Final work in operating systems:

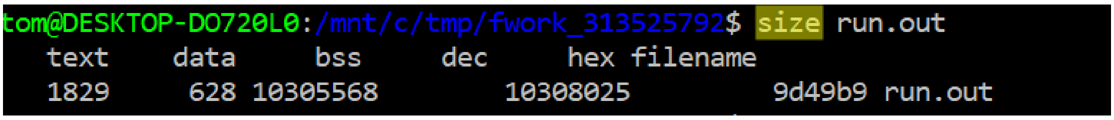
First question:

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:

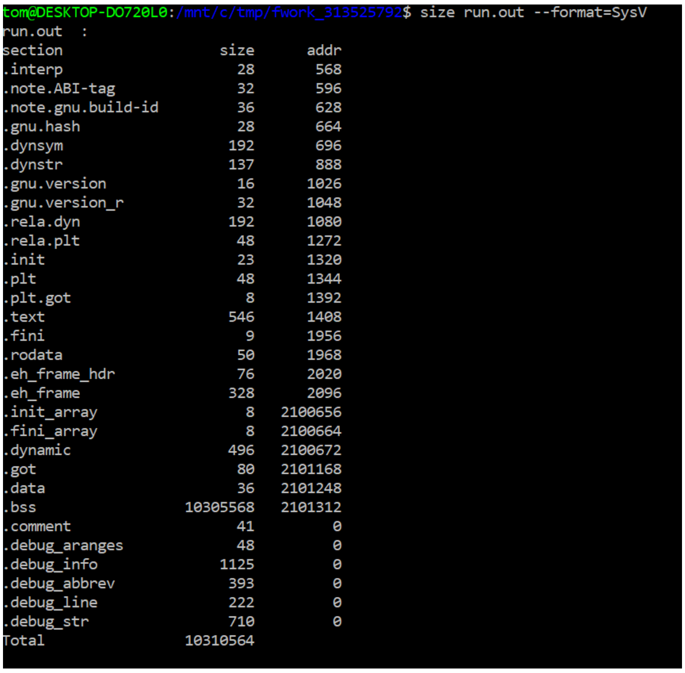


```
tom@DESKTOP-DO720L0:/mnt/c/tmp/fwork_313525792$ size run.out
   text    data     bss     dec     hex filename
   1829     628 10305568        10308025        9d49b9 run.out
```

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

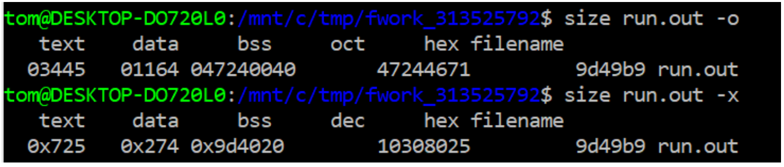
For example:

- Change the display format by –format=SysV



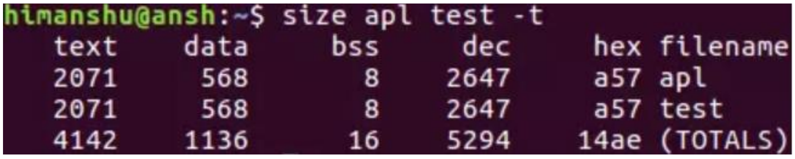
```
tom@DESKTOP-DO720L0:/mnt/c/tmp/fwork_313525792$ size run.out --format=SysV
run.out  :
section                 size      addr
.interp                   28       568
.note.ABI-tag             32       596
.note.gnu.build-id        36       628
.gnu.hash                 28       664
.dynsym                  192       696
.dynstr                  137       888
.gnu.version              16      1026
.gnu.version_r            32      1048
.rela.dyn                192      1080
.rela.plt                 48      1272
.init                     23      1320
.plt                      48      1344
.plt.got                   8      1392
.text                    546      1408
.fini                      9      1956
.rodata                   50      1968
.eh_frame_hdr             76      2020
.eh_frame                328      2096
.init_array                8   2100656
.fini_array                8   2100664
.dynamic                 496   2100672
.got                      80   2101168
.data                     36   2101248
.bss                10305568   2101312
.comment                  41         0
.debug_aranges            48         0
.debug_info             1125         0
.debug_abbrev            393         0
.debug_line              222         0
.debug_str               710         0
Total               10310564
```

