CS 410 Binary to C++ With Security Vulnerabilities Activity Template

Step 1: Convert the binary file to assembly code.
Step 2: Explain the functionality of the blocks of assembly code.

Blocks of Assembly Code	Explanation of Functionality
_Z11DisplayMenuv: .LFB14: .cfi_startproc pushq %rbp	_Z11DisplayMenuv:: Beginning of the function DisplayMenu. .LFB14:: Label marking the beginning of the function's basic block. .cfi_startproc: Marks the start of a new procedure. pushq %rbp: Pushes the value of the base pointer register %rbp onto the stack. This is done to save the caller's base pointer.
.cfi_def_cfa_offset 16 .cfi_offset 6, -16 movq %rsp, %rbp .cfi_def_cfa_register 6	.cfi_def_cfa_offset 16: Specifies the offset from %rsp where the Canonical Frame Address (CFA) is found. .cfi_offset 6, -16: Specifies that %rbp is located at an offset of -16 bytes from the CFA. movq %rsp, %rbp: Moves the value of the stack pointer %rsp into the base pointer register %rbp, establishing a new stack frame for the function. .cfi_def_cfa_register 6: Specifies %rbp as the canonical frame address register.

leaq .LC0(%rip), %rdi movl $0, %eax call printf@PLT	leaq .LC0(%rip), %rdi: Loads the effective address of the string .LC0 into register %rdi. This prepares the first argument for the printf function call. movl $0, %eax: Moves the value 0 into the %eax register. This is usually used to indicate success or no error. call printf@PLT: Calls the printf function to print the string specified by %rdi.
leaq .LC1(%rip), %rdi movl $0, %eax call printf@PLT	leaq .LC1(%rip), %rdi: Loads the effective address of the string .LC1 into register %rdi. This prepares the first argument for the printf function call. movl $0, %eax: Moves the value 0 into the %eax register. This is usually used to indicate success or no error. call printf@PLT: Calls the printf function to print the string specified by %rdi.
leaq .LC2(%rip), %rdi movl $0, %eax call printf@PLT	leaq .LC2(%rip), %rdi: Loads the effective address of the string .LC1 into register %rdi. This prepares the first argument for the printf function call. movl $0, %eax: Moves the value 0 into the %eax register. This is usually used to indicate success or no error. call printf@PLT: Calls the printf function to print the string specified by %rdi.

leaq .LC3(%rip), %rdi movl $0, %eax call printf@PLT	leaq .LC3(%rip), %rdi: Loads the effective address of the string .LC3 into register %rdi. This prepares the first argument for the printf function call. movl $0, %eax: Moves the value 0 into the %eax register. This is usually used to indicate success or no error. call printf@PLT: Calls the printf function to print the string specified by %rdi.
leaq .LC4(%rip), %rdi movl $0, %eax call printf@PLT	leaq .LC4(%rip), %rdi: Loads the effective address of the string .LC4 into register %rdi. This prepares the first argument for the printf function call. movl $0, %eax: Moves the value 0 into the %eax register. This is usually used to indicate success or no error. call printf@PLT: Calls the printf function to print the string specified by %rdi.
leaq .LC0(%rip), %rdi movl $0, %eax call printf@PLT	leaq .LC0(%rip), %rdi: Loads the effective address of the string .LC0 into register %rdi. This prepares the first argument for the printf function call. movl $0, %eax: Moves the value 0 into the %eax register. This is usually used to indicate success or no error. call printf@PLT: Calls the printf function to print the string specified by %rdi. This is a redundant call.
nop	nop: No operation. This is often used as a placeholder or for padding.

