**Final work in operating systems:**

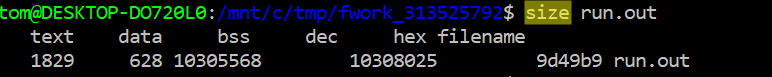
**First question:**

**Section 0:**

**Size:**

In order to see what size each area given in memory for a particular program we will be use size command.

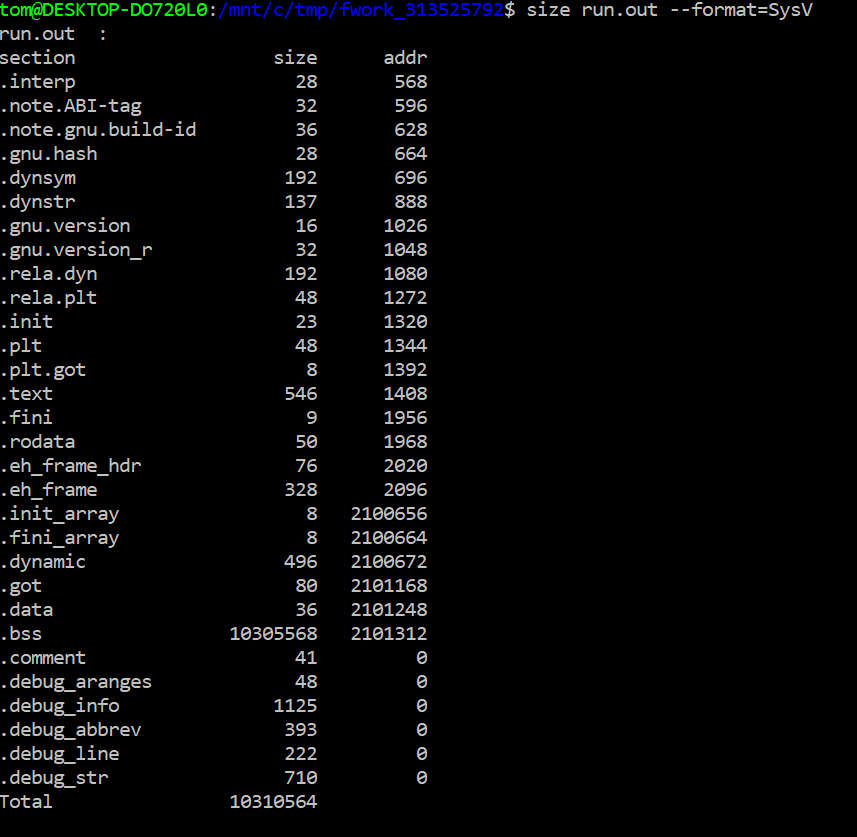
For example:



This command has several options that can be used by adding flags to the command.

For example:

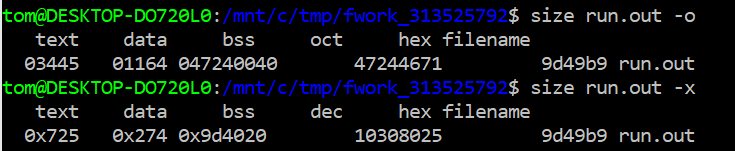
* Change the display format by –format=SysV



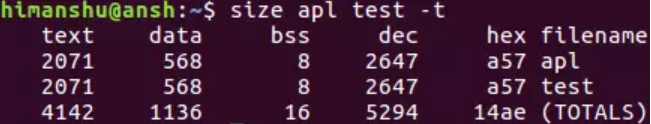
* By default, the section sizes are displayed in decimal base.

However, we can present this information in an octal or hexadecimal manner.

To do this, use the -o or -x options.



* You can see the size of the areas of some files, as follows: size -t [file1] [file2]…



There are other nice options you can just look at man or online and read about them.

**NM:**

The nm command basically displays information related to the symbols in an object file.  If no object files are listed as arguments, nm assumes the file *a.out*.

This command has several options that can be used by adding flags to the command.

For example:

* Sort symbols numerically by their addresses, rather than alphabetically by their names.

What do the letters say if it appears next to the symbol after we executed the command:

* **A:** The symbol's value is absolute and will not be changed by further linking.
* **B\b:** The symbol is in the uninitialized data section (known as BSS).
* **D\d:** The symbol is in the initialized data section.
* **T\t:** The symbol is in the text (code) section.
* **U:** The symbol is undefined.