- 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.



```
tom@DESKTOP-DO720L0:/mnt/c/tmp/fwork_313525792$ size run.out -o
   text    data     bss     oct     hex filename
  03445   01164 047240040       47244671        9d49b9 run.out
tom@DESKTOP-DO720L0:/mnt/c/tmp/fwork_313525792$ size run.out -x
   text    data     bss     dec     hex filename
  0x725   0x274 0x9d4020        10308025        9d49b9 run.out
```

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



```
himanshu@ansh:~$ size apl test -t
   text    data     bss     dec     hex filename
   2071     568       8    2647     a57 apl
   2071     568       8    2647     a57 test
   4142    1136      16    5294    14ae (TOTALS)
```

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.

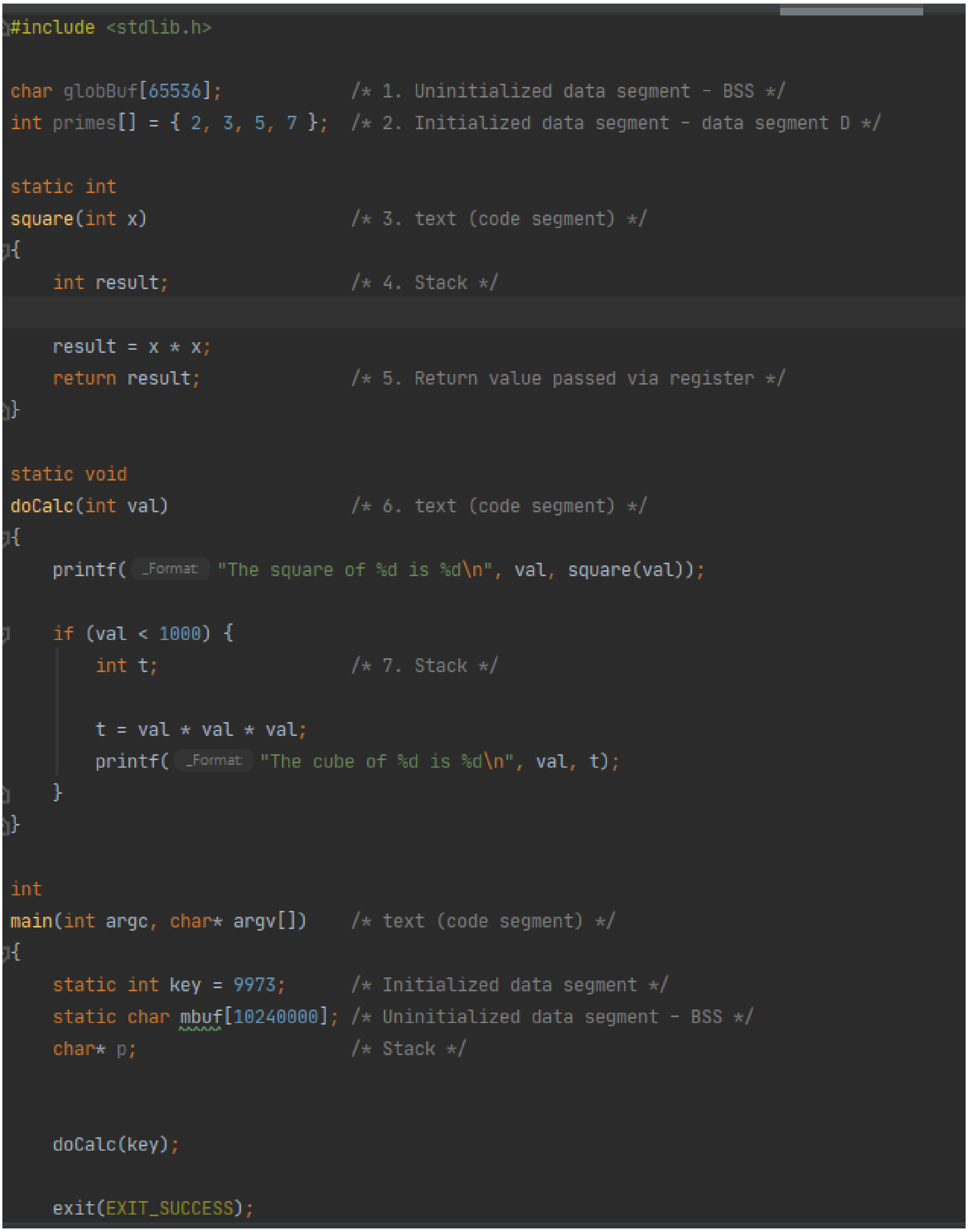
This command has several options that can be used by adding flags to the command.