popq %rbp .cfi_def_cfa 7, 8 ret .cfi_endproc	popq %rbp: Pops the value from the stack into the base pointer register %rbp, restoring the previous stack frame. .cfi_def_cfa 7, 8: Specifies that %rsp is the canonical frame address and that it's located 8 bytes above %rsp. ret: Returns control flow to the caller. .cfi_endproc: Marks the end of the procedure.
.LFE14:	.LFE14:: This is a local function end (LFE) marker. It marks the end of the assembly code for the function _Z11DisplayMenuv. The number 14 is an identifier.
Dump of assembler code for function main:	Marks the beginning of the dump for the main function.
0x0000000000000787 <+0>: push %rbp	push %rbp: Pushes the value of the base pointer register %rbp onto the stack. This is done to save the previous frame pointer before setting up a new one.
0x0000000000000788 <+1>: mov %rsp,%rbp	mov %rsp,%rbp: Moves the value of the stack pointer %rsp into the base pointer %rbp. This sets up the stack frame for the current function.
0x000000000000078b <+4>: sub $0x20,%rsp	sub $0x20,%rsp: Subtracts 0x20 (32 in decimal) from the stack pointer %rsp, making space on the stack for local variables.
0x000000000000078f <+8>: mov %fs:0x28,%rax	mov %fs:0x28,%rax: Moves the value at the memory address %fs:0x28 into the general-purpose register %rax. This is often used for accessing the Thread Local Storage (TLS).
0x0000000000000798 <+17>: mov %rax,-0x8(%rbp)	mov %rax,-0x8(%rbp): Moves the value of %rax (which contains the TLS information) into the memory location at %rbp-0x8. This likely sets up exception handling or stack canary.

0x000000000000079c <+21>: xor %eax,%eax	xor %eax,%eax: XORs the %eax register with itself, effectively setting it to zero. This is often used to clear a register for subsequent operations.
0x000000000000079e <+23>: movl $0x0,-0x14(%rbp)	movl $0x0,-0x14(%rbp): Moves the immediate value 0x0 into the memory location at %rbp-0x14. This likely initializes a local variable to zero.
0x00000000000007a5 <+30>: mov -0x14(%rbp),%eax	mov -0x14(%rbp),%eax: Moves the value stored at %rbp-0x14 (which was initialized to zero) into the %eax register.
0x00000000000007a8 <+33>: cmp $0x5,%eax	cmp $0x5,%eax: Compares the value in %eax with 0x5 (decimal 5).
0x00000000000007ab <+36>: je 0x8f9 <main+370>	je 0x8f9 <main+370>: Jumps to the address 0x8f9 (which likely corresponds to the end of the main function) if the previous comparison resulted in equality (i.e., if %eax is equal to 0x5).
0x00000000000007b1 <+42>: lea 0x1dc(%rip),%rdi # 0x994	lea 0x1dc(%rip),%rdi: Loads the effective address of the string located 0x1dc bytes ahead of the instruction pointer (%rip) into the destination register %rdi. This likely prepares the address of a string to be printed.
0x00000000000007b8 <+49>: callq 0x5c0 <puts@plt>	callq 0x5c0 <puts@plt>: Calls the puts function located at address 0x5c0. This likely prints the string stored at the address %rdi.
0x00000000000007bd <+54>: lea 0x1e1(%rip),%rdi # 0x9a5	lea 0x1e1(%rip),%rdi: This instruction loads the effective address (address relative to the instruction pointer %rip) of the string located 0x1e1 bytes ahead of the current instruction into the destination register %rdi. This likely prepares the address of a string to be printed.
0x00000000000007c4 <+61>: callq 0x5c0 <puts@plt>	callq 0x5c0 <puts@plt>: This instruction calls the puts function located at address 0x5c0. The address to print (presumably stored in %rdi) is passed as an argument to puts.

0x00000000000007c9 <+66>: lea 0x1e2(%rip),%rdi # 0x9b2	lea 0x1e2(%rip),%rdi: Similar to the first lea instruction, this loads the effective address of another string to be printed.
0x00000000000007d0 <+73>: callq 0x5c0 <puts@plt>	callq 0x5c0 <puts@plt>: Calls the puts function again, likely to print the string whose address is in %rdi.
0x00000000000007d5 <+78>: lea 0x1e6(%rip),%rdi # 0x9c2	lea 0x1e6(%rip),%rdi: Loads the effective address of another string to be printed.
0x00000000000007dc <+85>: callq 0x5c0 <puts@plt>	callq 0x5c0 <puts@plt>: Calls the puts function again to print the string.
0x00000000000007e1 <+90>: lea 0x1ea(%rip),%rdi # 0x9d2	lea 0x1ea(%rip),%rdi: Loads the effective address of another string to be printed.
0x00000000000007e8 <+97>: callq 0x5c0 <puts@plt>	callq 0x5c0 <puts@plt>: Calls the puts function again to print the string.
0x00000000000007ed <+102>: lea 0x1a0(%rip),%rdi # 0x994	lea 0x1a0(%rip),%rdi: This instruction is computing the effective address of the string located at 0x1a0 bytes ahead of the instruction pointer (%rip) and storing it in the %rdi register. This likely prepares the address of a string to be printed.
0x00000000000007f4 <+109>: callq 0x5c0 <puts@plt>	callq 0x5c0 <puts@plt>: This instruction calls the puts function located at address 0x5c0. The address of the string (presumably stored in %rdi) is passed as an argument to puts.
0x00000000000007f9 <+114>: lea -0x14(%rbp),%rax	lea -0x14(%rbp),%rax: This instruction calculates the effective address of a variable located at -0x14 bytes from the base pointer (%rbp) and stores it in the %rax register.
0x00000000000007fd <+118>: mov %rax,%rsi	mov %rax,%rsi: This instruction moves the value of %rax (the address of the variable) into the %rsi register, likely preparing it as an argument for a function call.