\* If it's a lowercase letter it means the symbol is local

There are other nice options you can just look at man or online and read about them.

**Objdump:**

displays information from object files. Displays the information in the way we request according to the options.

This command has several options that can be used by adding flags to the command.

For example:

* **d:** Display the assembler mnemonics for the machine instructions from *objfile*. This option only disassembles those sections which are expected to contain instructions.
* **S:** Display source code intermixed with disassembly, if possible.

There are other nice options you can just look at man or online and read about them.

**Section 1:**



**Section 2:**

1. Question: Where is allocated?

Line: line 5, char globBuf[65536];

Answer: In the data segment - **BSS** area, global and static variables are stored in this area that are initialized at 0 or not initialized at all in the program code.

1. Question: Where is allocated?

Line: line 6, int primes[] = { 2, 3, 5, 7 };

Answer: In the **Data** area, in this area global and static variables that are initialized in the program code (other than those that are initialized to 0) are kept in that area, as well as strings that are defined in the program code and cannot be changed.



1. Question: Where is allocated?

Line: line 9, square(int x)

Answer: In the **text** area, which contains executable instructions.



1. Question: Where is allocated?

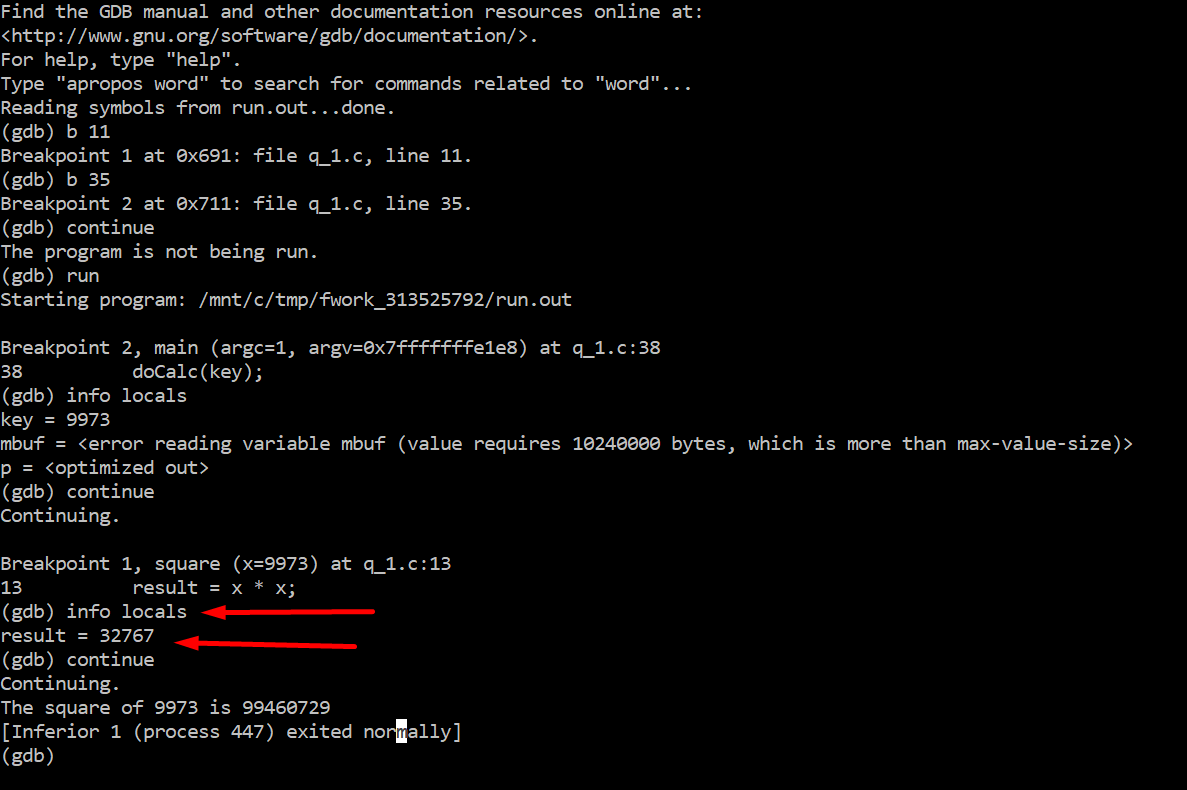
Line: line 11, int result;

Answer: In the **Stack** area. it is used to store all **local variables** and is used for **passing arguments** to the functions along with the return address of the instruction which is to be executed after the function call is over.