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.

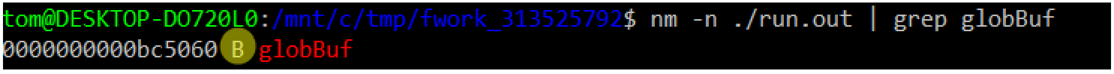
Section 1:



```c
#include <stdlib.h>

char globBuf[65536];            /* 1. Uninitialized data segment - BSS */
int primes[] = { 2, 3, 5, 7 };  /* 2. Initialized data segment - data segment D */

static int
square(int x)                   /* 3. text (code segment) */
{
    int result;                 /* 4. Stack */

    result = x * x;
    return result;              /* 5. Return value passed via register */
}

static void
doCalc(int val)                 /* 6. text (code segment) */
{
    printf( _Format: "The square of %d is %d\n", val, square(val));

    if (val < 1000) {
        int t;                  /* 7. Stack */

        t = val * val * val;
        printf( _Format: "The cube of %d is %d\n", val, t);
    }
}

int
main(int argc, char* argv[])    /* text (code segment) */
{
    static int key = 9973;      /* Initialized data segment */
    static char mbuf[10240000]; /* Uninitialized data segment - BSS */
    char* p;                    /* Stack */


    doCalc(key);

    exit(EXIT_SUCCESS);
```

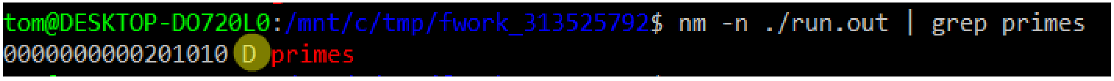
Section 2:

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.



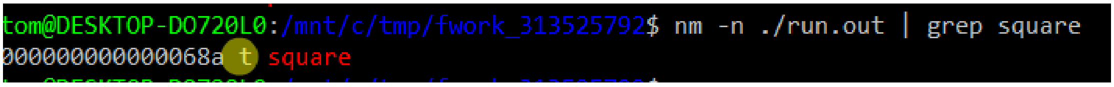
```
tom@DESKTOP-DO720L0:/mnt/c/tmp/fwork_313525792$ nm -n ./run.out | grep globBuf
0000000000bc5060 B globBuf
```

2. 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.