0x0000000000000800 <+121>: lea 0x1d9(%rip),%rdi # 0x9e0	lea 0x1d9(%rip),%rdi: Similar to the first lea instruction, this calculates the effective address of a format string located 0x1d9 bytes ahead of the instruction pointer (%rip) and stores it in %rdi. This is likely preparing the format string for a subsequent scanf function call.
0x0000000000000807 <+128>: mov $0x0,%eax	mov $0x0,%eax: This instruction moves the immediate value 0 into the %eax register, likely preparing it for use as an argument for a function call.
0x000000000000080c <+133>: callq 0x5e0 <scanf@plt>	callq 0x5e0 <scanf@plt>: This instruction calls the scanf function located at address 0x5e0, likely to read input from the user.
0x0000000000000811 <+138>: mov -0x14(%rbp),%eax	mov -0x14(%rbp),%eax: This instruction moves the value stored at -0x14 bytes from the base pointer (%rbp) into the %eax register. This likely retrieves user input.
0x0000000000000814 <+141>: cmp $0x1,%eax	cmp $0x1,%eax: This instruction compares the value in %eax (presumably user input) to the immediate value 1.
0x0000000000000817 <+144>: jne 0x85d <main+214>	jne 0x85d <main+214>: This instruction jumps to address 0x85d (which corresponds to line main+214) if the previous comparison result was not equal (i.e., if user input is not equal to 1).
0x0000000000000819 <+146>: lea -0xc(%rbp),%rdx	lea -0xc(%rbp),%rdx: This instruction calculates the effective address of a variable located at -0xc bytes from the base pointer (%rbp) and stores it in the %rdx register.
0x000000000000081d <+150>: lea -0x10(%rbp),%rax	lea -0x10(%rbp),%rax: This instruction calculates the effective address of a variable located at -0x10 bytes from the base pointer (%rbp) and stores it in the %rax register.

0x0000000000000821 <+154>: mov %rax,%rsi	mov %rax,%rsi: This instruction moves the value of %rax (the address of a variable) into the %rsi register, likely preparing it as an argument for a function call.
0x0000000000000824 <+157>: lea 0x1b8(%rip),%rdi # 0x9e3	lea 0x1b8(%rip),%rdi: Similar to previous lea instructions, this calculates the effective address of a format string located 0x1b8 bytes ahead of the instruction pointer (%rip) and stores it in %rdi. This is likely preparing the format string for a subsequent scanf function call.
0x000000000000082b <+164>: mov $0x0,%eax	mov $0x0,%eax: This instruction moves the immediate value 0 into the %eax register, likely preparing it for use as an argument for a function call.
0x0000000000000830 <+169>: callq 0x5e0 <scanf@plt>	callq 0x5e0 <scanf@plt>: This instruction calls the scanf function located at address 0x5e0, likely to read input from the user.
0x0000000000000835 <+174>: mov -0x10(%rbp),%edx	mov -0x10(%rbp),%edx: This instruction moves the value stored at -0x10 bytes from the base pointer (%rbp) into the %edx register.
0x0000000000000838 <+177>: mov -0xc(%rbp),%eax	mov -0xc(%rbp),%eax: This instruction moves the value stored at -0xc bytes from the base pointer (%rbp) into the %eax register.
0x000000000000083b <+180>: mov %edx,%ecx	mov %edx,%ecx: This instruction moves the value in %edx into the %ecx register.
0x000000000000083d <+182>: sub %eax,%ecx	sub %eax,%ecx: This instruction subtracts the value in %eax from the value in %ecx and stores the result in %ecx.
0x000000000000083f <+184>: mov -0xc(%rbp),%edx	mov -0xc(%rbp),%edx: This instruction moves the value stored at -0xc bytes from the base pointer (%rbp) into the %edx register.
0x0000000000000842 <+187>: mov -0x10(%rbp),%eax	mov -0x10(%rbp),%eax: This instruction moves the value stored at -0x10 bytes from the base pointer (%rbp) into the %eax register.

