CS-4325 (Assignment 2) - Glenn Noronha - 1057121

Construct 1: declaring and initializing an int

```
int a = 2;
```

```
d:   c7 45 fc 02 00 00 00    movl   $0x2,-0x4(%rbp)
```

movl $0x2, -0x4(%rbp): stores value 2 in a local variable

Construct 2: adding ints

```
int a = 4, b = 6;
int add = a + b;
```

```
 d:   c7 45 fc 04 00 00 00    movl   $0x4,-0x4(%rbp)
14:   c7 45 f8 06 00 00 00    movl   $0x6,-0x8(%rbp)
1b:   8b 55 fc                mov    -0x4(%rbp),%edx
1e:   8b 45 f8                mov    -0x8(%rbp),%eax
21:   01 d0                   add    %edx,%eax
23:   89 45 f4                mov    %eax,-0xc(%rbp)
```

movl $0x4, -0x4(%rbp): store value 4 in a local variable
movl $0x6, -0x8(%rbp): store value 6 in a local variable
mov -0x4(%rbp), %edx: load value from -0x4(%rbp) into %edx
mov -0x8(%rbp), %eax: load value from -0x8(%rbp) into %eax
add %edx, %eax: add values in %edx and %eax
mov %eax, -0xc(%rbp): store result of addition into a local variable

Construct 3: printf()

```
4       printf("Hi\n");
```

```
0000000000000000 <printf>:
   0:   55                      push   %rbp
   1:   53                      push   %rbx
   2:   48 83 ec 38             sub    $0x38,%rsp
   6:   48 8d 6c 24 30          lea    0x30(%rsp),%rbp
   b:   48 89 4d 20             mov    %rcx,0x20(%rbp)
   f:   48 89 55 28             mov    %rdx,0x28(%rbp)
  13:   4c 89 45 30             mov    %r8,0x30(%rbp)
  17:   4c 89 4d 38             mov    %r9,0x38(%rbp)
  1b:   48 8d 45 28             lea    0x28(%rbp),%rax
  1f:   48 89 45 f0             mov    %rax,-0x10(%rbp)
  23:   48 8b 5d f0             mov    -0x10(%rbp),%rbx
  27:   b9 01 00 00 00          mov    $0x1,%ecx
  2c:   48 8b 05 00 00 00 00    mov    0x0(%rip),%rax        # 33 <printf+0x33>
  33:   ff d0                   call   *%rax
  35:   48 89 c1                mov    %rax,%rcx
  38:   48 8b 45 20             mov    0x20(%rbp),%rax
  3c:   49 89 d8                mov    %rbx,%r8
  3f:   48 89 c2                mov    %rax,%rdx
  42:   e8 00 00 00 00          call   47 <printf+0x47>
  47:   89 45 fc                mov    %eax,-0x4(%rbp)
  4a:   8b 45 fc                mov    -0x4(%rbp),%eax
  4d:   48 83 c4 38             add    $0x38,%rsp
  51:   5b                      pop    %rbx
  52:   5d                      pop    %rbp
  53:   c3                      ret
```

push %rbp: save the old base pointer
push %rbx: save the old value of %rbx
sub $0x38, %rsp: allocate 56 bytes on the stack
lea 0x30(%rsp), %rbp: load the effective address of 0x30(%rsp) into %rbp (setup stack frame)
mov %rcx, 0x20(%rbp): store %rcx in the local variable at 0x20(%rbp)
mov %rdx, 0x28(%rbp): store %rdx in the local variable at 0x28(%rbp)
mov %r8, 0x30(%rbp): store %r8 in the local variable at 0x30(%rbp)
mov %r9, 0x38(%rbp): store %r9 in the local variable at 0x38(%rbp)
lea 0x28(%rbp), %rax: load the address of the local variable at 0x28(%rbp) into %rax
mov %rax, -0x10(%rbp): store %rax (address of the local variable) into -0x10(%rbp)
mov -0x10(%rbp), %rbx: load the stored address from -0x10(%rbp) into %rbx

mov $0x1, %ecx: set the value 1 in %ecx (prepare an argument for the function call)
mov 0x0(%rip), %rax: load the function address from memory, relative to the instruction pointer
call *%rax: call the function at the address stored in %rax
mov %rax, %rcx: move the return value from the call into %rcx
mov 0x20(%rbp), %rax: load the value from 0x20(%rbp) (restore the argument from earlier) into %rax
mov %rbx, %r8: move the value from %rbx into %r8 (prepare argument for another function)
mov %rax, %rdx: move the value from %rax into %rdx (prepare argument for another function)
call 47 <printf+0x47>: call another function (could be a part of the printf execution)
mov %eax, -0x4(%rbp): store the return value from the function into -0x4(%rbp)
mov -0x4(%rbp), %eax: load the return value from -0x4(%rbp) into %eax
add $0x38, %rsp: restore the stack pointer (deallocate 56 bytes)
pop %rbx: restore the old value of %rbx from the stack
pop %rbp: restore the old base pointer
ret: return from the function

Construct 4: if, else statement

```c
int a = 5;
if (a == 5) {
    printf("a is 5\n");
}
else{
    printf("a is not 5\n");
}
```

```
68:   83 7d fc 05             cmpl   $0x5,-0x4(%rbp)
6c:   75 11                   jne    7f <main+0x2b>
6e:   48 8d 05 00 00 00 00    lea    0x0(%rip),%rax        # 75 <main+0x21>
75:   48 89 c1                mov    %rax,%rcx
78:   e8 83 ff ff ff          call   0 <printf>
7d:   eb 0f                   jmp    8e <main+0x3a>
7f:   48 8d 05 08 00 00 00    lea    0x8(%rip),%rax        # 8e <main+0x3a>
86:   48 89 c1                mov    %rax,%rcx
89:   e8 72 ff ff ff          call   0 <printf>
```

cmpl $0x5, -0x4(%rbp): compare the value in local variable with 5
jne 7f <main+0x2b>: jump to else block if not equal
If block (a == 5)
lea 0x0(%rip), %rax: load address for printf (if condition)
mov %rax, %rcx: move address into %rcx (argument for printf)
call 0 <printf>: call printf to print the 'if' condition message
jmp 8e <main+0x3a>: skip the 'else' block and jump to end
Else block
lea 0x8(%rip), %rax: load address for printf (else condition)
mov %rax, %rcx: move address into %rcx (argument for printf)
call 0 <printf>: call printf to print the 'else' condition message

Construct 5: while loop

```c
int counter = 0;
while (counter < 3)
{
    printf("%d\n", counter);
    counter++;
}
```

```
54:   55                    push   %rbp
55:   48 89 e5              mov    %rsp,%rbp
58:   48 83 ec 30           sub    $0x30,%rsp
5c:   e8 00 00 00 00        call   61 <main+0xd>
61:   c7 45 fc 00 00 00 00  movl   $0x0,-0x4(%rbp)
68:   eb 18                 jmp    82 <main+0x2e>
6a:   8b 45 fc              mov    -0x4(%rbp),%eax
6d:   89 c2                 mov    %eax,%edx
6f:   48 8d 05 00 00 00 00  lea    0x0(%rip),%rax        # 76 <main+0x22>
76:   48 89 c1              mov    %rax,%rcx
79:   e8 82 ff ff ff        call   0 <printf>
7e:   83 45 fc 01           addl   $0x1,-0x4(%rbp)
82:   83 7d fc 02           cmpl   $0x2,-0x4(%rbp)
86:   7e e2                 jle    6a <main+0x16>
88:   b8 00 00 00 00        mov    $0x0,%eax
8d:   48 83 c4 30           add    $0x30,%rsp
91:   5d                    pop    %rbp
92:   c3                    ret
93:   90                    nop
94:   90                    nop
```

movl $0x0, -0x4(%rbp): initialize local variable to 0
jmp 82 <main+0x2e>: jump to the comparison at the end of the loop
Loop start
mov -0x4(%rbp), %eax: load local variable into %eax
mov %eax, %edx: move value into %edx (for printf argument)
lea 0x0(%rip), %rax: load address for printf
mov %rax, %rcx: move address into %rcx (for printf argument)
call 0 <printf>: call printf to print value
addl $0x1, -0x4(%rbp): increment local variable by 1
Loop condition
cmpl $0x2, -0x4(%rbp): compare local variable with 2
jle 6a <main+0x16>: if local variable is <= 2, jump back to the loop start
Exit loop
mov $0x0, %eax: set return value to 0