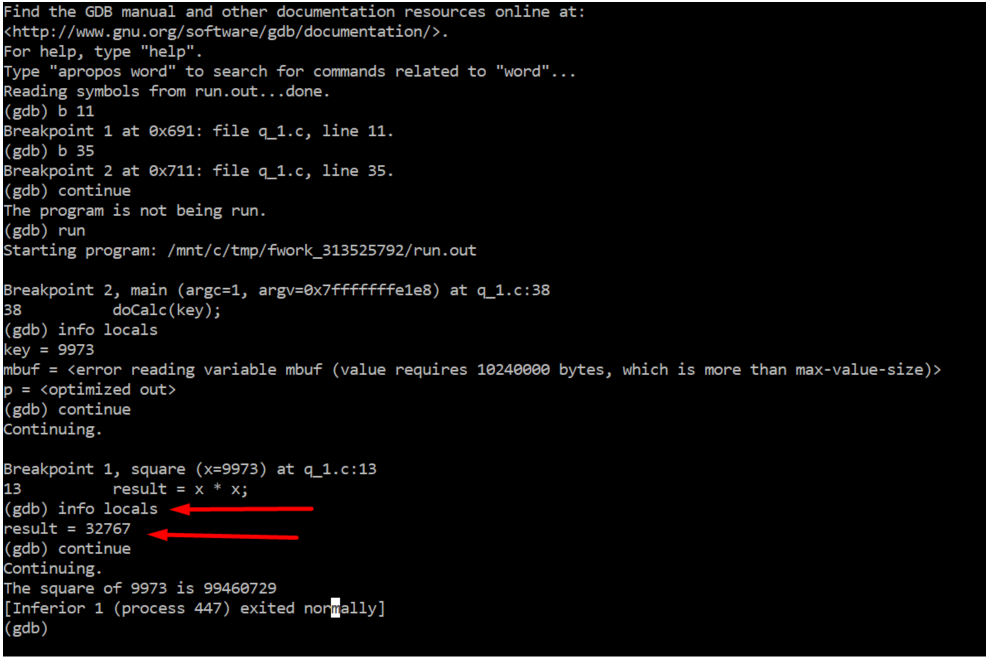
```
tom@DESKTOP-DO720L0:/mnt/c/tmp/fwork_313525792$ nm -n ./run.out | grep primes
0000000000201010 D primes
```

3. Question: Where is allocated?
   Line: line 9, square(int x)
   Answer: In the **text** area, which contains executable instructions.



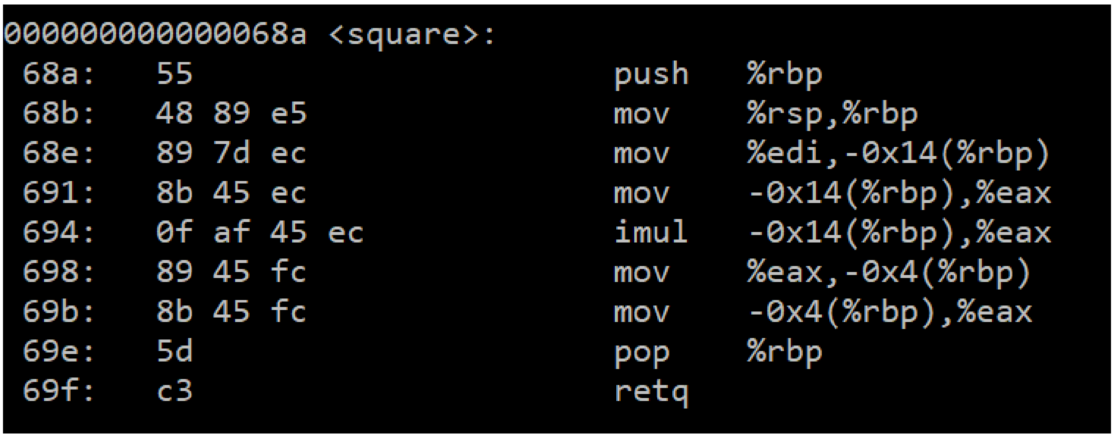
```
tom@DESKTOP-DO720L0:/mnt/c/tmp/fwork_313525792$ nm -n ./run.out | grep square
000000000000068a t square
```

4. 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.