0x0000000000000845 <+190>: mov %eax,%esi	mov %eax,%esi: This instruction moves the value in %eax into the %esi register, likely preparing it as an argument for a subsequent function call.
0x0000000000000847 <+192>: lea 0x19b(%rip),%rdi # 0x9e9	lea 0x19b(%rip),%rdi: Similar to previous lea instructions, this calculates the effective address of a format string located 0x19b bytes ahead of the instruction pointer (%rip) and stores it in %rdi. This is likely preparing the format string for a subsequent printf function call.
0x000000000000084e <+199>: mov $0x0,%eax	mov $0x0,%eax: This instruction moves the immediate value 0 into the %eax register, likely preparing it for use as an argument for a function call.
0x0000000000000853 <+204>: callq 0x5d0 <printf@plt>	callq 0x5d0 <printf@plt>: Calls the printf function located at address 0x5d0, presumably to print some formatted output.
0x0000000000000858 <+209>: jmpq 0x7a5 <main+30>	jmpq 0x7a5 <main+30>: Jumps to the address 0x7a5, which is likely the beginning of the main function or a loop.
0x000000000000085d <+214>: mov -0x14(%rbp),%eax	mov -0x14(%rbp),%eax: Moves the value stored at the memory address (%rbp - 0x14) into the %eax register.
0x0000000000000860 <+217>: cmp $0x2,%eax	cmp $0x2,%eax: Compares the value in %eax with the immediate value 0x2.
0x0000000000000863 <+220>: jne 0x8a9 <main+290>	jne 0x8a9 <main+290>: Jumps to address 0x8a9 if the previous comparison result was not equal (if %eax is not equal to 0x2).
0x0000000000000865 <+222>: lea -0xc(%rbp),%rdx	lea -0xc(%rbp),%rdx: Loads the effective address of the memory location (%rbp - 0xc) into the %rdx register.
0x0000000000000869 <+226>: lea -0x10(%rbp),%rax	lea -0x10(%rbp),%rax: Loads the effective address of the memory location (%rbp - 0x10) into the %rax register.
0x000000000000086d <+230>: mov %rax,%rsi	mov %rax,%rsi: Moves the value in %rax into the %rsi register.

0x0000000000000870 <+233>: lea 0x16c(%rip),%rdi # 0x9e3	lea 0x16c(%rip),%rdi # 0x9e3: Loads the effective address of a memory location calculated as 0x16c bytes ahead of the instruction pointer (%rip) into the %rdi register. This is likely preparing an argument for a scanf function call.
0x0000000000000877 <+240>: mov $0x0,%eax	mov $0x0,%eax: Moves the immediate value 0x0 into the %eax register.
0x000000000000087c <+245>: callq 0x5e0 <scanf@plt>	callq 0x5e0 <scanf@plt>: Calls the scanf function located at address 0x5e0, presumably to read some input from the user.
0x0000000000000881 <+250>: mov -0x10(%rbp),%edx	mov -0x10(%rbp),%edx: Moves the value stored at the memory address (%rbp - 0x10) into the %edx register.
0x0000000000000884 <+253>: mov -0xc(%rbp),%eax	mov -0xc(%rbp),%eax: Moves the value stored at the memory address (%rbp - 0xc) into the %eax register.
0x0000000000000887 <+256>: mov %edx,%ecx	mov %edx,%ecx: Moves the value in %edx into the %ecx register.
0x0000000000000889 <+258>: sub %eax,%ecx	sub %eax,%ecx: Subtracts the value in %eax from the value in %ecx, storing the result in %ecx.
0x000000000000088b <+260>: mov -0xc(%rbp),%edx	mov -0xc(%rbp),%edx: Moves the value stored at the memory address (%rbp - 0xc) into the %edx register.
0x000000000000088e <+263>: mov -0x10(%rbp),%eax	mov -0x10(%rbp),%eax: Moves the value stored at the memory address (%rbp - 0x10) into the %eax register.
0x0000000000000891 <+266>: mov %eax,%esi	mov %eax,%esi: Moves the value in %eax into the %esi register.
0x0000000000000893 <+268>: lea 0x14f(%rip),%rdi # 0x9e9	lea 0x14f(%rip),%rdi # 0x9e9: Loads the effective address of a memory location calculated as 0x14f bytes ahead of the instruction pointer (%rip) into the %rdi register. This is likely preparing an argument for a printf function call.