Construct 6: for loop

```c
for(int i = 0; i < 3; i++){
    printf("%d\n", i);
}
```

```
0000000000000054 <main>:
  54:   55                    push   %rbp
  55:   48 89 e5              mov    %rsp,%rbp
  58:   48 83 ec 30           sub    $0x30,%rsp
  5c:   e8 00 00 00 00        call   61 <main+0xd>
  61:   c7 45 fc 00 00 00 00  movl   $0x0,-0x4(%rbp)
  68:   eb 18                 jmp    82 <main+0x2e>
  6a:   8b 45 fc              mov    -0x4(%rbp),%eax
  6d:   89 c2                 mov    %eax,%edx
  6f:   48 8d 05 00 00 00 00  lea    0x0(%rip),%rax        # 76 <main+0x22>
  76:   48 89 c1              mov    %rax,%rcx
  79:   e8 82 ff ff ff        call   0 <printf>
  7e:   83 45 fc 01           addl   $0x1,-0x4(%rbp)
  82:   83 7d fc 02           cmpl   $0x2,-0x4(%rbp)
  86:   7e e2                 jle    6a <main+0x16>
  88:   b8 00 00 00 00        mov    $0x0,%eax
  8d:   48 83 c4 30           add    $0x30,%rsp
  91:   5d                    pop    %rbp
  92:   c3                    ret
  93:   90                    nop
  94:   90                    nop
```

movl $0x0, -0x4(%rbp): initialize loop counter to 0
jmp 82 <main+0x2e>: jump to the loop condition check
Loop body
mov -0x4(%rbp), %eax: load loop counter into %eax
mov %eax, %edx: move loop counter into %edx (for printf argument)
lea 0x0(%rip), %rax: load address for printf
mov %rax, %rcx: move address into %rcx (for printf argument)
call 0 <printf>: call printf to print loop counter
addl $0x1, -0x4(%rbp): increment loop counter by 1

Loop condition
cmpl $0x2, -0x4(%rbp): compare loop counter with 2
jle 6a <main+0x16>: if loop counter is <= 2, repeat loop
Exit the loop
mov $0x0, %eax: set return value to 0

Construct 7: function definition and calling

```c
int add(int a, int b){
    return a + b;
}

int main()
{
    int sum = add(10, 20);
    return 0;
}
```

```
0000000000000000 <add>:
   0:   55                      push   %rbp
   1:   48 89 e5                mov    %rsp,%rbp
   4:   89 4d 10                mov    %ecx,0x10(%rbp)
   7:   89 55 18                mov    %edx,0x18(%rbp)
   a:   8b 55 10                mov    0x10(%rbp),%edx
   d:   8b 45 18                mov    0x18(%rbp),%eax
  10:   01 d0                   add    %edx,%eax
  12:   5d                      pop    %rbp
  13:   c3                      ret

0000000000000014 <main>:
  14:   55                      push   %rbp
  15:   48 89 e5                mov    %rsp,%rbp
  18:   48 83 ec 30             sub    $0x30,%rsp
  1c:   e8 00 00 00 00          call   21 <main+0xd>
  21:   ba 14 00 00 00          mov    $0x14,%edx
  26:   b9 0a 00 00 00          mov    $0xa,%ecx
  2b:   e8 d0 ff ff ff          call   0 <add>
  30:   89 45 fc                mov    %eax,-0x4(%rbp)
  33:   b8 00 00 00 00          mov    $0x0,%eax
  38:   48 83 c4 30             add    $0x30,%rsp
  3c:   5d                      pop    %rbp
  3d:   c3                      ret
  3e:   90                      nop
  3f:   90                      nop
```

add() function:
push %rbp: save old base pointer
mov %rsp, %rbp: set up stack frame
mov %ecx, 0x10(%rbp): store first argument (in %ecx) in a local variable
mov %edx, 0x18(%rbp): store second argument (in %edx) in a local variable
mov 0x10(%rbp), %edx: load first argument from local variable into %edx
mov 0x18(%rbp), %eax: load second argument from local variable into %eax
add %edx, %eax: add values in %edx and %eax
pop %rbp: restore base pointer
ret: return from the function (result in %eax)

main() function:
push %rbp: save old base pointer
mov %rsp, %rbp: set up stack frame
sub $0x30, %rsp: allocate 48 bytes of space on stack
call 21 <main+0xd>: call a function
mov $0x14, %edx: load value 20 (0x14) into %edx (argument 2 for `add`)
mov $0xa, %ecx: load value 10 (0xa) into %ecx (argument 1 for `add`)
call 0 <add>: call `add` function (which returns the sum of %ecx and %edx)
mov %eax, -0x4(%rbp): store result (sum) from `add` function into a local variable
mov $0x0, %eax: set return value to 0 (standard return for `main`)
add $0x30, %rsp: restore stack pointer (deallocate 48 bytes of stack space)
pop %rbp: restore old base pointer
ret: return from `main`

Construct 8: recursion

```c
int factorial(int n) {
    if (n == 0) {
        return 1;
    } else {
        return n * factorial(n - 1);
    }
}
```

```
0000000000000054 <factorial>:
  54:   55                      push   %rbp
  55:   48 89 e5                mov    %rsp,%rbp
  58:   48 83 ec 20             sub    $0x20,%rsp
  5c:   89 4d 10                mov    %ecx,0x10(%rbp)
  5f:   83 7d 10 00             cmpl   $0x0,0x10(%rbp)
  63:   75 07                   jne    6c <factorial+0x18
  65:   b8 01 00 00 00          mov    $0x1,%eax
  6a:   eb 11                   jmp    7d <factorial+0x29
  6c:   8b 45 10                mov    0x10(%rbp),%eax
  6f:   83 e8 01                sub    $0x1,%eax
  72:   89 c1                   mov    %eax,%ecx
  74:   e8 db ff ff ff          call   54 <factorial>
  79:   0f af 45 10             imul   0x10(%rbp),%eax
  7d:   48 83 c4 20             add    $0x20,%rsp
  81:   5d                      pop    %rbp
  82:   c3                      ret
```

push %rbp: save old base pointer
mov %rsp, %rbp: set up stack frame
sub $0x20, %rsp: allocate 32 bytes on stack
mov %ecx, 0x10(%rbp): store argument n in local variable at 0x10(%rbp)
cmpl $0x0, 0x10(%rbp): compare n with 0
jne 6c <factorial+0x18>: if n != 0, jump to recursive case
Base case
mov $0x1, %eax: if n == 0 set return value to 1
jmp 7d <factorial+0x29>: jump to return part of function
Recursive case
mov 0x10(%rbp), %eax: load n into %eax
sub $0x1, %eax: compute n - 1
mov %eax, %ecx: move the result into %ecx for recursive call
call 54 <factorial>: call factorial(n - 1)
imul 0x10(%rbp), %eax: multiply result of factorial(n - 1) by n
Return
add $0x20, %rsp: deallocate stack space
pop %rbp: restore old base pointer
ret: return from function with result in %eax

Construct 9: array

```c
int arr[5] = {1, 2, 3, 4, 5};
```

```
   d:   c7 45 e0 01 00 00 00    movl   $0x1,-0x20(%rbp)
  14:   c7 45 e4 02 00 00 00    movl   $0x2,-0x1c(%rbp)
  1b:   c7 45 e8 03 00 00 00    movl   $0x3,-0x18(%rbp)
  22:   c7 45 ec 04 00 00 00    movl   $0x4,-0x14(%rbp)
  29:   c7 45 f0 05 00 00 00    movl   $0x5,-0x10(%rbp)
```

movl $0x1, -0x20(%rbp): store 1 at offset -0x20 from base pointer
movl $0x2, -0x1c(%rbp): store 2 at offset -0x1c from base pointer
movl $0x3, -0x18(%rbp): store 3 at offset -0x18 from base pointer
movl $0x4, -0x14(%rbp): store 4 at offset -0x14 from base pointer
movl $0x5, -0x10(%rbp): store 5 at offset -0x10 from base pointer

Construct 10: 2D array

```
int arr[2][3] = {{1, 2, 3}, {4, 5, 6}};
```

```
  d:   c7 45 e0 01 00 00 00    movl   $0x1,-0x20(%rbp)
 14:   c7 45 e4 02 00 00 00    movl   $0x2,-0x1c(%rbp)
 1b:   c7 45 e8 03 00 00 00    movl   $0x3,-0x18(%rbp)
 22:   c7 45 ec 04 00 00 00    movl   $0x4,-0x14(%rbp)
 29:   c7 45 f0 05 00 00 00    movl   $0x5,-0x10(%rbp)
 30:   c7 45 f4 06 00 00 00    movl   $0x6,-0xc(%rbp)
```

movl $0x1, -0x20(%rbp): store 1 at offset -0x20
movl $0x2, -0x1c(%rbp): store 2 at offset -0x1c
movl $0x3, -0x18(%rbp): store 3 at offset -0x18
movl $0x4, -0x14(%rbp): store 4 at offset -0x14
movl $0x5, -0x10(%rbp): store 5 at offset -0x10
movl $0x6, -0x0c(%rbp): store 6 at offset -0x0c

Construct 11: pointers and pointer arithmetic

```
int a = 10;
int *ptr = &a;
*ptr += 5;
```

```
  d:   c7 45 f4 0a 00 00 00    movl   $0xa,-0xc(%rbp)
 14:   48 8d 45 f4             lea    -0xc(%rbp),%rax
 18:   48 89 45 f8             mov    %rax,-0x8(%rbp)
 1c:   48 8b 45 f8             mov    -0x8(%rbp),%rax
 20:   8b 00                   mov    (%rax),%eax
 22:   8d 50 05                lea    0x5(%rax),%edx
 25:   48 8b 45 f8             mov    -0x8(%rbp),%rax
 29:   89 10                   mov    %edx,(%rax)
```

movl $0xa, -0xc(%rbp): store 10 at offset -0xc from base pointer
lea -0xc(%rbp), %rax: load address of -0xc(%rbp) into %rax
mov %rax, -0x8(%rbp): store address in local variable at -0x8(%rbp)
mov -0x8(%rbp), %rax: load address stored in -0x8(%rbp) into %rax
mov (%rax), %eax: dereference the address and load value into %eax
lea 0x5(%rax), %rdx: add 5 to value in %rax and store result in %rdx
mov %rdx, %rax: move result of the adding into %rax
mov %eax, -0x4(%rbp): store result in local variable at -0x4(%rbp)