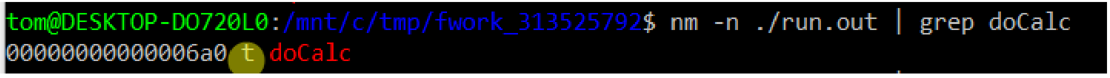
```
Find the GDB manual and other documentation resources online at:
<http://www.gnu.org/software/gdb/documentation/>.
For help, type "help".
Type "apropos word" to search for commands related to "word"...
Reading symbols from run.out...done.
(gdb) b 11
Breakpoint 1 at 0x691: file q_1.c, line 11.
(gdb) b 35
Breakpoint 2 at 0x711: file q_1.c, line 35.
(gdb) continue
The program is not being run.
(gdb) run
Starting program: /mnt/c/tmp/fwork_313525792/run.out

Breakpoint 2, main (argc=1, argv=0x7fffffffe1e8) at q_1.c:38
38          doCalc(key);
(gdb) info locals
key = 9973
mbuf = <error reading variable mbuf (value requires 10240000 bytes, which is more than max-value-size)>
p = <optimized out>
(gdb) continue
Continuing.

Breakpoint 1, square (x=9973) at q_1.c:13
13          result = x * x;
(gdb) info locals
result = 32767
(gdb) continue
Continuing.
The square of 9973 is 99460729
[Inferior 1 (process 447) exited normally]
(gdb)
```

5. 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.



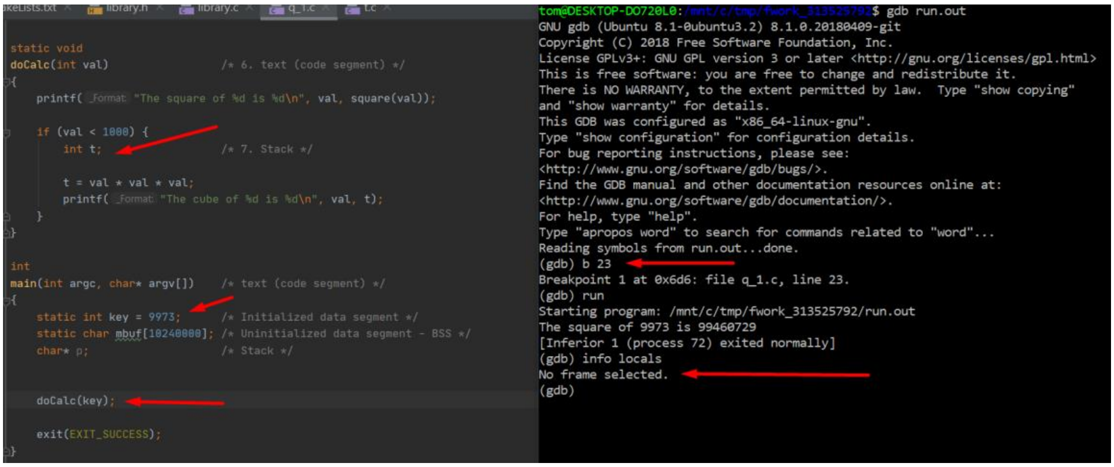
```
000000000000068a <square>:
 68a:   55                      push   %rbp
 68b:   48 89 e5                mov    %rsp,%rbp
 68e:   89 7d ec                mov    %edi,-0x14(%rbp)
 691:   8b 45 ec                mov    -0x14(%rbp),%eax
 694:   0f af 45 ec             imul   -0x14(%rbp),%eax
 698:   89 45 fc                mov    %eax,-0x4(%rbp)
 69b:   8b 45 fc                mov    -0x4(%rbp),%eax
 69e:   5d                      pop    %rbp
 69f:   c3                      retq
```

6. Question: Where is allocated?
   Line: line 18
   Answer: In the **text** area, which contains executable instructions.