I compile as follows: gcc -g q\_1.c -o run.out

Then I execute the software with GDB as follows: gdb run.out

Before I execute, I put breakpoint in line 11 and after I ran and got to breakpoint I ran the command "info locals" which prints all the local variables.



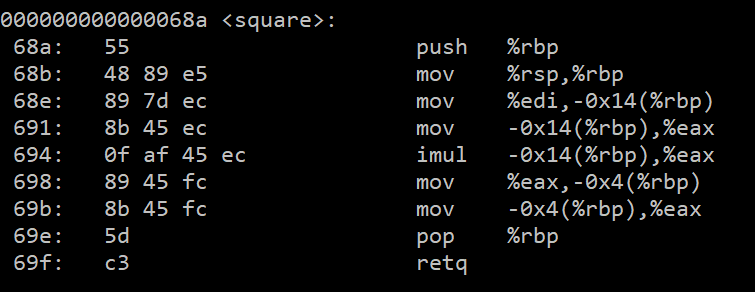
1. Question: How the return value is passed?

Line: line 14, return result;

Answer: The value returned by a register.

I execute: objdump -d run.out

It can be seen that after the multiplication of the variable has been calculated it is passed by a register called eax whose known as register that pass the returned value from a function.



1. Question: Where is allocated?

Line: line 18

Answer: In the **text** area, which contains executable instructions.



1. Question: Where is allocated?

Line: line 23, int t;

Answer: In the **Stack** area. it is used to store all **local variables** and is used for **passing arguments** to the functions along with the return address of the instruction which is to be executed after the function call is over.

Note that the variable t is inside block if so if it does not enter this block the variable t will not be created on the Stack and so it really happens because in the function main the value we send to the function doCalc is static int

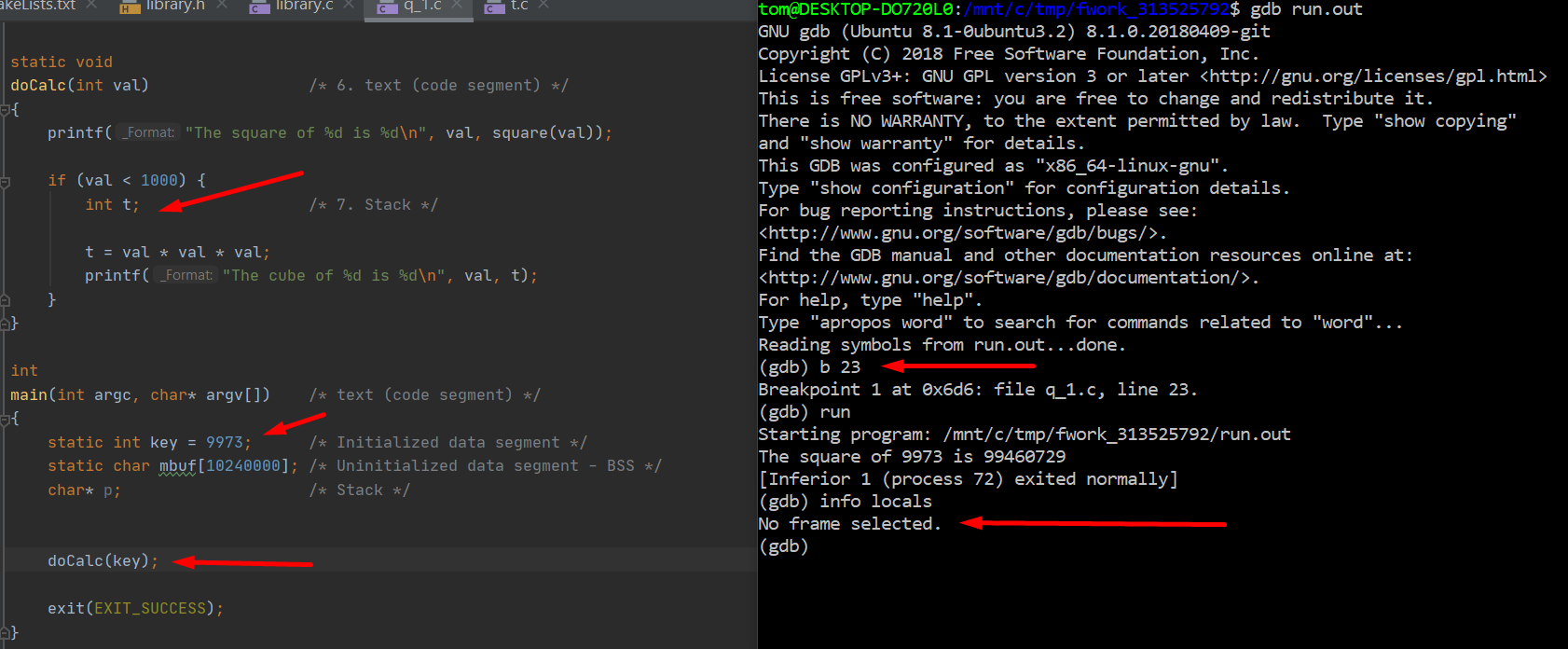
key = 9973 which is definitely not less than 1000 so t Not created so if I do the:

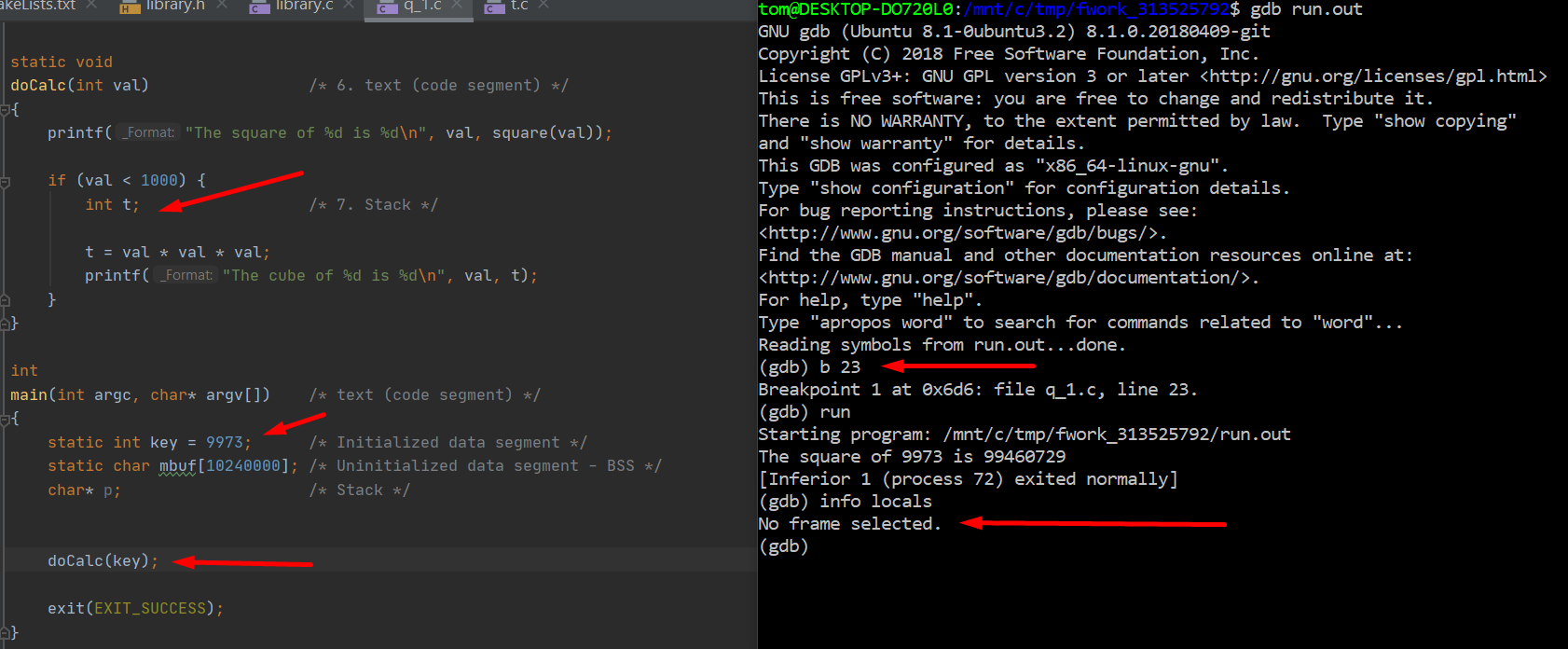
Compile as follows: gcc -g q\_1.c -o run.out

Then I execute the software with GDB as follows: gdb run.out

Before running I put breakpoint in line 23 and after I ran and got to the breakpoint I execute the command "info locals" which prints all the local variables but t was not created so it is not printed.