Construct 12: scanf()

```
scanf("%d", &num);
```

```
0000000000000000 <scanf>:
  0:   55                    push   %rbp
  1:   53                    push   %rbx
  2:   48 83 ec 38           sub    $0x38,%rsp
  6:   48 8d 6c 24 30        lea    0x30(%rsp),%rbp
  b:   48 89 4d 20           mov    %rcx,0x20(%rbp)
  f:   48 89 55 28           mov    %rdx,0x28(%rbp)
 13:   4c 89 45 30           mov    %r8,0x30(%rbp)
 17:   4c 89 4d 38           mov    %r9,0x38(%rbp)
 1b:   48 8d 45 28           lea    0x28(%rbp),%rax
 1f:   48 89 45 f0           mov    %rax,-0x10(%rbp)
 23:   48 8b 5d f0           mov    -0x10(%rbp),%rbx
 27:   b9 00 00 00 00        mov    $0x0,%ecx
 2c:   48 8b 05 00 00 00 00  mov    0x0(%rip),%rax        # 33 <scanf+0x33>
 33:   ff d0                 call   *%rax
 35:   48 89 c1              mov    %rax,%rcx
 38:   48 8b 45 20           mov    0x20(%rbp),%rax
 3c:   49 89 d8              mov    %rbx,%r8
 3f:   48 89 c2              mov    %rax,%rdx
 42:   e8 00 00 00 00        call   47 <scanf+0x47>
 47:   89 45 fc              mov    %eax,-0x4(%rbp)
 4a:   8b 45 fc              mov    -0x4(%rbp),%eax
 4d:   48 83 c4 38           add    $0x38,%rsp
 51:   5b                    pop    %rbx
 52:   5d                    pop    %rbp
 53:   c3                    ret
```

push %rbp: save old base pointer
push %rbx: save old value of %rbx
sub $0x38, %rsp: allocate 56 bytes on stack
lea 0x30(%rsp), %rbp: load effective address
mov %rcx, 0x20(%rbp): store first argument in a local variable
mov %rdx, 0x28(%rbp): store second argument in a local variable
mov %r8, 0x30(%rbp): store third argument in a local variable
mov %r9, 0x38(%rbp): store fourth argument in a local variable
lea 0x28(%rbp), %rax: load address of local variable into %rax
mov %rax, -0x10(%rbp): store address in a local variable

mov $0x0, %ecx: set up zero value
mov 0x0(%rip), %rax: load memory address of a function call
call *%rax: call the function at the loaded address (scanf)
mov %rax, %rcx: store the return value from scanf in %rcx
mov 0x20(%rbp), %rax: load previously stored argument into %rax
mov %rbx, %r8: move value from %rbx into %r8 (used for another argument)
mov %rax, %rdx: move value from %rax into %rdx
call 47 <scanf+0x47>:
mov %eax, -0x4(%rbp): store result from second call into a local variable
mov -0x4(%rbp), %eax: load stored result back into %eax
add $0x38, %rsp: deallocate stack space
pop %rbx: restore old value of %rbx
pop %rbp: restore old base pointer
ret: return from the function

Construct 13: structures

```
struct Person {
    char name[50];
    int age;
};
```

```
0000000000000000 <main>:
   0:   55                      push   %rbp
   1:   48 89 e5                mov    %rsp,%rbp
   4:   48 83 ec 60             sub    $0x60,%rsp
   8:   e8 00 00 00 00          call   d <main+0xd>
   d:   c7 45 f4 15 00 00 00    movl   $0x15,-0xc(%rbp)
  14:   48 8d 45 c0             lea    -0x40(%rbp),%rax
  18:   c7 00 4a 61 63 65       movl   $0x6563614a,(%rax)
  1e:   66 c7 40 04 79 00       movw   $0x79,0x4(%rax)
  24:   b8 00 00 00 00          mov    $0x0,%eax
  29:   48 83 c4 60             add    $0x60,%rsp
  2d:   5d                      pop    %rbp
  2e:   c3                      ret
  2f:   90                      nop
```

sub $0x60, %rsp: allocate space on stack
movl $0x15, -0xc(%rbp): store value 21 in struct
lea -0x40(%rbp), %rax: load address of struct
movl $0x4a61634, (%rax): store in struct field
movw $0x79, 0x4(%rax): store in struct field

Construct 14: bitwise AND and OR

```
unsigned int a = 5;
unsigned int b = 9;
unsigned int c = a & b;
unsigned int d = a | b;
```

```
  61:   c7 45 fc 05 00 00 00    movl   $0x5,-0x4(%rbp)
  68:   c7 45 f8 09 00 00 00    movl   $0x9,-0x8(%rbp)
  6f:   8b 45 fc                mov    -0x4(%rbp),%eax
  72:   23 45 f8                and    -0x8(%rbp),%eax
  75:   89 45 f4                mov    %eax,-0xc(%rbp)
  78:   8b 45 fc                mov    -0x4(%rbp),%eax
  7b:   0b 45 f8                or     -0x8(%rbp),%eax
  7e:   89 45 f0                mov    %eax,-0x10(%rbp)
```

movl $0x5, -0x8(%rbp): store 5 in a variable
movl $0x4, -0x4(%rbp): store 4 in a variable
and -0x8(%rbp), %eax: pbitwise AND on -0x8(%rbp) and store result in %eax
mov %eax, -0xc(%rbp): store result of AND in another location
or -0x8(%rbp), %eax: perform bitwise OR on -0x8(%rbp) (5) and store result in %eax
mov %eax, -0x10(%rbp): store result of OR in another location

Construct 15: switch

```c
int num = 1;
switch (num) {
    case 1:
        printf("Number is one\n");
        break;
    case 2:
        printf("Number is two\n");
        break;
    default:
        printf("Number is neither one nor two\n");
}
```

```
0000000000000054 <main>:
  54:   55                      push   %rbp
  55:   48 89 e5                mov    %rsp,%rbp
  58:   48 83 ec 30             sub    $0x30,%rsp
  5c:   e8 00 00 00 00          call   61 <main+0xd>
  61:   c7 45 fc 01 00 00 00    movl   $0x1,-0x4(%rbp)
  68:   83 7d fc 01             cmpl   $0x1,-0x4(%rbp)
  6c:   74 08                   je     76 <main+0x22>
  6e:   83 7d fc 02             cmpl   $0x2,-0x4(%rbp)
  72:   74 13                   je     87 <main+0x33>
  74:   eb 22                   jmp    98 <main+0x44>
  76:   48 8d 05 00 00 00 00    lea    0x0(%rip),%rax        # 7d <main+0x29>
  7d:   48 89 c1                mov    %rax,%rcx
  80:   e8 7b ff ff ff          call   0 <printf>
  85:   eb 20                   jmp    a7 <main+0x53>
  87:   48 8d 05 0f 00 00 00    lea    0xf(%rip),%rax        # 9d <main+0x49>
  8e:   48 89 c1                mov    %rax,%rcx
  91:   e8 6a ff ff ff          call   0 <printf>
  96:   eb 0f                   jmp    a7 <main+0x53>
  98:   48 8d 05 20 00 00 00    lea    0x20(%rip),%rax        # bf <main+0x6b>
  9f:   48 89 c1                mov    %rax,%rcx
  a2:   e8 59 ff ff ff          call   0 <printf>
  a7:   b8 00 00 00 00          mov    $0x0,%eax
  ac:   48 83 c4 30             add    $0x30,%rsp
  b0:   5d                      pop    %rbp
  b1:   c3                      ret
  b2:   90                      nop
  b3:   90                      nop
```

cmpl $0x1, -0x4(%rbp): compare -0x4(%rbp) (variable) with 1
je 87 <main+0x33>: if equal to 1, jump to case 1
cmpl $0x2, -0x4(%rbp): compare -0x4(%rbp) with 2
je 97 <main+0x43>: if equal to 2, jump to case 2
jmp a7 <main+0x53>: if none, jump to default case
Case 1
lea 0x0(%rip), %rax: load address for printf argument (case 1)
mov %rax, %rcx: move argument to %rcx
call 0 <printf>: call printf for case 1
jmp a7 <main+0x53>: jump to end
Case 2
lea 0xf(%rip), %rax: load address for printf argument (case 2)
mov %rax, %rcx: move argument to %rcx
call 0 <printf>: call printf for case 2
jmp a7 <main+0x53>: jump to end
Default case
lea 0x20(%rip), %rax: load address for printf argument (default case)
mov %rax, %rcx: move argument to %rcx
call 0 <printf>: call printf for default case