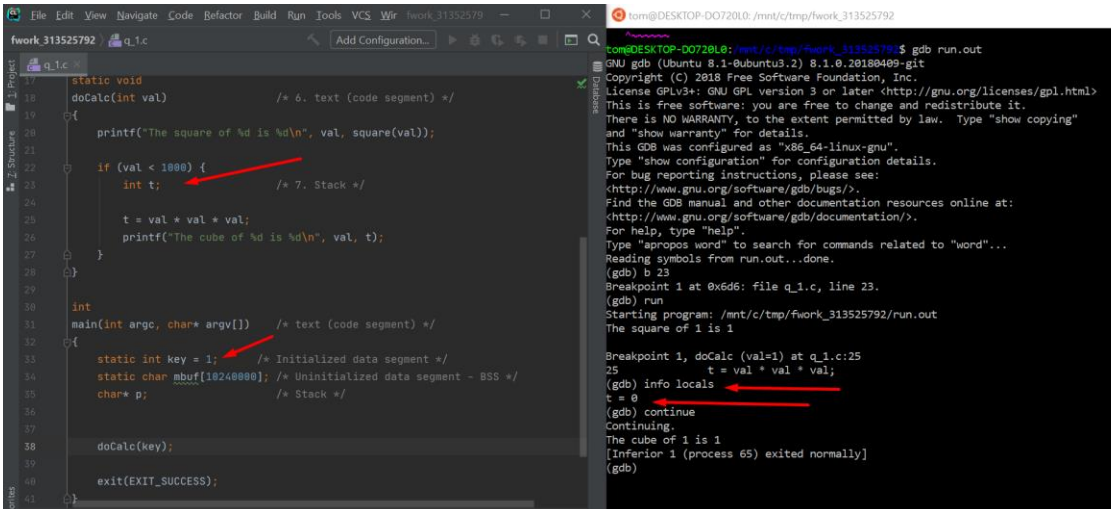
```
tom@DESKTOP-DO720L0:/mnt/c/tmp/fwork_313525792$ nm -n ./run.out | grep doCalc
00000000000006a0 t doCalc
```

7. 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 next commands:
   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.



```
static void
doCalc(int val)                /* 6. text (code segment) */
{
    printf( _Format: "The square of %d is %d\n", val, square(val));

    if (val < 1000) {
        int t;                 /* 7. Stack */

        t = val * val * val;
        printf( _format: "The cube of %d is %d\n", val, t);
    }
}

int
main(int argc, char* argv[])   /* text (code segment) */
{
    static int key = 9973;     /* Initialized data segment */
    static char mbuf[10240000]; /* Uninitialized data segment - BSS */
    char* p;                   /* Stack */


    doCalc(key);

    exit(EXIT_SUCCESS);
}
```

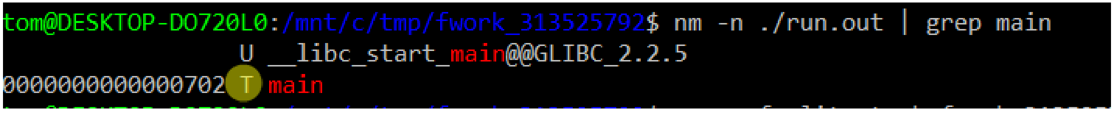
```
tom@DESKTOP-DO720L0:/mnt/c/tmp/fwork_313525792$ gdb run.out
GNU gdb (Ubuntu 8.1-0ubuntu3.2) 8.1.0.20180409-git
Copyright (C) 2018 Free Software Foundation, Inc.
License GPLv3+: GNU GPL version 3 or later <http://gnu.org/licenses/gpl.html>
This is free software: you are free to change and redistribute it.
There is NO WARRANTY, to the extent permitted by law.  Type "show copying"
and "show warranty" for details.
This GDB was configured as "x86_64-linux-gnu".
Type "show configuration" for configuration details.
For bug reporting instructions, please see:
<http://www.gnu.org/software/gdb/bugs/>.
Find the GDB manual and other documentation resources online at:
<http://www.gnu.org/software/gdb/documentation/>.
For help, type "help".
Type "apropos word" to search for commands related to "word"...
Reading symbols from run.out...done.
(gdb) b 23
Breakpoint 1 at 0x6d6: file q_1.c, line 23.
(gdb) run
Starting program: /mnt/c/tmp/fwork_313525792/run.out
The square of 9973 is 99460729
[Inferior 1 (process 72) exited normally]
(gdb) info locals
No frame selected.
(gdb)
```



```
17  static void
18  doCalc(int val)                /* 6. text (code segment) */
19  {
20      printf("The square of %d is %d\n", val, square(val));
21
22      if (val < 1000) {
23          int t;                 /* 7. Stack */
24
25          t = val * val * val;
26          printf("The cube of %d is %d\n", val, t);
27      }
28  }
29
30  int
31  main(int argc, char* argv[])   /* text (code segment) */
32  {
33      static int key = 1;        /* Initialized data segment */
34      static char mbuf[10240000]; /* Uninitialized data segment - BSS */
35      char* p;                   /* Stack */
36
37
38      doCalc(key);
39
40      exit(EXIT_SUCCESS);
41  }
```

