DA4002 - Algorithms, Data Structures and Problem Solving

-----------------------------------------------------------------------------------------------------------------

**Lab1 - Group 1**

-----------------------------------------------------------------------------------------------------------------

Eric Heindl

Esteban Rueda

1. Modify the program so that it prints some other message instead, for example “Bye

bye…”.

printf("This is a test from Eric & Esteban !\n");

2. What happens when you erase the statement “return 0;” from the source code and

then recompile?

Did you expect this?

When we executed the program without “return 0” it didn’t give us an error which we didnt expect.

3. One of the most useful options for gcc is to enable warning messages.

This is done by adding “-Wall” to the command.

Compiler warnings are a very good way of catching potential bugs very early, before

even running the program.

To try this out, you have to edit the Makefile so that it contains the following version

of the gcc specification:

gcc -Wall hello-world.c -o hello-world

After you saved the new Makefile and rebuild the project, how does the compiler

output change?

What does the new message mean?

Think about why it is a good idea to always pass -Wall to the compiler and edit the

source code until all warnings are gone.

Now, put the “return 0;” statement back before continuing…

We didn’t get any error message with the -Wall parameter in the makefile when compiling.

4. What happens if you erase the semicolon (the “;” character) after the printf() and

rebuild the project?

Try to understand the compiler messages, and put the semicolon back before

continuing with the next task…

5. What happens when you change “main” to something else, for example “blah“?

Can you still compile the project?

Try to explain the message you get.

Compiler message says we should put a semicolon before return.

gcc -Wall hello-world.c -o hello-world

hello-world.c: In function 'main':

hello-world.c:68:3: error: expected ';' before 'return'

return 0;

^~~~~~

The program didnt find the Main function so it returned 1 (exit).

c:/mingw/bin/../lib/gcc/mingw32/6.3.0/../../../libmingw32.a(main.o):(.text.startup+0xa0): undefined reference to `WinMain@16'

collect2.exe: error: ld returned 1 exit status

1. Try to predict what the builtin-types application prints when you run it.

Then create a new project (following the steps for execise 1.1 but with a different

project name) to build and run it.

Does it print what you expected?

We expected the program to print the values and the sizes of the variables.

It printed pretty much what we expected.

%zu does not work in our laptops so we changed it to its equivalent %lu.

FORMATTED OUTPUT

int:

%d format 42

double:

%f format 17.900000

%e format 1.790000e+001

char:

%c format Q

%d format 81

unsigned long long:

%llu format 1234567890123456789

MEMORY SIZES

int: 4 bytes

double: 8 bytes

char: 1 byte

long long: 8 bytes

SIGNED and UNSIGNED

the size does not change:

4 is 4

but they are interpreted differently:

signed: -99

unsigned: 4294967197

2. (Optional)

The variable cc gets initialized using a single-character constant 'Q'.

What happens when you initialize it with an integer constant instead?

Try it out by setting cc = 81.

Read the manual page about ASCII codes (also available in the terminal by giving the

command “man ascii“) and find out what number you have to use in order to get the

letter 'z'.

Check your answer by modifying and running the builtin-types application.

The result didn’t change when using 81 because the equivalent code for Q is 81 in ASCII.

We used number 122 which is the equivalent in ASCII for letter ‘z’.

3. (Optional)

What is the largest value that can be stored in a variable of type int, assuming that it

is represented with 4 bytes (32 bits)?

What happens when you try to initialize an int variable with a value that is larger than

that?

Try it out by modifying the builtin-types.c source code and recompiling.

How about using an unsigned int instead, does that influence the largest value that

can be stored?

How about the smallest value for int and unsigned int?

-2,147,483,648 to 2,147,483,647 is the range for signed int and 0 to 4,294,967,295 is the range for unsigned int.

.\builtin-types.c:37:8: warning: large integer implicitly truncated to unsigned type [-Woverflow]

ee = 4294967296;

^

Largest value for unsigned int: 4,294,967,295

Smallest value for int: -2,147,483,648

Smallest value for unsigned int: 0

1. Compile and run the type-conversion application.

The result is probably not what you expected. Try to explain why before reading on…

Answer: in an assignment the expression to the right is evaluated first, in this case a

division.

In the division both arguments are integers so / is interpreted as integer division,

which truncates the remainder of the division.

This is probably not what was intended!

Only after the integer division is the result converted into the type of the variable

being assigned, in this case a double-precision floating point.

2. It is possible for the programmer to force type conversion by preceding the expression

to be converted by the type name in parenthesis, such as “(double)“.

Try it out! Experiment with the program so that the result stored in the variable q is

0.6 instead of 0.0.

x/y is 0.000000 For the integer division.

x/y is 0.600000 For the double division.

1. Assume we have three integer variables called a, b, and c, with initial values b=5 and

c=8.

What is the value of a, b, and c after each line of the following program fragment?

First find the answer by hand, then write a program to verify your reply.

2. a = b++ + c++;

3. a = b++ + ++c;

4. a = ++b + c++;

a = ++b + ++c;

5. What is the result of the following expression?

Write a program to verify your answer.

(Hint: place the expression into an if-else construct which prints different messages

depending on the result of the expression.)

1 && 0 || 1

13 - 14 - 14 - 15

if(1 && 0 || 1)

{

printf ("OK\n");

}else

{

printf("NOT OK\n");

}

It always prints “OK”.

**Addition Table:**

#include <stdio.h>

int main(int argc, char \*\* argv)

{

printf(" Addition Table\n ");

for(int i=0; i<10; i++)

{

printf("| %d ",i);

}

printf("| \n -----------------------------------------\n");

for(int i=0; i< 10; ++i)

{

printf(" %d | ",i);

for(int j=0;j<10;j++)

{

printf("%d ",i+j);

if((i+j)<10){printf(" ");}

}

printf("\n");

}

return 0;

}

**Multiplication Table:**

#include <stdio.h>

int main(int argc, char \*\* argv)

{

printf(" Multiplication Table\n ");

for(int i=0; i<10; i++)

{

printf("| %d ",i);

}

printf("| \n -----------------------------------------\n");

for(int i=0; i< 10; ++i)

{

printf(" %d | ",i);

for(int j=0;j<10;j++)

{

printf("%d ",i\*j); //change + to \*

if((i\*j)<10){printf(" ");}

}

printf("\n");

}

return 0;

}

2.Write a program that lists all pairs of positive integers (a,b) satisfying the following two

conditions:

o a < b < 1000

o “(a\*a + b\*b + 1) / (a\*b)” is an integer

#include <stdio.h>

int main (int argc, char \*\* argv)

{

double a, b = 0;

for ( a = 0; a < 1000; a++)

{

for (b = 0; b < 1000; b++)

{

if (a < b)

{

float c = (a\*a + b\*b + 1)/(a\*b);

if(c-(int)c == 0)

{

printf("%i %i \n", (int)a, (int)b);

}

}

}

}

return 0;

}

**Fibonacci Series:**

#include <stdio.h>

int main()

{

int f1 = 1, f2 = 1, temp;

printf("Fibonacci Series: ");

for (int i = 1; i <= 20; ++i)

{

printf("%d, ", f1);

temp = f1+f2;

f1 = f2;

f2 = temp;

}

return 0;

}