0x000000000000089a <+275>: mov $0x0,%eax	mov $0x0,%eax: Moves the immediate value 0x0 into the %eax register.
0x000000000000089f <+280>: callq 0x5d0 <printf@plt>	callq 0x5d0 <printf@plt>: Calls the printf function located at address 0x5d0, presumably to print some formatted output.
0x00000000000008a4 <+285>: jmpq 0x7a5 <main+30>	jmpq 0x7a5 <main+30>: Jumps to the address 0x7a5, likely for looping back to the beginning of the main function or a loop.
0x00000000000008a9 <+290>: mov -0x14(%rbp),%eax	mov -0x14(%rbp),%eax: Moves the value stored at the memory address (%rbp - 0x14) into the %eax register.
0x00000000000008ac <+293>: cmp $0x3,%eax	cmp $0x3,%eax: Compares the value in %eax with the immediate value 0x3.
0x00000000000008af <+296>: jne 0x7a5 <main+30>	jne 0x7a5 <main+30>: Jumps to address 0x7a5 if the previous comparison result was not equal (if %eax is not equal to 0x3).
0x00000000000008b5 <+302>: lea -0xc(%rbp),%rdx	lea -0xc(%rbp),%rdx: Loads the effective address of the memory location (%rbp - 0xc) into the %rdx register.
0x00000000000008b9 <+306>: lea -0x10(%rbp),%rax	lea -0x10(%rbp),%rax: Loads the effective address of the memory location (%rbp - 0x10) into the %rax register.
0x00000000000008bd <+310>: mov %rax,%rsi	mov %rax,%rsi: Moves the value in %rax into the %rsi register.
0x00000000000008c0 <+313>: lea 0x11c(%rip),%rdi # 0x9e3	lea 0x11c(%rip),%rdi # 0x9e3: Loads the effective address of a memory location calculated as 0x11c bytes ahead of the instruction pointer (%rip) into the %rdi register. This is likely preparing an argument for a scanf function call.
0x00000000000008c7 <+320>: mov $0x0,%eax	mov $0x0,%eax: Moves the immediate value 0x0 into the %eax register.
0x00000000000008cc <+325>: callq 0x5e0 <scanf@plt>	callq 0x5e0 <scanf@plt>: Calls the scanf function located at address 0x5e0, presumably to read some input from the user.