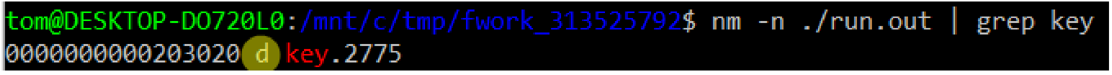
```
tom@DESKTOP-DO720L0:/mnt/c/tmp/fwork_313525792$ gdb run.out
GNU gdb (Ubuntu 8.1-0ubuntu3.2) 8.1.0.20180409-git
Copyright (C) 2018 Free Software Foundation, Inc.
License GPLv3+: GNU GPL version 3 or later <http://gnu.org/licenses/gpl.html>
This is free software: you are free to change and redistribute it.
There is NO WARRANTY, to the extent permitted by law.  Type "show copying"
and "show warranty" for details.
This GDB was configured as "x86_64-linux-gnu".
Type "show configuration" for configuration details.
For bug reporting instructions, please see:
<http://www.gnu.org/software/gdb/bugs/>.
Find the GDB manual and other documentation resources online at:
<http://www.gnu.org/software/gdb/documentation/>.
For help, type "help".
Type "apropos word" to search for commands related to "word"...
Reading symbols from run.out...done.
(gdb) b 23
Breakpoint 1 at 0x6d6: file q_1.c, line 23.
(gdb) run
Starting program: /mnt/c/tmp/fwork_313525792/run.out
The square of 1 is 1

Breakpoint 1, doCalc (val=1) at q_1.c:25
25              t = val * val * val;
(gdb) info locals
t = 0
(gdb) continue
Continuing.
The cube of 1 is 1
[Inferior 1 (process 65) exited normally]
(gdb)
```

8. <u>Question:</u> Where is allocated?
   <u>Line:</u> line 31, main(int argc, char* argv[])
   <u>Answer:</u> In the **text** area, which contains executable instructions.



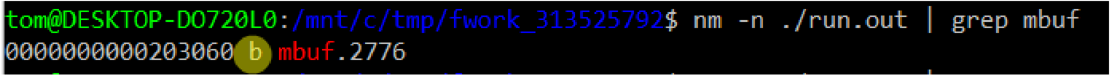
```
tom@DESKTOP-DO720L0:/mnt/c/tmp/fwork_313525792$ nm -n ./run.out | grep main
                 U __libc_start_main@@GLIBC_2.2.5
0000000000000702 T main
```

9. <u>Question:</u> Where is allocated?
   <u>Line:</u> line 33, static int key = 9973;
   <u>Answer:</u> 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.