Construct 16: union

```c
union Data {
    int i;
    float f;
    char c;
};

int main() {
    union Data data;
    data.i = 10;
    printf("data.i = %d\n", data.i);
    data.f = 3.14;
    printf("data.f = %f\n", data.f);
    data.c = 'A';
    printf("data.c = %c\n", data.c);

    return 0;
}
```

```
58:   48 83 ec 30             sub    $0x30,%rsp
5c:   e8 00 00 00 00          call   61 <main+0xd>
61:   c7 45 fc 01 00 00 00    movl   $0x1,-0x4(%rbp)
68:   83 7d fc 01             cmpl   $0x1,-0x4(%rbp)
6c:   74 08                   je     76 <main+0x22>
6e:   83 7d fc 02             cmpl   $0x2,-0x4(%rbp)
72:   74 13                   je     87 <main+0x33>
74:   eb 22                   jmp    98 <main+0x44>
76:   48 8d 05 00 00 00 00    lea    0x0(%rip),%rax        # 7d <main+0x29>
7d:   48 89 c1                mov    %rax,%rcx
80:   e8 7b ff ff ff          call   0 <printf>
7d:   48 89 c1                mov    %rax,%rcx
80:   e8 7b ff ff ff          call   0 <printf>
85:   eb 20                   jmp    a7 <main+0x53>
87:   48 8d 05 0f 00 00 00    lea    0xf(%rip),%rax        # 9d <main+0x49>
8e:   48 89 c1                mov    %rax,%rcx
91:   e8 6a ff ff ff          call   0 <printf>
96:   eb 0f                   jmp    a7 <main+0x53>
98:   48 8d 05 20 00 00 00    lea    0x20(%rip),%rax        # bf <main+0x6b>
9f:   48 89 c1                mov    %rax,%rcx
7d:   48 89 c1                mov    %rax,%rcx
80:   e8 7b ff ff ff          call   0 <printf>
85:   eb 20                   jmp    a7 <main+0x53>
87:   48 8d 05 0f 00 00 00    lea    0xf(%rip),%rax        # 9d <main+0x49>
8e:   48 89 c1                mov    %rax,%rcx
91:   e8 6a ff ff ff          call   0 <printf>
96:   eb 0f                   jmp    a7 <main+0x53>
7d:   48 89 c1                mov    %rax,%rcx
80:   e8 7b ff ff ff          call   0 <printf>
85:   eb 20                   jmp    a7 <main+0x53>
87:   48 8d 05 0f 00 00 00    lea    0xf(%rip),%rax        # 9d <main+0x49>
8e:   48 89 c1                mov    %rax,%rcx
91:   e8 6a ff ff ff          call   0 <printf>
7d:   48 89 c1                mov    %rax,%rcx
80:   e8 7b ff ff ff          call   0 <printf>
85:   eb 20                   jmp    a7 <main+0x53>
7d:   48 89 c1                mov    %rax,%rcx
80:   e8 7b ff ff ff          call   0 <printf>
7d:   48 89 c1                mov    %rax,%rcx
80:   e8 7b ff ff ff          call   0 <printf>
85:   eb 20                   jmp    a7 <main+0x53>
87:   48 8d 05 0f 00 00 00    lea    0xf(%rip),%rax        # 9d <main+0x49>
8e:   48 89 c1                mov    %rax,%rcx
91:   e8 6a ff ff ff          call   0 <printf>
96:   eb 0f                   jmp    a7 <main+0x53>
98:   48 8d 05 20 00 00 00    lea    0x20(%rip),%rax        # bf <main+0x6b>
9f:   48 89 c1                mov    %rax,%rcx
a2:   e8 59 ff ff ff          call   0 <printf>
a7:   b8 00 00 00 00          mov    $0x0,%eax
ac:   48 83 c4 30             add    $0x30,%rsp
b0:   5d                      pop    %rbp
```

sub $0x30, %rsp: allocate space on stack
Union initialization
movl $0x1, -0x4(%rbp): initialize union field with value 1

cmpl $0x1, -0x4(%rbp): compare value in union field with 1
je 74 <main+0x20>: jump to case where union stores value 1
cmpl $0x2, -0x4(%rbp): compare value in union field with 2
je 7c <main+0x28>: jump to case where union stores value 2
jmp 96 <main+0x48>: jump to default case if union holds another value
Case 1
lea 0x0(%rip), %rax: load address of printf argument
mov %rax, %rcx: move argument for printf
call 0 <printf>: call printf for case 1
jmp 96 <main+0x48>: jump to end
Case 2
lea 0xf(%rip), %rax: load address of printf argument
mov %rax, %rcx: move argument for printf
call 0 <printf>: call printf for case 2
jmp 96 <main+0x48>: jump to end
Default case
lea 0x20(%rip), %rax: load address of printf argument
mov %rax, %rcx: move argument for printf
call 0 <printf>: call printf for default case

mov $0x0, %eax: set return value to 0
add $0x30, %rsp: restore stack pointer
pop %rbp: restore base pointer
ret: return from the function

Construct 17: static variable

```c
void counter() {
    static int count = 0;
    count++;
    printf("Function called %d times\n", count);
}

int main() {
    counter();
    counter();
    counter();
    return 0;
}
```

```
5c:   8b 05 00 00 00 00       mov    0x0(%rip),%eax        # 62 <counter+0xe>
62:   83 c0 01                add    $0x1,%eax
65:   89 05 00 00 00 00       mov    %eax,0x0(%rip)        # 6b <counter+0x17>
```

mov 0x0(%rip), %eax: load static variable from memory into %eax
add $0x1, %eax: increment static variable by 1
mov %eax, 0x0(%rip): store updated static variable back into memory

Construct 18: writing to a file

```c
int main() {
    FILE *fp = fopen("test.txt", "w");
    if (fp == NULL) return -1;
    fprintf(fp, "Hello, World!\n");
    fclose(fp);
    return 0;
}
```

```
7b:   48 8b 45 f8             mov    -0x8(%rbp),%rax
7f:   48 8d 15 0b 00 00 00    lea    0xb(%rip),%rdx        # 91 <main+0x4e>
86:   48 89 c1                mov    %rax,%rcx
89:   e8 72 ff ff ff          call   0 <fprintf>
```

```
0000000000000000 <fprintf>:
   0:   55                      push   %rbp
   1:   48 89 e5                mov    %rsp,%rbp
   4:   48 83 ec 30             sub    $0x30,%rsp
   8:   48 89 4d 10             mov    %rcx,0x10(%rbp)
   c:   48 89 55 18             mov    %rdx,0x18(%rbp)
  10:   4c 89 45 20             mov    %r8,0x20(%rbp)
  14:   4c 89 4d 28             mov    %r9,0x28(%rbp)
  18:   48 8d 45 20             lea    0x20(%rbp),%rax
  1c:   48 89 45 f0             mov    %rax,-0x10(%rbp)
  20:   48 8b 4d f0             mov    -0x10(%rbp),%rcx
  24:   48 8b 55 18             mov    0x18(%rbp),%rdx
  28:   48 8b 45 10             mov    0x10(%rbp),%rax
  2c:   49 89 c8                mov    %rcx,%r8
  2f:   48 89 c1                mov    %rax,%rcx
  32:   e8 00 00 00 00          call   37 <fprintf+0x37>
  37:   89 45 fc                mov    %eax,-0x4(%rbp)
  3a:   8b 45 fc                mov    -0x4(%rbp),%eax
  3d:   48 83 c4 30             add    $0x30,%rsp
  41:   5d                      pop    %rbp
  42:   c3                      ret
```

mov -0x8(%rbp), %rdx: load the data to be written from memory (at -0x8(%rbp)) into %rdx
lea 0x2(%rip), %rax: load the address (possibly file pointer or buffer location) into %rax
mov %rax, %rcx: move the address of the file or buffer to %rcx (for the writing operation)
call 69 <main+0x26>: call the function responsible for file writing

fprintf()
push %rbp: save base pointer
mov %rsp, %rbp: set up stack frame
sub $0x30, %rsp: allocate space on stack (48 bytes)

mov %rcx, 0x10(%rbp): move argument into local variable
mov %rdx, 0x18(%rbp): move argument into local variable
lea 0x20(%rbp), %rax: load address into %rax
mov %rax, -0x10(%rbp): store address into local variable
mov 0x10(%rbp), %rcx: move format string back into %rcx
mov 0x18(%rbp), %rdx: move second argument back into %rdx
mov %rax, %rcx: prepare format string for fprintf
call 37 <fprintf+0x37>: call to do write operation
mov %eax, -0x4(%rbp): store return value of fprintf
add $0x30, %rsp: restore stack pointer
pop %rbp: restore base pointer
ret: return from the function

