1. Hello2: use GDB (along with any tools covered in class) to reverse "hello2.exe".

   a. How did you disassemble the executable? Provide the **steps** and **tools** used during the disassembly process.

      1.Gdb hello2.exe

      2.Info target // I notice it's it was packed up by UPX

```
(gdb) info target
Local exec file:
        `E:\umd_class\reverse_engineer\class_assignments\assignment3\hello2.exe', file type pei-i386.
        Entry point: 0x413620
        0x00401000 - 0x0040e000 is UPX0
        0x0040e000 - 0x00413a00 is UPX1
        0x00414000 - 0x00414200 is UPX2
```

      3. upx -d hello2.exe//unpack the value

```
E:\umd_class\reverse_engineer\class_assignments\assignment3>upx -d hello2.exe
                       Ultimate Packer for eXecutables
                          Copyright (C) 1996 - 2020
UPX 3.96w       Markus Oberhumer, Laszlo Molnar & John Reiser   Jan 23rd 2020

        File size         Ratio      Format      Name
   --------------------   ------   -----------   -----------
     60399 <-     37871   62.70%    win32/pe     hello2.exe

Unpacked 1 file.
```

      4.Gdb hello2.exe

      5.Info target

```
(gdb) info target
Symbols from "E:\umd_class\reverse_engineer\class_assignments\assignment3\hello2.exe".
Local exec file:
        `E:\umd_class\reverse_engineer\class_assignments\assignment3\hello2.exe', file type pei-i386.
        Entry point: 0x4012c0
        0x00401000 - 0x00409014 is .text
        0x0040a000 - 0x0040a02c is .data
        0x0040b000 - 0x0040b820 is .rdata
        0x0040c000 - 0x0040d73c is .eh_fram
        0x0040e000 - 0x0040ea34 is .bss
        0x0040f000 - 0x0040f6fc is .idata
        0x00410000 - 0x00410018 is .CRT
        0x00411000 - 0x00411020 is .tls
```

b. What is the behavior of the executable? Provide evidence in the **assembly code** to support your findings.

1.Call the rand function to generate random number.

```
push    ebp
mov     ebp, esp
and     esp, 0FFFFFFF0h
sub     esp, 20h
call    ___main
mov     [esp+20h+Time], 0 ; Time
call    _time
mov     [esp+20h+Time], eax ; Seed
call    _srand
call    _rand
```

function

srand

`void srand (unsigned int seed);`

Initialize random number generator

The pseudo-random number generator is initialized using the argument passed as *seed*.

std::rand

Defined in header <cstdlib>

`int rand();`

Returns a pseudo-random integral value between `0` and `RAND_MAX` (`0` and `RAND_MAX` included).

`std::srand()` seeds the pseudo-random number generator used by `rand()`. If `rand()` is used before any calls to `std::srand()`, `rand()` behaves as if it was seeded with `std::srand(1)`.

Each time `rand()` is seeded with `std::srand()`, it must produce the same sequence of values on successive calls.

Other functions in the standard library may call `rand`. It is implementation-defined which functions do so.

It is implementation-defined whether `rand()` is thread-safe.

2. There's a while loop, if satisfy the condition of comparing with the local variables, then output "hello world!" and continue, else return and exit the main function.

```
loc_4014BA:
mov     eax, [esp+20h+var_4]
cmp     eax, [esp+20h+var_8]
jl      short loc_4014A9
```

```
loc_4014A9:
mov     [esp+20h+Time], offset Buffer ; "hello world!"
call    _puts
add     [esp+20h+var_4], 1
```

```
mov     eax, 0
leave
retn
_main endp
```

3. I find out it is printing "Hello World" in random times.

2.Warmup: use GDB (along with any tools covered in class) to reverse "warmup.exe".

a.What did you learn about how this program operates? Provide **program inputs** and evidence in the **assembly code** to support your findings.

1.
```
call    ___main
cmp     [ebp+argc], 4
jnz     short loc_401423
```

b.
c.
d.
e.
```
jnz     short loc_401423

mov     eax, [ebp+argv]
add     eax, 4
mov     eax, [eax]
mov     [esp+30h+String], eax ; String
call    _atoi
mov     [esp+30h+var_4], eax
mov     eax, [ebp+argv]
add     eax, 8
mov     eax, [eax]
mov     [esp+30h+String], eax ; String
call    _atol
mov     [esp+30h+var_8], eax
mov     eax, [ebp+argv]
add     eax, 0Ch
mov     eax, [eax]
mov     [esp+30h+String], eax ; String
call    _atof
mov     [esp+30h+var_14], eax
fild    [esp+30h+var_14]
fstp    [esp+30h+var_C]
mov     [esp+30h+String], offset Buffer ; "Good! But what are the argument types?"
call    _puts
mov     eax, 0
jmp     short locret_401428

loc_401423:
mov     eax, 1

locret_401428:
leave
retn
_main endp
```

