Örebro University
School of Science and Technology
Thomas Padron-McCarthy (thomas.padron-mccarthy@oru.se)

Exam for

Programmering grundkurs and Programmering C

for D1 and others, including the distance learning course

Thursday, August 22, 2013

Valid as exam for:
DT1029 Datateknik A, Programmering grundkurs, provkod 0100
DT1030 Datateknik A, Tillämpad datavetenskap, provkod 0410
DT1006 Datateknik A, Programmering C, distans, provkod 0100
DT1016 Datateknik A, Programmering grundkurs, provkod 0100
DT1007 Datateknik A, Tillämpad datavetenskap, provkod 0410

(**English version**. There is also a Swedish version of this exam.)

Aids:	No aids.
Score requirements:	Maximum score is 40. To pass (the grade 3 or G) 20 points are needed.
Results and solutions:	Notified by email or on the course home page, http://basen.oru.se/kurser/c/2012-2013-p2/, no later than Thursday, September 12, 2013.
Return of exams:	After the results have been announced, exams can be retrieved from the university's central exam retrieval office.
Responsible teacher and on-call:	Thomas Padron-McCarthy, telephone 070-73 47 013.

- Write clearly. Solutions that can't be read can't give any points. Vague and ambiguous wording will be misinterpreted.
- Write the personal exam code on each sheet submitted. Do *not* write your name or person number on the sheets.
- Write only on one side of the paper. Do not use red writing.
- Assumptions in addition to those in the problems must be stated.
- You can write explanations of your reasoning. Even an answer that is wrong may give credit, if there is an explanation that shows that the main ideas were right.

LYCKA TILL!

Precedence and associativity of operators in C

The main operators:

Priority	Category	Operator	Associativity
Highest	Unary postfix operators	(), [], ->, ., ++, --	left
	Unary prefix operators	!, ++, --, +, -, *, &, sizeof, (*type*)	right
	Multiplication etc	*, /, %	left
	Addition etc	+, -	left
	Comparisons	<, <=, >=, >	left
	Equality comparisons	==, !=	left
	Logical AND	&&	left
	Logical OR	||	left
Lowest	Assignment	=, +=, -=, *=, /=, %=	right

Some useful library functions

stdlib.h

```
int rand(void);
void srand(unsigned int seed);
void *malloc(size_t size);
void *realloc(void *ptr, size_t size);
void free(void *ptr);
void exit(int status);
void qsort(void *base, size_t nmemb, size_t size,
           int(*compar)(const void *, const void *));
```

stdio.h

```
FILE *fopen(const char *path, const char *mode);
int fclose(FILE *stream);
int getc(FILE *stream);
int getchar(void);
int ungetc(int c, FILE *stream);
char *fgets(char *s, int size, FILE *stream);
char *gets(char *s);
int putc(int c, FILE *stream);
int printf(const char *format, ...);
int fprintf(FILE *stream, const char *format, ...);
int sprintf(char *str, const char *format, ...);
int snprintf(char *str, size_t size, const char *format, ...);
int scanf(const char *format, ...);
int fscanf(FILE *stream, const char *format, ...);
int sscanf(const char *str, const char *format, ...);
size_t fread(void *ptr, size_t size, size_t nmemb, FILE *stream);
size_t fwrite(const void *ptr, size_t size, size_t nmemb, FILE *stream);
```

string.h

```
size_t strlen(const char *s);
char *strcpy(char *dest, const char *src);
char *strncpy(char *dest, const char *src, size_t n);
int strcmp(const char *s1, const char *s2);
int strncmp(const char *s1, const char *s2, size_t n);
char *strcat(char *dest, const char *src);
char *strncat(char *dest, const char *src, size_t n);
char *strstr(const char *haystack, const char *needle);
void *memmove(void *dest, const void *src, size_t n);
```

ctype.h

```
int isalnum(int c);
int isalpha(int c);
int isblank(int c);
int isdigit(int c);
int islower(int c);
int isprint(int c);
int ispunct(int c);
int isspace(int c);
int isupper(int c);
```

Task 1 (1 p)

What are the values of the following expressions in C?

a) 4 * 3 − 2 + 1

b) 4 − 3*2 + 1

c) 4 / 3 − 4 / 2

Task 2 (3 p)

What is printed when the following C program is executed?

```
#include <stdio.h>

int g(int x, int y) {
    int i, a, b, c;
    a = 17;
    b = 2;
    c = 38;
    for (i = x; i < y; i++)
        printf("*");
    return x + y;
}

int main(void) {
    int a;
    int b, c;
    for (a = 1; a < 3; ++a) {
        b = a + 1;
        c = g(a, b);
        printf("a = %d, b = %d, c = %d\n", a, b, c);
    }
    return 0;
}
```

Task 3 (3 p)

Write a complete C program (with **#include** and all else that is needed) that repeatedly reads three integers, and prints their average value (with decimals). The program should read three integers until the user enters three numbers that are all equal to zero.