Construct 19: typedef

```c
typedef int MyInt;

int main() {
    MyInt num = 13;
```

```
58:   48 83 ec 30           sub    $0x30,%rsp
5c:   e8 00 00 00 00        call   61 <main+0xd>
61:   c7 45 fc 0d 00 00 00  movl   $0xd,-0x4(%rbp)
68:   8b 45 fc              mov    -0x4(%rbp),%eax
6b:   89 c2                 mov    %eax,%edx
6d:   48 8d 05 00 00 00 00  lea    0x0(%rip),%rax        # 74 <main+0x20>
74:   48 89 c1              mov    %rax,%rcx
```

sub $0x30, %rsp: allocate space on the stack
movl $0xd, -0x4(%rbp): store value 13 into the location reserved for typedeft
mov -0x4(%rbp), %eax: move the value stored in the typedef to %eax
mov %eax, %edx: move value in %eax into %edx
lea 0x0(%rip), %rax: load the address (likely a format string or file pointer)
mov %rax, %rcx: move the format string or file pointer into %rcx (for printf)

Construct 20: break

```c
for (int i = 0; i < 5; i++) {
    if (i == 3) {
        break;
    }
```

```
6a:   83 7d fc 03           cmpl   $0x3,-0x4(%rbp)
6e:   74 20                 je     90 <main+0x3c>
```

cmpl $0x3, -0x4(%rbp): compare variable with 3
je 90 <main+0x3c>: if equal to 3 jump to end of loop (break)

My_prorgam.c

```c
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

typedef int ElementType;

struct SortStats {
    int comparisons;
    int swaps;
};

// Function prototypes
void insertionSort(ElementType arr[], int n, struct SortStats *stats);
void quickSort(ElementType arr[], int low, int high, struct SortStats *stats);
int partition(ElementType arr[], int low, int high, struct SortStats *stats);
void printArray(ElementType arr[], int size);
void writeStatsToFile(struct SortStats stats, const char *algorithm);
void swap(ElementType *a, ElementType *b, struct SortStats *stats);

int main() {
    // Variable declaration
    int choice;

    // Array
    ElementType data[] = {64, 34, 25, 12, 22, 11, 90};
    int n = sizeof(data) / sizeof(data[0]);

    // Initialize structure to track stats
    struct SortStats stats = {0, 0};

    printf("Original array:\n");
    printArray(data, n);

    // User input
    printf("Array: {64, 34, 25, 12, 22, 11, 90}\n");
    printf("Choose a sorting algorithm:\n");
    printf("1. Quick Sort\n");
    printf("2. Insertion Sort\n");
    printf("Enter your choice (1-2): ");
    scanf("%d", &choice);

    // switch for pickign sorting algorithm
    switch (choice) {
        case 1:
            quickSort(data, 0, n - 1, &stats);
            writeStatsToFile(stats, "Quick Sort");
            break;
```

```
        case 2:
            insertionSort(data, n, &stats);
            writeStatsToFile(stats, "Insertion Sort");
            break;
        default:
            printf("Invalid choice.\n");
            return 0;
    }

    printf("Sorted array:\n");
    printArray(data, n);
    return 0;
}

void insertionSort(ElementType arr[], int n, struct SortStats *stats) {
    int i, j;
    ElementType key;
    for (i = 1; i < n; i++) {
        key = arr[i];
        j = i - 1;
        while (j >= 0 && arr[j] > key) {
            stats->comparisons++;
            arr[j + 1] = arr[j];
            stats->swaps++;
            j--;
        }
        arr[j + 1] = key;
    }
}

// Partition function for quicksort
int partition(ElementType arr[], int low, int high, struct SortStats *stats) {
    ElementType pivot = arr[high];
    int i = (low - 1);
    for (int j = low; j < high; j++) {
        stats->comparisons++;
        if (arr[j] < pivot) {
            i++;
            swap(&arr[i], &arr[j], stats);
        }
    }
    swap(&arr[i + 1], &arr[high], stats);
    return (i + 1);
}

void quickSort(ElementType arr[], int low, int high, struct SortStats *stats) {
    if (low < high) {
        int pi = partition(arr, low, high, stats);
        quickSort(arr, low, pi - 1, stats);
```

```
        quickSort(arr, pi + 1, high, stats);
    }
}

void swap(ElementType *a, ElementType *b, struct SortStats *stats) {
    ElementType temp = *a;
    *a = *b;
    *b = temp;
    stats->swaps++;
}

void printArray(ElementType arr[], int size) {
    for (int i = 0; i < size; i++)
        printf("%d ", arr[i]);
    printf("\n");
}
void writeStatsToFile(struct SortStats stats, const char *algorithm) {
    FILE *fp = fopen("sort_stats.txt", "a");
    if (fp != NULL) {
        fprintf(fp, "%s:\n", algorithm);
        fprintf(fp, "Comparisons: %d\n", stats.comparisons);
        fprintf(fp, "Swaps: %d\n\n", stats.swaps);
        fclose(fp);
        printf("Sorting statistics written to file.\n");
    } else {
        printf("Failed to open file for writing.\n");
    }
}
```