1. Compare if the input meet number of 3 (including “this” pointer”)
2. If equal
    a. Do some adding
    b. Convert string to int
    c. Convert string to long

d. Convert string to float

e. Puts "Good! But what are the argument types?"

f. Return 0

3. Else

Return 1

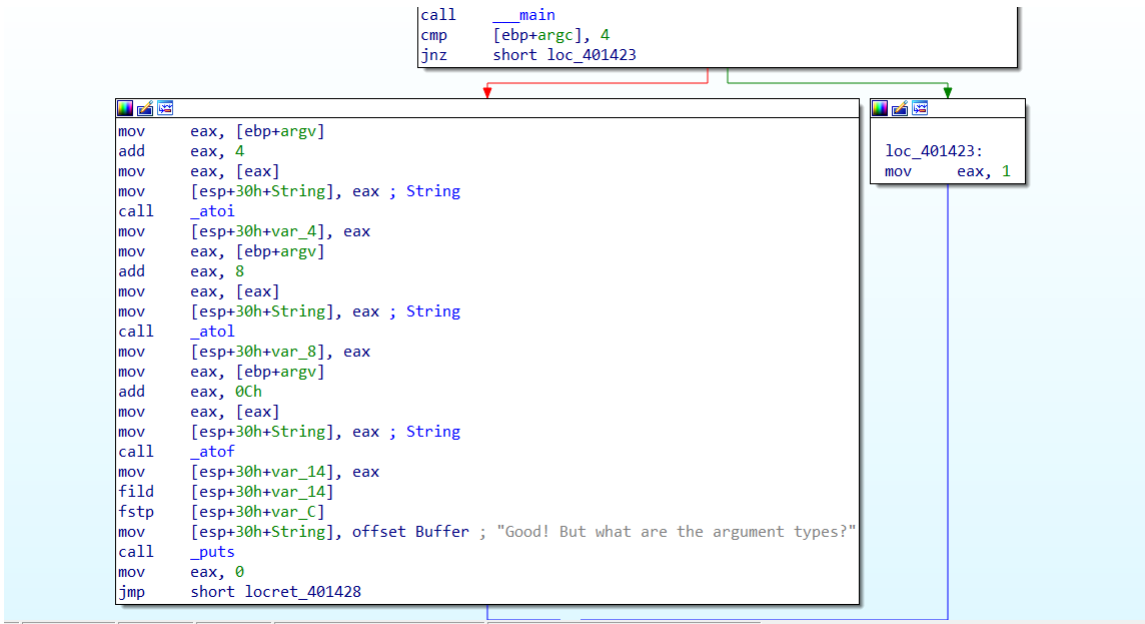
Input :

```
; Attributes: bp-based frame fuzzy-sp

; int __cdecl main(int argc, const char **argv, const char **envp)
public _main
_main proc near

String= dword ptr -30h
var_14= dword ptr -14h
var_C= dword ptr -0Ch
var_8= dword ptr -8
var_4= dword ptr -4
argc= dword ptr  8
argv= dword ptr  0Ch
envp= dword ptr  10h

push    ebp
mov     ebp, esp
and     esp, 0FFFFFFF0h
sub     esp, 30h
call    ___main
cmp     [ebp+argc], 4
jnz     short loc_401423
```

There are 3 inputs.

And in the program, there are local variables, a string, var_4,var_8,var_14,var_c.

b. Does main() call any functions in this program? If so, what are their names, number and types parameters? Provide evidence in the **assembly code** to support your findings.



The main function calls _atoi,_atol,_atof, _puts.

Parameters for the function:

Name	Number of parameters	Parameters type
__ atoi	1	String
__atol	1	String
__atof	1	String
_puts	1	char

```
; Attributes: thunk

; int __cdecl atoi(const char *String)
public _atoi
_atoi proc near

String= dword ptr  4

jmp     ds:__imp__atoi
_atoi endp
```

```
; Attributes: thunk

; int __cdecl atol(const char *String)
public _atol
_atol proc near

String= dword ptr  4

jmp     ds:__imp__atol
_atol endp
```

```
; Attributes: thunk

; double __cdecl atof(const char *String)
public _atof
_atof proc near

String= dword ptr  4

jmp     ds:__imp__atof
_atof endp
```

```
; Attributes: thunk

; int __cdecl puts(const char *Buffer)
public _puts
_puts proc near

Buffer= dword ptr  4

jmp     ds:__imp__puts
_puts endp
```