0x00000000000008d1 <+330>: mov -0x10(%rbp),%edx	mov -0x10(%rbp),%edx: Moves the value stored at the memory address (%rbp - 0x10) into the %edx register.
0x00000000000008d4 <+333>: mov -0xc(%rbp),%eax	mov -0xc(%rbp),%eax: Moves the value stored at the memory address (%rbp - 0xc) into the %eax register.
0x00000000000008d7 <+336>: mov %edx,%ecx	mov %edx,%ecx: Moves the value in %edx into the %ecx register.
0x00000000000008d9 <+338>: sub %eax,%ecx	sub %eax,%ecx: Subtracts the value in %eax from the value in %ecx, storing the result in %ecx.
0x00000000000008db <+340>: mov -0xc(%rbp),%edx	mov -0xc(%rbp),%edx: Moves the value stored at the memory address (%rbp - 0xc) into the %edx register.
0x00000000000008de <+343>: mov -0x10(%rbp),%eax	mov -0x10(%rbp),%eax: Moves the value stored at the memory address (%rbp - 0x10) into the %eax register.
0x00000000000008e1 <+346>: mov %eax,%esi	mov %eax,%esi: Moves the value in %eax into the %esi register.
0x00000000000008e3 <+348>: lea 0xff(%rip),%rdi # 0x9e9	lea 0xff(%rip),%rdi # 0x9e9: Loads the effective address of a memory location calculated as 0xff bytes ahead of the instruction pointer (%rip) into the %rdi register. This is likely preparing an argument for a printf function call.
0x00000000000008ea <+355>: mov $0x0,%eax	mov $0x0,%eax: Moves the immediate value 0x0 into the %eax register.
0x00000000000008ef <+360>: callq 0x5d0 <printf@plt>	callq 0x5d0 <printf@plt>: Calls the printf function located at address 0x5d0, presumably to print some formatted output.
0x00000000000008f4 <+365>: jmpq 0x7a5 <main+30>	jmpq 0x7a5 <main+30>: Jumps to the address 0x7a5, likely for looping back to the beginning of the main function or a loop.
0x00000000000008f9 <+370>: mov $0x0,%edi	mov $0x0,%edi: Moves the immediate value 0x0 into the %edi register.

0x00000000000008fe <+375>: callq 0x5f0 <exit@plt>	callq 0x5f0 <exit@plt>: Calls the exit function located at address 0x5f0, presumably to terminate the program execution.

Step 3: Convert the assembly code to binary.
Step 4: Convert the assembly code to C++ code.

Blocks of Assembly Code	C++ Code
_Z11DisplayMenuv: .LFB14: .cfi_startproc	double DisplayMenu () {
leaq .LC0(%rip), %rdi movl $0, %eax call printf@PLT	printf ("----------------");
leaq .LC1(%rip), %rdi movl $0, %eax call printf@PLT	printf ("- 1)Add -");
leaq .LC2(%rip), %rdi movl $0, %eax call printf@PLT	printf ("- 2)Subtract -");
leaq .LC3(%rip), %rdi movl $0, %eax call printf@PLT	printf ("- 3)Multiply -");
leaq .LC4(%rip), %rdi movl $0, %eax call printf@PLT	printf ("- 4)Exit -");
leaq .LC4(%rip), %rdi movl $0, %eax call printf@PLT	printf ("----------------"); }
0x0000000000000787 <+0>: push %rbp 0x0000000000000788 <+1>: mov %rsp,%rbp 0x000000000000078b <+4>: sub $0x20,%rsp	int main () {
0x000000000000079e <+23>: movl $0x0,-0x14(%rbp)	int choice = 0;
0x00000000000007a5 <+30>: mov -0x14(%rbp),%eax 0x00000000000007a8 <+33>: cmp $0x5,%eax 0x00000000000007ab <+36>: je 0x8f9 <main+370>	while (choice != 5) {

0x00000000000007b1 <+42>: lea 0x1dc(%rip),%rdi # 0x994
0x00000000000007b8 <+49>: callq 0x5c0 <puts@plt>
	printf ("----------------\n");
	printf ("- 1)Add -\n");
	printf ("- 2)Subtract -\n");
	printf ("- 3)Multiply -\n");
	printf ("- 4)Exit -\n");
	printf ("----------------\n");

0x00000000000007bd <+54>: lea 0x1e1(%rip),%rdi # 0x9a5 0x00000000000007c4 <+61>: callq 0x5c0 <puts@plt>	scanf ("%d", &choice);
0x0000000000000811 <+138>: mov -0x14(%rbp),%eax 0x0000000000000814 <+141>: cmp $0x1,%eax 0x0000000000000817 <+144>: jne 0x85d <main+214>	if (choice == 1) {
0x0000000000000819 <+146>: lea -0xc(%rbp),%rdx 0x000000000000081d <+150>: lea -0x10(%rbp),%rax 0x0000000000000821 <+154>: mov %rax,%rsi 0x0000000000000824 <+157>: lea 0x1b8(%rip),%rdi # 0x9e3 0x000000000000082b <+164>: mov $0x0,%eax 0x0000000000000830 <+169>: callq 0x5e0 <scanf@plt>	scanf("%d %d", &n1, &n2);