Assembly Code: code in zip file too

```
.file   "my_program.c"

.text

.def    scanf   .scl    3   .type   32  .endef
.seh_proc   scanf
scanf:
    pushq   %rbp                # Save the base pointer
    .seh_pushreg    %rbp
    pushq   %rbx                # Save rbx register
    .seh_pushreg    %rbx
    subq    $56, %rsp           # Allocate 56 bytes on  stack for local variables
    .seh_stackalloc 56
    leaq    48(%rsp), %rbp      # Set up  base pointer to 48 bytes from the stack pointer
    .seh_setframe   %rbp, 48
    .seh_endprologue
    movq    %rcx, 32(%rbp)      # Store first argument (rcx) in a local variable
```

```
    movq    %rdx, 40(%rbp)      # Store second argument (rdx) in a local variable
    movq    %r8, 48(%rbp)       # Store third argument (r8) in a local variable
    movq    %r9, 56(%rbp)       # Store fourth argument (r9) in a local variable
    leaq    40(%rbp), %rax      # Load  address of a local variable into rax
    movq    %rax, -16(%rbp)     # Store  address in another local variable
    movq    -16(%rbp), %rbx     # Load  stored address into rbx
    movl    $0, %ecx            # Set ecx to 0
    movq    __imp___acrt_iob_func(%rip), %rax
    call    *%rax
    movq    %rax, %rcx          # Move result into rcx
    movq    32(%rbp), %rax      # Load stored argument from local variable
    movq    %rbx, %r8           # Move rbx into r8 for argument passing
    movq    %rax, %rdx          # Move rax into rdx for argument passing
    call    __mingw_vfscanf
    movl    %eax, -4(%rbp)      # Store return value in a local variable
    movl    -4(%rbp), %eax      # Move return value back into eax
    addq    $56, %rsp           # Deallocate stack space
    popq    %rbx                # Restore rbx register
    popq    %rbp                # Restore base pointer
    ret                         # Return from function
.seh_endproc


.def    fprintf .scl    3   .type   32  .endef
.seh_proc   fprintf
fprintf:
    pushq   %rbp                # Save base pointer
    .seh_pushreg    %rbp        # Record base pointer in SEH
    movq    %rsp, %rbp          # Set new base pointer
    .seh_setframe   %rbp, 0     # Record base pointer frame in SEH
    subq    $48, %rsp           # Allocate 48 bytes on the stack
    .seh_stackalloc 48
    .seh_endprologue
    movq    %rcx, 16(%rbp)      # Store first argument (rcx) in a local variable
    movq    %rdx, 24(%rbp)      # Store second argument (rdx) in a local variable
    movq    %r8, 32(%rbp)       # Store third argument (r8) in a local variable
    movq    %r9, 40(%rbp)       # Store fourth argument (r9) in a local variable
    leaq    32(%rbp), %rax      # Load address of a local variable into rax
    movq    %rax, -16(%rbp)     # Store address in another local variable
    movq    -16(%rbp), %rcx     # Load stored address into rcx
    movq    24(%rbp), %rdx      # Load second argument back into rdx
    movq    16(%rbp), %rax      # Load first argument back into rax
    movq    %rcx, %r8           # Move rcx into r8 for argument passing
    movq    %rax, %rcx          # Move rax into rcx for argument passing
    call    __mingw_vfprintf
    movl    %eax, -4(%rbp)      # Store return value in a local variable
    movl    -4(%rbp), %eax      # Move return value back into eax
    addq    $48, %rsp           # Deallocate  stack space
    popq    %rbp                # Restore base pointer
```

```
    ret                         # Return from the function
.seh_endproc


.def    printf  .scl    3   .type   32  .endef
.seh_proc   printf
printf:
    pushq   %rbp                # Save base pointer
    .seh_pushreg    %rbp
    pushq   %rbx                # Save rbx register
    .seh_pushreg    %rbx        # Record rbx in SEH
    subq    $56, %rsp           # Allocate 56 bytes on the stack
    .seh_stackalloc 56
    leaq    48(%rsp), %rbp      # Set up base pointer
    .seh_setframe   %rbp, 48
    .seh_endprologue
    movq    %rcx, 32(%rbp)      # Store first argument (rcx) in a local variable
    movq    %rdx, 40(%rbp)      # Store second argument (rdx) in a local variable
    movq    %r8, 48(%rbp)       # Store third argument (r8) in a local variable
    movq    %r9, 56(%rbp)       # Store fourth argument (r9) in a local variable
    leaq    40(%rbp), %rax      # Load address of a local variable into rax
    movq    %rax, -16(%rbp)     # Store address in another local variable
    movq    -16(%rbp), %rbx     # Load stored address into rbx
    movl    $1, %ecx            # Set ecx to 1
    movq    __imp___acrt_iob_func(%rip), %rax
    call    *%rax
    movq    %rax, %rcx          # Move result into rcx
    movq    32(%rbp), %rax      # Load stored argument from local variable
    movq    %rbx, %r8           # Move rbx into r8 for argument passing
    movq    %rax, %rdx          # Move rax into rdx for argument passing
    call    __mingw_vfprintf    # Call printf implementation (vfprintf)
    movl    %eax, -4(%rbp)      # Store return value in a local variable
    movl    -4(%rbp), %eax      # Move return value back into eax
    addq    $56, %rsp           # Deallocate stack space
    popq    %rbx                # Restore rbx register
    popq    %rbp                # Restore base pointer
    ret                         # Return from the function
.seh_endproc
    .def    __main  .scl    2   .type   32  .endef
    .section .rdata,"dr"
.LC0:
    .ascii "Original array:\12\0"
    .align 8
.LC1:
    .ascii "Array: {64, 34, 25, 12, 22, 11, 90}\12\0"
.LC2:
    .ascii "Choose a sorting algorithm:\12\0"
.LC3:
    .ascii "1. Quick Sort\12\0"
```

```
.LC4:
    .ascii "2. Insertion Sort\12\0"
.LC5:
    .ascii "Enter your choice (1-2): \0"
.LC6:
    .ascii "%d\0"
.LC7:
    .ascii "Quick Sort\0"
.LC8:
    .ascii "Insertion Sort\0"
.LC9:
    .ascii "Invalid choice.\12\0"
.LC10:
    .ascii "Sorted array:\12\0"
.text
.globl  main
.def    main    .scl    2   .type   32  .endef
.seh_proc   main
main:
    pushq   %rbp                # Save base pointer
    .seh_pushreg    %rbp
    movq    %rsp, %rbp          # Set new base pointer
    .seh_setframe   %rbp, 0
    subq    $96, %rsp           # Allocate 96 bytes on the stack
    .seh_stackalloc 96
    .seh_endprologue
    call    __main
    movl    $64, -48(%rbp)      # Initialize  array with first element 64
    movl    $34, -44(%rbp)      # Initialize array with second element 34
    movl    $25, -40(%rbp)      # Initialize array with third element 25
    movl    $12, -36(%rbp)      # Initialize array with fourth element 12
    movl    $22, -32(%rbp)      # Initialize array with fifth element 22
    movl    $11, -28(%rbp)      # Initialize array with sixth element 11
    movl    $90, -24(%rbp)      # Initialize array with seventh element 90
    movl    $7, -4(%rbp)        # Set number of elements in array to 7
    movl    $0, -56(%rbp)       # Initialize stats (comparisons) to 0
    movl    $0, -52(%rbp)       # Initialize stats (swaps) to 0
    leaq    .LC0(%rip), %rax    # Load Original array: message
    movq    %rax, %rcx          # Move it into rcx for argument passing
    call    printf              # Call printf to print message
    movl    -4(%rbp), %edx      # Load array size into edx
    leaq    -48(%rbp), %rax     # Load address of the array into rax
    movq    %rax, %rcx          # Move array address into rcx
    call    printArray          # Call printArray to print original array
    leaq    .LC1(%rip), %rax    # Load Choose a sorting algorithm: message
    movq    %rax, %rcx          # Move it into rcx for argument passing
    call    printf              # Call printf to print message
    leaq    .LC2(%rip), %rax    # Load 1. Quick Sort message
    movq    %rax, %rcx          # Move it into rcx for argument passing
```

```
    call    printf                  # Call printf to print option
    leaq    .LC3(%rip), %rax        # Load 2. Insertion Sort message
    movq    %rax, %rcx              # Move it into rcx for argument passing
    call    printf                  # Call printf to print the option
    leaq    .LC4(%rip), %rax        # Load Enter your choice message
    movq    %rax, %rcx              # Move it into rcx for argument passing
    call    printf                  # Call printf to prompt user input
    leaq    -8(%rbp), %rax          # Load address to store user input
    movq    %rax, %rdx              # Move address into rdx
    leaq    .LC5(%rip), %rax        # Load format string "%d"
    movq    %rax, %rcx              # Move it into rcx for argument passing
    call    scanf                   # Call scanf to get user input
    movl    -8(%rbp), %eax          # Load user input into eax
    cmpl    $1, %eax                # Compare input with 1
    je  .L8                         # Jump to quick sort if input is 1
    cmpl    $2, %eax                # Compare input with 2
    je  .L9                         # Jump to insertion sort if input is 2
    jmp .L14                        # Jump to invalid choice

.L8:                                # Quick Sort block
    movl    -4(%rbp), %eax          # Load array size into eax
    leal    -1(%rax), %ecx          # Subtract 1 from array size
    leaq    -56(%rbp), %rdx         # Load address for stats into rdx
    leaq    -48(%rbp), %rax         # Load address of the array into rax
    movq    %rdx, %r9               # Move stats address into r9
    movl    %ecx, %r8d              # Move modified size into r8d
    movl    $0, %edx                # Set low index (0) into edx
    movq    %rax, %rcx              # Move array address into rcx
    call    quickSort               # Call quickSort function
    movq    -56(%rbp), %rax         # Load stats into rax
    leaq    .LC6(%rip), %rdx        # Load Quick Sort string
    movq    %rax, %rcx              # Move stats address into rcx
    call    writeStatsToFile        # Call writeStatsToFile
    jmp .L11                        # Jump to the next block

.L9:                                # Insertion Sort block
    leaq    -56(%rbp), %rcx         # Load stats address into rcx
    movl    -4(%rbp), %edx          # Load array size into edx
    leaq    -48(%rbp), %rax         # Load array address into rax
    movq    %rcx, %r8               # Move stats address into r8
    movq    %rax, %rcx              # Move array address into rcx
    call    insertionSort           # Call insertionSort function
    movq    -56(%rbp), %rax         # Load stats into rax
    leaq    .LC7(%rip), %rdx        # Load Insertion Sort string
    movq    %rax, %rcx              # Move stats address into rcx
    call    writeStatsToFile        # Call writeStatsToFile
    jmp .L11                        # Jump to the next block

.L14:                               # Invalid choice block
```

```
    leaq    .LC8(%rip), %rax      # Load Invalid choice string
    movq    %rax, %rcx            # Move it into rcx
    call    printf                # Call printf to display message
    movl    $0, %eax              # Set return value to 0
    jmp .L13                      # Jump to the end

.L11:                             # After sorting, print the sorted array
    leaq    .LC9(%rip), %rax      # Load "Sorted array" string
    movq    %rax, %rcx            # Move it into rcx
    call    printf                # Call printf to print message
    movl    -4(%rbp), %edx        # Load array size into edx
    leaq    -48(%rbp), %rax       # Load array address into rax
    movq    %rax, %rcx            # Move array address into rcx
    call    printArray            # Call printArray to print sorted array
    movl    $0, %eax              # Set return value to 0

.L13:                             # Cleanup and return
    addq    $96, %rsp             # Deallocate the stack space
    popq    %rbp                  # Restore base pointer
    ret                           # Return from main
.seh_endproc

.globl  insertionSort
.def    insertionSort   .scl    2   .type   32  .endef
.seh_proc   insertionSort
insertionSort:
    pushq   %rbp                    # Save base pointer
    .seh_pushreg    %rbp
    movq    %rsp, %rbp              # Set base pointer to stack pointer
    .seh_setframe   %rbp, 0
    subq    $16, %rsp               # Allocate 16 bytes on stack for local variables
    .seh_stackalloc 16
    .seh_endprologue
    movq    %rcx, 16(%rbp)          # Store array address in local variable
    movl    %edx, 24(%rbp)          # Store array size in local variable
    movq    %r8, 32(%rbp)           # Store stats pointer in local variable
    movl    $1, -4(%rbp)            # Initialize loop variable i to 1

    jmp .L16                        # Jump to start of loop

.L20:                               # Loop
    movl    -4(%rbp), %eax          # Load i into eax
    cltq
    leaq    0(,%rax,4), %rdx        # Compute array index i * 4 (b/c int array)
    movq    16(%rbp), %rax          # Load array base address
    addq    %rdx, %rax              # Add index to base address
    movl    (%rax), %eax            # Load arr[i] into eax
    movl    %eax, -12(%rbp)         # Store key = arr[i]
    movl    -4(%rbp), %eax          # Load i into eax
```

```
    subl    $1, %eax                    # j = i - 1
    movl    %eax, -8(%rbp)              # Store j

    jmp .L17                            # inner loop check

.L19:                                   # Inner loop body
    movq    32(%rbp), %rax              # Load stats pointer
    movl    (%rax), %eax                # Load stats->comparisons
    leal    1(%eax), %edx               # comparisons++
    movq    32(%rbp), %rax              # Load stats pointer
    movl    %edx, (%rax)                # Store stats->comparisons
    movl    -8(%rbp), %eax              # Load j into eax
    cltq
    leaq    0(,%rax,4), %rdx            # Compute array index j * 4
    movq    16(%rbp), %rax              # Load array base address
    addq    %rdx, %rax                  # Add index to base address
    movl    (%rax), %eax                # Load arr[j] into eax
    movl    %eax, -8(%rbp)              # Store arr[j+1] = arr[j]
    subl    $1, -8(%rbp)                # j--

.L17:                                   # inner loop check
    cmpl    $0, -8(%rbp)                # Compare j with 0
    js  .L18                            # If j < 0, exit loop
    movl    -8(%rbp), %eax              # Load j
    cltq
    leaq    0(,%rax,4), %rdx            # Compute array index j * 4
    movq    16(%rbp), %rax              # Load array base address
    addq    %rdx, %rax                  # Add index to base address
    movl    (%rax), %eax                # Load arr[j]
    cmpl    %eax, -12(%rbp)             # Compare arr[j] with key
    jl  .L19                            # If arr[j] > key, continue inner loop

.L18:                                   # Store key in arr[j+1]
    movl    -8(%rbp), %eax              # Load j
    cltq
    addq    $1, %rax                    # j + 1
    leaq    0(,%rax,4), %rdx            # Compute arr[j+1]
    movq    16(%rbp), %rax              # Load array base address
    addq    %rax, %rdx                  # Add index to base address
    movl    -12(%rbp), %eax             # Load key
    movl    %eax, (%rdx)                # Store key in arr[j+1]
    addl    $1, -4(%rbp)                # i++

.L16:                                   # Loop check
    movl    -4(%rbp), %eax              # Load i
    cmpl    24(%rbp), %eax              # Compare i with n
    jl  .L20                            # If i < n, continue loop

    nop
```

```
    addq    $16, %rsp                   # Deallocate stack space
    popq    %rbp                        # Restore base pointer