But if I change the value of key to less than 1000 then in variable t it will appear when I run the "info locals" command.

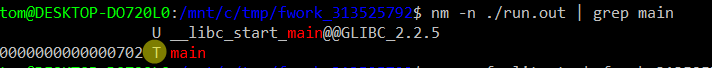




1. Question: Where is allocated?

Line: line 31, main(int argc, char\* argv[])

Answer: In the **text** area, which contains executable instructions.



1. Question: Where is allocated?

Line: line 33, static int key = 9973;

Answer: In the **Data** area, in this area global and static variables that are initialized in the program code (other than those that are initialized to 0) are kept in that area, as well as strings that are defined in the program code and cannot be changed.



1. Question: Where is allocated?

Line: line 34, static char mbuf[10240000];

Answer: In the data segment - **BSS** area, global and static variables are stored in this area that are initialized at 0 or not initialized at all in the program code.



1. Question: Where is allocated?

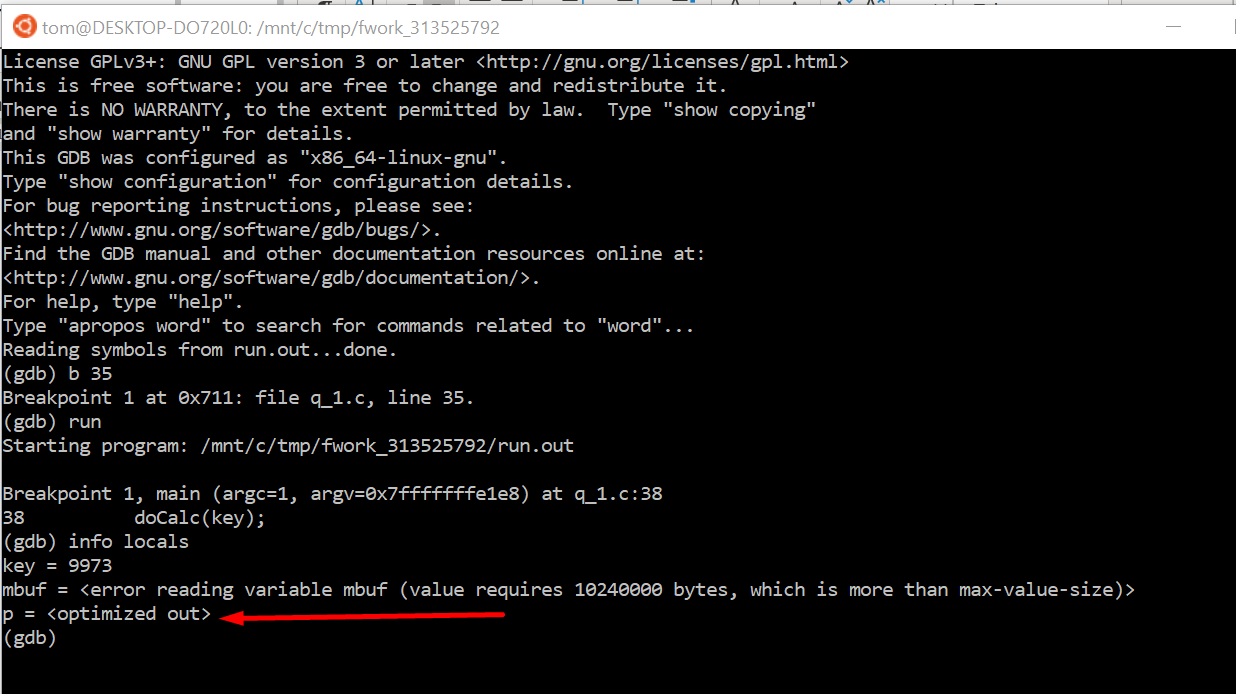
Line: line 35, char\* p;

Answer: In the **Stack** area. it is used to store all **local variables** and is used for **passing arguments** to the functions along with the return address of the instruction which is to be executed after the function call is over.

I compile as follows: gcc -g q\_1.c -o run.out

Then I execute the software with GDB as follows: gdb run.out

Before I execute, I put breakpoint in line 35 and after I ran and got to break point I ran the command "info locals" which prints all the local variables.



Sources for the question:

<https://sourceware.org/gdb/current/onlinedocs/gdb/Frame-Info.html#index-info-locals-435>

<https://stackoverflow.com/questions/6261392/printing-all-global-variables-local-variables>