0x0000000000000835 <+174>: mov -0x10(%rbp),%edx 0x0000000000000838 <+177>: mov -0xc(%rbp),%eax 0x000000000000083b <+180>: mov %edx,%ecx 0x000000000000083d <+182>: sub %eax,%ecx 0x000000000000083f <+184>: mov -0xc(%rbp),%edx 0x0000000000000842 <+187>: mov -0x10(%rbp),%eax 0x0000000000000845 <+190>: mov %eax,%esi 0x0000000000000847 <+192>: lea 0x19b(%rip),%rdi # 0x9e9 0x000000000000084e <+199>: mov $0x0,%eax 0x0000000000000853 <+204>: callq 0x5d0 <printf@plt> 0x0000000000000858 <+209>: jmpq 0x7a5 <main+30>	printf("%d - %d = %d\n", n1, n2, n1-n2); }
0x000000000000085d <+214>: mov -0x14(%rbp),%eax 0x0000000000000860 <+217>: cmp $0x2,%eax 0x0000000000000863 <+220>: jne 0x8a9 <main+290>	else if (choice == 2) {
0x0000000000000865 <+222>: lea -0xc(%rbp),%rdx 0x0000000000000869 <+226>: lea -0x10(%rbp),%rax 0x000000000000086d <+230>: mov %rax,%rsi 0x0000000000000870 <+233>: lea 0x16c(%rip),%rdi # 0x9e3 0x0000000000000877 <+240>: mov $0x0,%eax 0x000000000000087c <+245>: callq 0x5e0 <scanf@plt>	scanf("%d %d", &n1, &n2);

0x0000000000000881 <+250>: mov -0x10(%rbp),%edx 0x0000000000000884 <+253>: mov -0xc(%rbp),%eax 0x0000000000000887 <+256>: mov %edx,%ecx 0x0000000000000889 <+258>: sub %eax,%ecx 0x000000000000088b <+260>: mov -0xc(%rbp),%edx 0x000000000000088e <+263>: mov -0x10(%rbp),%eax 0x0000000000000891 <+266>: mov %eax,%esi 0x0000000000000893 <+268>: lea 0x14f(%rip),%rdi # 0x9e9 0x000000000000089a <+275>: mov $0x0,%eax 0x000000000000089f <+280>: callq 0x5d0 <printf@plt> 0x00000000000008a4 <+285>: jmpq 0x7a5 <main+30>	printf("%d - %d = %d\n", n1, n2, n1-n2); }
0x00000000000008a9 <+290>: mov -0x14(%rbp),%eax 0x00000000000008ac <+293>: cmp $0x3,%eax 0x00000000000008af <+296>: jne 0x7a5 <main+30>	else if (choice == 3) {

0x00000000000008b5 <+302>: lea -0xc(%rbp),%rdx 0x00000000000008b9 <+306>: lea -0x10(%rbp),%rax 0x00000000000008bd <+310>: mov %rax,%rsi 0x00000000000008c0 <+313>: lea 0x11c(%rip),%rdi # 0x9e3 0x00000000000008c7 <+320>: mov $0x0,%eax 0x00000000000008cc <+325>: callq 0x5e0 <scanf@plt>	scanf("%d %d", &n1, &n2);
0x00000000000008d1 <+330>: mov -0x10(%rbp),%edx 0x00000000000008d4 <+333>: mov -0xc(%rbp),%eax 0x00000000000008d7 <+336>: mov %edx,%ecx 0x00000000000008d9 <+338>: sub %eax,%ecx 0x00000000000008db <+340>: mov -0xc(%rbp),%edx 0x00000000000008de <+343>: mov -0x10(%rbp),%eax 0x00000000000008e1 <+346>: mov %eax,%esi 0x00000000000008e3 <+348>: lea 0xff(%rip),%rdi # 0x9e9 0x00000000000008ea <+355>: mov $0x0,%eax 0x00000000000008ef <+360>: callq 0x5d0 <printf@plt>	printf("%d - %d = %d\n", n1, n2, n1-n2); } }
0x00000000000008f4 <+365>: jmpq 0x7a5 <main+30> 0x00000000000008f9 <+370>: mov $0x0,%edi 0x00000000000008fe <+375>: callq 0x5f0 <exit@plt>	exit (0); }