    ret                                 # Return
.seh_endproc

.globl  partition
.def    partition   .scl    2   .type   32  .endef
.seh_proc   partition
partition:
    pushq   %rbp                        # Save base pointer
    .seh_pushreg    %rbp
    movq    %rsp, %rbp                  # Set base pointer to stack pointer
    .seh_setframe   %rbp, 0
    subq    $48, %rsp                   # Allocate 48 bytes on stack for local variables
    .seh_stackalloc 48
    .seh_endprologue
    movq    %rcx, 16(%rbp)              # Store array address in local variable
    movl    %edx, 24(%rbp)              # Store low index in local variable
    movl    %r8d, 32(%rbp)              # Store high index in local variable
    movq    %r9, 40(%rbp)               # Store stats pointer in local variable
    movl    32(%rbp), %eax              # Load arr[high] (pivot)
    cltq
    leaq    0(,%rax,4), %rdx            # Compute index high * 4
    movq    16(%rbp), %rax              # Load array base address
    addq    %rdx, %rax                  # Add index to base address
    movl    (%rax), %eax                # Load arr[high] into eax
    movl    %eax, -12(%rbp)             # Store pivot value
    movl    24(%rbp), %eax              # Load low index
    subl    $1, %eax                    # low - 1 (index of smaller element)
    movl    %eax, -4(%rbp)              # Store i (initial value)
    movl    24(%rbp), %eax              # Load low index
    movl    %eax, -8(%rbp)              # Initialize j to low

    jmp .L22                            # Jump to loop start

.L24:                                   # Loop
    movq    40(%rbp), %rax              # Load stats pointer
    movl    (%rax), %eax                # Load stats->comparisons
    leal    1(%eax), %edx               # comparisons++
    movq    40(%rbp), %rax              # Load stats pointer
    movl    %edx, (%rax)                # Store updated stats->comparisons
    movl    -8(%rbp), %eax              # Load j
    cltq
    leaq    0(,%rax,4), %rdx            # Compute array index j * 4
    movq    16(%rbp), %rax              # Load array base address
    addq    %rdx, %rax                  # Add index to base address
    movl    (%rax), %eax                # Load arr[j]
    cmpl    %eax, -12(%rbp)             # Compare arr[j] with pivot
    jle .L23                            # If arr[j] <= pivot, jump to L23
```

```
        addl    $1, -4(%rbp)            # i++
    movl    -8(%rbp), %eax              # Load j into eax
    cltq
    leaq    0(,%rax,4), %rdx            # Compute array index j * 4
    movq    16(%rbp), %rax              # Load base address of the array
    addq    %rdx, %rax                  # Add index to base address
    movl    (%rax), %eax                # Load arr[j]
    cmpl    %eax, -12(%rbp)             # Compare arr[j] with pivot
    jle .L23                            # If arr[j] <= pivot, continue
    addl    $1, -4(%rbp)                # i++

    movl    -4(%rbp), %eax              # Load i into eax
    cltq
    leaq    0(,%rax,4), %rcx            # Compute arr[i] index
    movq    16(%rbp), %rax              # Load array base address
    addq    %rax, %rcx                  # Add index to base address
    movq    40(%rbp), %r8               # Load address of stats into r8
    call    swap                        # Call swap to swap arr[i] and arr[j]
.L23:
    addl    $1, -8(%rbp)                # j++

.L22:
    movl    -8(%rbp), %eax              # Load j into eax
    cmpl    32(%rbp), %eax              # Compare j with high (index)
    jl  .L24                            # If j < high, repeat loop

    movl    32(%rbp), %eax              # Load high index into eax
    cltq
    leaq    0(,%rax,4), %rdx            # high * 4
    movq    16(%rbp), %rax              # Load array base address
    addq    %rax, %rdx                  # Add high index to base address
    movl    -4(%rbp), %eax              # Load i into eax
    cltq
    addq    $1, %rax                    # i++
    leaq    0(,%rax,4), %rcx            # Compute arr[i+1] address
    movq    16(%rbp), %rax              # Load array base address
    addq    %rax, %rcx                  # Add index to base address
    movq    40(%rbp), %r8               # Load stats pointer into r8
    call    swap                        # Call swap to swap arr[i] and arr[high]

    movl    -4(%rbp), %eax              # Load i
    addl    $1, %eax                    # i++
    addq    $48, %rsp                   # Deallocate stack space
    popq    %rbp                        # Restore base pointer
    ret                                 # Return from partition
.seh_endproc

    .seh_endproc
.globl  quickSort
```

```
.def    quickSort   .scl    2   .type   32  .endef
.seh_proc   quickSort
quickSort:
    pushq   %rbp                        # Save base pointer
    .seh_pushreg    %rbp
    movq    %rsp, %rbp                  # Set base pointer to stack pointer
    .seh_setframe   %rbp, 0
    subq    $48, %rsp                   # Allocate 48 bytes on stack
    .seh_stackalloc 48
    .seh_endprologue

    movq    %rcx, 16(%rbp)              # Store array address
    movl    %edx, 24(%rbp)              # Store low index
    movl    %r8d, 32(%rbp)              # Store high index
    movq    %r9, 40(%rbp)               # Store stats pointer

    movl    24(%rbp), %eax              # Load low index
    cmpl    32(%rbp), %eax              # Compare low with high
    jge .L28                            # If low >= high, exit

    movq    40(%rbp), %r8               # Load stats pointer into r8
    movl    32(%rbp), %ecx              # Load high into ecx
    movl    24(%rbp), %edx              # Load low into edx
    movq    16(%rbp), %rax              # Load array base address
    movq    %r8, %r9                    # Store stats in r9 for the partition call
    movl    %ecx, %r8d                  # Store high into r8d
    movq    %rax, %rcx                  # Store array into rcx for the partition call
    call    partition                   # Call partition function

    movl    %eax, -4(%rbp)              # Store partition index into local variable
    movl    -4(%rbp), %eax              # Load partition index
    leal    -1(%eax), %r8d              # partition - 1
    movq    40(%rbp), %rcx              # Load stats pointer into rcx
    movl    24(%rbp), %edx              # Load low into edx
    movq    16(%rbp), %rax              # Load array into rax
    movq    %rcx, %r9                   # Store stats pointer into r9
    movq    %rax, %rcx                  # Store array into rcx
    call    quickSort                   # Recursive call to quickSort (low, partition - 1)

    movl    -4(%rbp), %eax              # Load partition index
    leal    1(%eax), %edx               # partition + 1
    movq    40(%rbp), %r8               # Load stats pointer into r8
    movl    32(%rbp), %ecx              # Load high into ecx
    movq    16(%rbp), %rax              # Load array into rax
    movq    %r8, %r9                    # Store stats pointer into r9
    movq    %rax, %rcx                  # Store array into rcx
    call    quickSort                   # Recursive call to quickSort (partition + 1, high)

.L28:
```

```
    nop
    addq    $48, %rsp                   # Deallocate stack space
    popq    %rbp                        # Restore base pointer
    ret                                 # Return
.seh_endproc

.globl  swap
.def    swap    .scl    2   .type   32  .endef
.seh_proc   swap
swap:
    pushq   %rbp                        # Save base pointer
    .seh_pushreg    %rbp
    movq    %rsp, %rbp                  # Set base pointer to stack pointer
    .seh_setframe   %rbp, 0
    subq    $16, %rsp                   # Allocate 16 bytes on stack
    .seh_stackalloc 16
    .seh_endprologue
    movq    %rcx, 16(%rbp)              # Store address of first element
    movq    %rdx, 24(%rbp)              # Store address of second element
    movq    %r8, 32(%rbp)               # Store stats pointer
    movq    16(%rbp), %rax              # Load first element's address into rax
    movl    (%rax), %eax                # Load the first element's value
    movl    %eax, -4(%rbp)              # Store the value in local variable temp
    movq    24(%rbp), %rax              # Load second element's address into rax
    movl    (%rax), %edx                # Load the second element's value into edx
    movq    16(%rbp), %rax              # Load first element's address into rax
    movl    %edx, (%rax)                # Store the second element's value in the first
element's location
    movq    24(%rbp), %rax              # Load second element's address into rax
    movl    -4(%rbp), %edx              # Load the value
    movl    -4(%rbp), %edx              # Load the value from temp
    movl    %edx, (%rax)                # Store temp in the second element's location
    movq    32(%rbp), %rax              # Load stats pointer
    movl    4(%rax), %eax               # Load stats->swaps
    leal    1(%eax), %edx               # swaps++
    movq    32(%rbp), %rax              # Load stats pointer
    movl    %edx, 4(%rax)               # Store updated stats->swaps
    nop
    addq    $16, %rsp                   # Deallocate stack space
    popq    %rbp                        # Restore base pointer
    ret                                 # Return
.seh_endproc

    .section .rdata,"dr"
.LC11:
    .ascii "%d \0"
.LC12:
    .ascii "\12\0"
    .text
```

```
.globl  printArray
.def    printArray  .scl    2   .type   32  .endef
.seh_proc   printArray
    pushq   %rbp                        # Save base pointer
    .seh_pushreg    %rbp
    movq    %rsp, %rbp                  # Set base pointer to stack pointer
    .seh_setframe   %rbp, 0
    subq    $48, %rsp                   # Allocate 48 bytes on stack
    .seh_stackalloc 48
    .seh_endprologue

    movq    %rcx, 16(%rbp)              # Store array address
    movl    %edx, 24(%rbp)              # Store array size
    movl    $0, -4(%rbp)                # i = 0

    jmp .L31                            # Jump to loop start

.L32:                                   # Loop
    movl    -4(%rbp), %eax              # Load index i
    cltq
    leaq    0(,%rax,4), %rdx            # Compute index i * 4
    movq    16(%rbp), %rax              # Load array base address
    addq    %rdx, %rax                  # Add index to base address
    movl    (%rax), %eax                # Load arr[i]
    movl    %eax, %edx                  # Move arr[i] to edx (argument for printf)
    leaq    .LC11(%rip), %rax           # Load format string address (for %d)
    movq    %rax, %rcx                  # Move format string to rcx
    call    printf                      # Call printf to print arr[i]

    addl    $1, -4(%rbp)                # index++
.L31:                                   # Loop check
    movl    -4(%rbp), %eax              # Load i
    cmpl    24(%rbp), %eax              # Compare i with size
    jl  .L32                            # If i < size, repeat loop

    leaq    .LC12(%rip), %rax           # Load newline character address
    movq    %rax, %rcx                  # Move newline character to rcx
    call    printf                      # Call printf to print newline

    nop
    addq    $48, %rsp                   # Deallocate stack space
    popq    %rbp                        # Restore base pointer
    ret                                 # Return
.seh_endproc

    .section .rdata,"dr"
.LC13:
    .ascii "a\0"
.LC14:
```

```
    .ascii "sort_stats.txt\0"
.LC15:
    .ascii "%s:\12\0"
.LC16:
    .ascii "Comparisons: %d\12\0"
.LC17:
    .ascii "Swaps: %d\12\12\0"
    .align 8
.LC18:
    .ascii "Sorting statistics written to file.\12\0"
    .align 8
.LC19:
    .ascii "Failed to open file for writing.\12\0"
    .text
.globl  writeStatsToFile
.def    writeStatsToFile    .scl    2   .type   32  .endef
.seh_proc   writeStatsToFile
writeStatsToFile:
    pushq   %rbp                        # Save base pointer
    .seh_pushreg    %rbp                # Record base pointer in SEH
    movq    %rsp, %rbp                  # Set base pointer to stack pointer
    .seh_setframe   %rbp, 0
    subq    $48, %rsp                   # Allocate 48 bytes on stack
    .seh_stackalloc 48
    .seh_endprologue

    movq    %rcx, 16(%rbp)              # Store stats pointer
    movq    %rdx, 24(%rbp)              # Store algorithm name
    leaq    .LC13(%rip), %rax           # Load "a" (append mode) string address
    movq    %rax, %rdx                  # Move append mode string to rdx
    leaq    .LC14(%rip), %rax           # Load file name string address
    movq    %rax, %rcx                  # Move file name to rcx
    call    fopen                       # Call fopen to open file

    movq    %rax, -8(%rbp)              # Store file pointer
    cmpq    $0, -8(%rbp)                # Check if file pointer is NULL
    je  .L34                            # If file is NULL, jump to error handling

    movq    24(%rbp), %rdx              # Load algorithm name
    movq    -8(%rbp), %rax              # Load file pointer
    movq    %rdx, %r8                   # Move algorithm name to r8 (argument for fprintf)
    leaq    .LC15(%rip), %rdx           # Load format string ("%s:\n")
    movq    %rax, %rcx                  # Move file pointer to rcx (argument for fprintf)
    call    fprintf                     # Call fprintf to write algorithm name to file

    movl    16(%rbp), %edx              # Load stats->comparisons
    movq    -8(%rbp), %rax              # Load file pointer
    movl    %edx, %r8d                  # Move comparisons to r8d (argument for fprintf)
    leaq    .LC16(%rip), %rdx           # Load format string ("Comparisons: %d\n")
```

```
    movq    %rax, %rcx                  # Move file pointer to rcx (argument for fprintf)
    call    fprintf                     # Call fprintf to write comparisons to file

    movl    20(%rbp), %edx              # Load stats->swaps
    movq    -8(%rbp), %rax              # Load file pointer
    movl    %edx, %r8d                  # Move swaps to r8d (argument for fprintf)
    leaq    .LC17(%rip), %rdx           # Load format string ("Swaps: %d\n\n")
    movq    %rax, %rcx                  # Move file pointer to rcx (argument for fprintf)
    call    fprintf                     # Call fprintf to write swaps to file

    movq    -8(%rbp), %rax              # Load file pointer
    movq    %rax, %rcx                  # Move file pointer to rcx (argument for fclose)
    call    fclose                      # Call fclose to close the file

    leaq    .LC18(%rip), %rax           # Load "Sorting statistics written to file." string
address
    movq    %rax, %rcx                  # Move string to rcx
    call    printf                      # Call printf to display success message
    jmp .L36                            # Jump to function end

.L34:
    leaq    .LC19(%rip), %rax           # Load "Failed to open file for writing." string
address
    movq    %rax, %rcx                  # Move string to rcx
    call    printf                      # Call printf to display error message

.L36:
    nop
    addq    $48, %rsp                   # Deallocate stack space
    popq    %rbp                        # Restore base pointer
    ret                                 # Return
.seh_endproc


    .def    __mingw_vfscanf .scl    2   .type   32  .endef
    .def    __mingw_vfprintf    .scl    2   .type   32  .endef
    .def    fopen   .scl    2   .type   32  .endef
    .def    fclose  .scl    2   .type   32  .endef
```

