. Global variables should never be used but global constants and definitions are fine.
14. Put a comment on the line above the header line of a function definition.
15. Switch statement case labels should be indented and the associated code statements further indented on the following lines.

```
/*************************************************************************
 * w11p02agentDB.c
 * maintains list of agents in text mode
 * 882266
 * Version a
 * ********************************************************************/

#include <stdio.h>

struct personnel
{
    char name[40];
    int agnumb;
    float height;
};

int addAgent(struct personnel agent[], int noAgt);
void listall(struct personnel agent[], int noAgt);
void wfile(struct personnel agent[], int noAgt);
int rfile(struct personnel agent[]);
void emptyBuffer(void);

int main(void)
{
    struct personnel agent[50];
    int noAgt = 0;
    int ch = 'X';
    printf("\tAgents Database\n"
           "\t===============\n");
    while (ch != 'q')
    {
        printf("\n'e'\tenter new agent"
               "\n'l'\tlist all agents"
               "\n'w'\twrite file"
               "\n'r'\tread file"
               "\n'q'\tquit"
               "\n\nSelect: ");
        ch = getchar();
        emptyBuffer();
        switch (ch)
        {
            case 'e' :
                noAgt = addAgent(agent, noAgt);
                break;
            case 'l' :
                listall(agent, noAgt);
                break;
            case 'w' :
                wfile(agent, noAgt);
                break;
            case 'r' :
                noAgt = rfile(agent);
                break;
            case 'q' :
                break;
            default :
                puts("\nEnter only selections listed");
        }
    }
    return 0;
}
/* puts a new agent in the database */
int addAgent(struct personnel agent[], int noAgt)
{
    printf("\nRecord %d.\nEnter name: ", noAgt + 1);
    scanf("%39[^\n]", agent[noAgt].name);
    printf("Enter agent number (3 digits): ");
    scanf("%d", &agent[noAgt].agnumb);
    printf("Enter height in inches: ");
    scanf("%f", &agent[noAgt].height);
    emptyBuffer();
    return ++noAgt;
}
/* lists all agents and data */
void listall(struct personnel agent[], int noAgt)
{
    int i;
    if(noAgt < 1)
    {
        printf("\nEmpty list.\n");
        return;
    }
    for(i = 0; i < noAgt; i++)
    {
        printf("\nRecord number %d\n", i + 1);
        printf("\tName:         %s\n", agent[i].name);
        printf("\tAgent number: %03d\n", agent[i].agnumb);
        printf("\tHeight:       %4.2f\n", agent[i].height);
    }
}
/* writes agents to file */
void wfile(struct personnel agent[], int noAgt)
{
    FILE *fptr;
    int i;
    struct personnel rec;
    if(noAgt < 1)
    {
        printf("\nCan't write empty list.\n");
        return;
    }
    if ((fptr = fopen("agents.txt", "w")) == NULL)
    {
        printf("\nCan't open file agents.txt\n");
        return;
    }
    for(i = 0; i < noAgt; i++)
    {
        rec = agent[i];
        fprintf(fptr, "%s:%d %f\n", rec.name, rec.agnumb, rec.height);
    }
    fclose(fptr);
    printf("\nFile of %d records written.\n", noAgt);
}
/* reads records from file into array */
int rfile(struct personnel agent[])
{
    FILE *fptr;
    int noAgt = 0;
    struct personnel rec;
    if((fptr = fopen("agents.txt", "r")) == NULL)
    {
        printf("\nCan't open file agents.txt\n");
        return noAgt;
    }
    while(fscanf(fptr, "%39[^:]: %d %f\n", rec.name,
        &rec.agnumb, &rec.height) == 3)
    {
        agent[noAgt++] = rec;
    }
    fclose(fptr);
    printf("\nFile read.  Total agents is now %d.\n", noAgt);
    return noAgt;
}
/* Empty the keyboard buffer */
void emptyBuffer(void)
{
    while(getchar() != '\n')
    {
        ;
    }
}
```