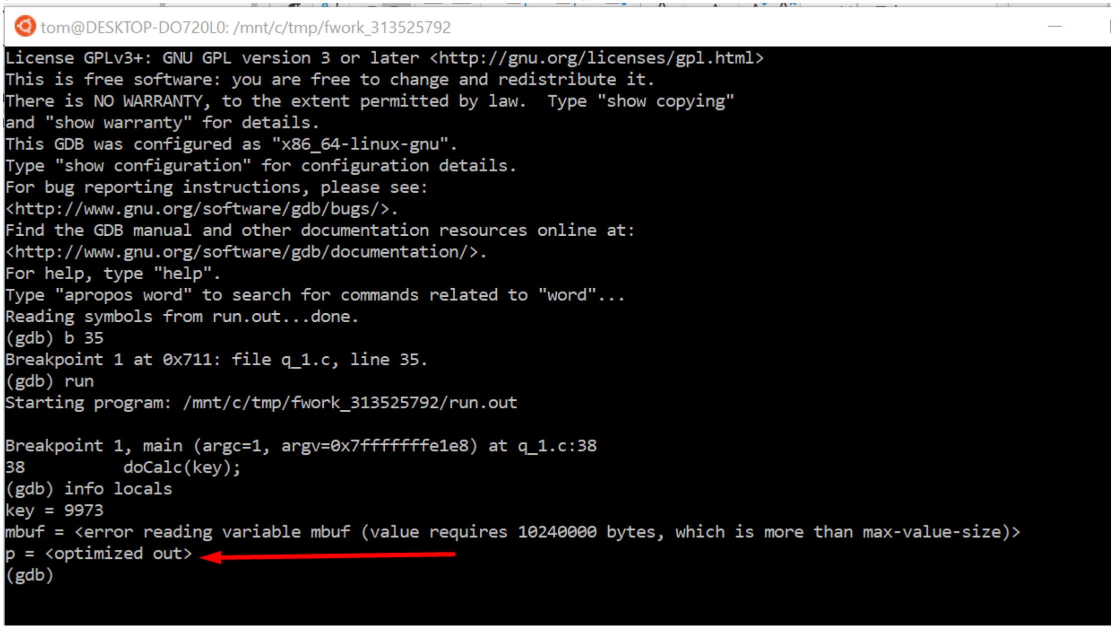
```
tom@DESKTOP-DO720L0:/mnt/c/tmp/fwork_313525792$ nm -n ./run.out | grep key
0000000000203020 d key.2775
```

10. <u>Question:</u> Where is allocated?
    <u>Line:</u> line 34, static char mbuf[10240000];
    <u>Answer:</u> 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.



```
tom@DESKTOP-DO720L0:/mnt/c/tmp/fwork_313525792$ nm -n ./run.out | grep mbuf
0000000000203060 b mbuf.2776
```

11. 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.



tom@DESKTOP-DO720L0: /mnt/c/tmp/fwork_313525792

```
License GPLv3+: GNU GPL version 3 or later <http://gnu.org/licenses/gpl.html>
This is free software: you are free to change and redistribute it.
There is NO WARRANTY, to the extent permitted by law.  Type "show copying"
and "show warranty" for details.
This GDB was configured as "x86_64-linux-gnu".
Type "show configuration" for configuration details.
For bug reporting instructions, please see:
<http://www.gnu.org/software/gdb/bugs/>.
Find the GDB manual and other documentation resources online at:
<http://www.gnu.org/software/gdb/documentation/>.
For help, type "help".
Type "apropos word" to search for commands related to "word"...
Reading symbols from run.out...done.
(gdb) b 35
Breakpoint 1 at 0x711: file q_1.c, line 35.
(gdb) run
Starting program: /mnt/c/tmp/fwork_313525792/run.out

Breakpoint 1, main (argc=1, argv=0x7fffffffe1e8) at q_1.c:38
38          doCalc(key);
(gdb) info locals
key = 9973
mbuf = <error reading variable mbuf (value requires 10240000 bytes, which is more than max-value-size)>
p = <optimized out>
(gdb)
```

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