0th level: code in zip file

0th level optimization is the level that performs basic optimization by remove any redundant code like pop/pushing registers that are not necessary and manual stack manipulation. The compiler sets this as default. The assembly code is a direct translation of C code to assembly without thinking about performance. It is typically easier to read but also inefficient because excess usage of stack space and memory and no instruction level optimizations. Also, each function has a complete setup and teardown. Finally, printf and scanf are not called with inlining or shortcuts, which leads to extra function calls that take up time.

1st level: code in zip file

1st level optimation is the level that has readability and performance in mind, by removing unnecessary instructions and using the CPU smarter, without drastically changing the code. It reduces stack allocations so printf and scanf don't take up as much space on the stack, function setup and teardown is more streamlined compared to oth level optimization, and registers are used much more efficiently.

2nd level: code in zip file

2nd level optimization is the level that is all about speed. The code is now very optimized because of techniques like reordering instructions, loop optimization, and inlining. Instructions are reordered to improve CPU pipelining, which allows instructions that don't depend on each other to be executed in parallel. In quickSort, instructions where data is being moved are reordered to reduce stalls.
Some small functions might be inlined into the code meaning there's less need for function calls. The compiler also pre calculates some values, which reduces the work the CPU has to do at runtime. Function like partition and quickSort have highly optimized control flows with less comparisons and jumps. Finally, loops and jumps are simplified further to use fewer instructions, and eliminating unnecessary instructions. The code uses more xorl %reg, %reg for making the registers zero instead of movl $0, %reg to reduce the instruction size.