For this and all other tasks on the exam:
Typically, error handling is a large part of a program. For instance, what should happen if the user writes **Donald** when she really should enter a number? Here, however, no error handling is needed, unless explicitly required in the task.

For this and all other tasks on the exam:
One can ignore details that are only needed when developing console application in Visual Studio, such as weird character codes for **ÅÄÖ**, and that the window with program disappears when the program ends.

Scenario

Here is a record type, **struct Triangle**, which we will use to store data about triangles:

```
struct Triangle {
    double a, b, c;
};
```

a, **b** and **c** are the lengths of the sides of the triangle. The sides can be stored in any order, so there is for example no guarantee that the triangle's longest side is anyone specific of **a**, **b** or **c**. All lengths must be greater than zero.

Task 4 (1 p)

Define a variable of type **struct Triangle** and initialize it with data for the triangle with sides **3**, **4** and **3.5**.

Task 5 (2 p)

Write a function called **show_triangle**, which shows the data of a triangle on the screen. It should print the triangle data, with appropriate labels, to the standard output. You may select whether you want the function header to look like this:

```
void show_triangle(struct Triangle t)
```

or like this:

```
void show_triangle(struct Triangle *tp)
```

Task 6 (3 p)

Write a function called **read_triangle**, which reads the data of a triangle. The function should print the appropriate prompts on the standard output, and read data from the standard input (which is normally connected to the keyboard). You may select whether you want the function header to look like this:

```
struct Triangle read_triangle(void)
```

or like this:

```
void read_triangle(struct Triangle* tp)
```

Task 7 (5 p)

We want to create the function **min**, which returns the length of the shortest side of a triangle, and the function **max**, which returns the length of the longest side of a triangle. The functions should take a triangle (**struct Triangle**), or a pointer to it, as a parameter.

Write the functions **min** and **max**.

Task 8 (2 p)

Write the function **scale**, which re-scales a triangle by a faktor, which should be specified as an argument. If you for example send the factor **2**, all three sides of the triangle should become twice as long.

Task 9 (3 p)

Write the function **area**, which returns the area of a triangle. Calculate the area **A** using the following formula:

$$A=\frac{a}{2}\sqrt{b^2-\left(\frac{a^2+b^2-c^2}{2\mathrm{a}}\right)^2}$$

Task 10 (4 p)

Write a **main** function that has a local variable of the triangle type, and that does the following:

- read data into the variable using the function **read_triangle**
- prints the triangle using **show_triangle**
- calculates tha area of the triangle using **area**, and prints it
- asks for and lets the user input a scaling factor
- applies that scaling factor on the triangle using the function **scale**
- prints the re-scaled triangle using **show_triangle**
- calculates tha area of the re-scaled triangle using **area**, and prints it

For this and all other tasks on the exam:
If you need to use something from a previous task or sub-task, such as a data type or a function that was written in the previous task, you do not have to write the same code again. You may also do the task even if you have not done that previous task.

Task 11 (3 p)

Write the function **max_area**, which returns the area of the largest triangle in an array. It should take an array of triangles as argument, and an integer that indicates the number of triangles in the array, and it will then return the area of the triangle in the array that has the greatest area. Use the function **area** to calculate the areas.

Task 12 (3 p)

We want to test the function **max_area**. Write a **main** function that calls **max_area** with suitable data, and verifies that it gives the expected response. If the function gives an incorrect answer, a clear error message should be printed. Otherwise there is no need to print anything.

For realistic testing we would need more than just one call, but here we only do one. Thus there is only a single test case. Try to make that test case as good as possible!

Task 13 (3 p)

Write a C program that reads data about one (yes, just one) triangle from the user, by using the function **read_triangle**, and saves it to a file. Decide for yourself whether it should be a text file or binary file. Write in your solution which you chose!

If the file can not be opened, an error message should be printed, and the program should be terminated.

Do not forget to specify if you chose a text file or a binary file!

Task 14 (3 p)

Write a C program that reads the file from the task above, and prints the stored triangle by using the function **show_triangle**.

If the file can not be opened, an error message should be printed, and the program should be terminated.

Task 15 (1 p)

What does the following C program write? Explain your answer!

```
#include <stdio.h>

int main(void) {
    double x = 0.2, y = 6.0, z = 1.2;
    printf("%f\n", x * y);
    printf("%f\n", z);
    if (x * y == z)
        printf("Lika, förstås.\n");
    else
        printf("Va? Inte lika!\n");
    return 0;
}
```