Decompiler User Survey 3

October, 2021

1 User Survey 3

1.1 Self Assessment

The questions of this group are dedicated to self assessment to allow us to better understand trends in the answers of the users.

Did you take part in one of the previous decompiler surveys?

	Yes	No
Decompiler Survey 1 - October 2020	○	○
Decompiler Survey 2 - April 2021	○	○

How much time (approx.) did you spend working with C code?

- ○ None
- ○ A few hours
- ○ Several days
- ○ More than a year
- ○ On a regular basis

How much time did you spend reversing executables before?

- ○ None
- ○ A few hours
- ○ Several days
- ○ More than a year
- ○ On a regular basis

1.2 Decompiler Comparison

In this part, we will show you code snippets that are decompiled with different decompilers.

Clove (Hex-Rays):

```
_int64 sub_401C3D()
{
    char s[8]; // [rsp+0h] [rbp-60h] BYREF
    __int64 v2; // [rsp+8h] [rbp-58h]
    __int64 v3; // [rsp+10h] [rbp-50h]
    __int64 v4; // [rsp+18h] [rbp-48h]
    __int64 v5; // [rsp+20h] [rbp-40h]
    __int64 v6; // [rsp+28h] [rbp-38h]
    __int64 v7; // [rsp+30h] [rbp-30h]
    __int64 v8; // [rsp+38h] [rbp-28h]
    char v9; // [rsp+40h] [rbp-20h]
    __int64 v10; // [rsp+50h] [rbp-10h] BYREF
    int v11; // [rsp+58h] [rbp-8h]
    unsigned int v12; // [rsp+5Ch] [rbp-4h]

    *(_QWORD *)s = 0LL;
    v2 = 0LL;
    v3 = 0LL;
    v4 = 0LL;
    v5 = 0LL;
    v6 = 0LL;
    v7 = 0LL;
    v8 = 0LL;
    v9 = 0;
    printf("Enter any binary number: ");
    __isoc99_scanf("%lld", &v10);
    v12 = v10;
    while ( v10 > 0 )
    {
    v11 = v10 % 10000;
    if ( v11 == 1111 )
    {
        *(_WORD *)&s[strlen(s)] = 70;
    }
    else if ( v11 <= 1111 )
    {
        if ( v11 == 1110 )
        {
        *(_WORD *)&s[strlen(s)] = 69;
        }
        else if ( v11 == 1101 )
        {
        *(_WORD *)&s[strlen(s)] = 68;
        }
        else if ( v11 <= 1101 )
        {
        if ( v11 == 1100 )
        {
            *(_WORD *)&s[strlen(s)] = 67;
        }
        else if ( v11 == 1011 )
        {
            *(_WORD *)&s[strlen(s)] = 66;
        }
        else if ( v11 <= 1011 )
        {
            if ( v11 == 1010 )
            {
            *(_WORD *)&s[strlen(s)] = 65;
            }
            else if ( v11 == 1001 )
```

```
            {
            *(_WORD *)&s[strlen(s)] = 57;
            }
            else if ( v11 <= 1001 )
            {
            if ( v11 == 1000 )
            {
                *(_WORD *)&s[strlen(s)] = 56;
            }
            else if ( v11 == 111 )
            {
                *(_WORD *)&s[strlen(s)] = 55;
            }
            else if ( v11 <= 111 )
            {
                if ( v11 == 110 )
                {
                *(_WORD *)&s[strlen(s)] = 54;
                }
                else if ( v11 == 101 )
                {
                *(_WORD *)&s[strlen(s)] = 53;
                }
                else if ( v11 <= 101 )
                {
                if ( v11 == 100 )
                {
                    *(_WORD *)&s[strlen(s)] = 52;
                }
                else if ( v11 == 11 )
                {
                    *(_WORD *)&s[strlen(s)] = 51;
                }
                else if ( v11 <= 11 )
                {
                    if ( v11 == 10 )
                    {
                    *(_WORD *)&s[strlen(s)] = 50;
                    }
                    else if ( v11 )
                    {
                    if ( v11 == 1 )
                        *(_WORD *)&s[strlen(s)] = 49;
                    }
                    else
                    {
                    *(_WORD *)&s[strlen(s)] = 48;
                    }
                }
                }
            }
            }
        }
        }
    }
    v10 /= 10000LL;
    }
    printf("Binary number: %lld\\n", v12);
    printf("Hexadecimal number: %s", s);
    return 0LL;
}
```

Cumin (Ghidra):

```
undefined8 FUN_00401c3d(void)

{
```

```
  size_t sVar1;
  undefined8 local_68;
  undefined8 local_60;
  undefined8 local_58;
  undefined8 local_50;
  undefined8 local_48;
  undefined8 local_40;
  undefined8 local_38;
  undefined8 local_30;
  undefined local_28;
  long local_18;
  int local_10;
  uint local_c;

  local_68 = 0;
  local_60 = 0;
  local_58 = 0;
  local_50 = 0;
  local_48 = 0;
  local_40 = 0;
  local_38 = 0;
  local_30 = 0;
  local_28 = 0;
  printf("Enter any binary number: ");
  __isoc99_scanf(&DAT_00403259,&local_18);
  local_c = (uint)local_18;
  for (; 0 < local_18; local_18 = local_18 / 10000) {
    local_10 = (int)local_18 + (int)(local_18 / 10000) * -10000;
    if (local_10 == 0x457) {
      sVar1 = strlen((char *)&local_68);
      *(undefined2 *)((long)&local_68 + sVar1) = 0x46;
    }
    else {
      if (local_10 < 0x458) {
        if (local_10 == 0x456) {
          sVar1 = strlen((char *)&local_68);
          *(undefined2 *)((long)&local_68 + sVar1) = 0x45;
        }
        else {
          if (local_10 < 0x457) {
            if (local_10 == 0x44d) {
              sVar1 = strlen((char *)&local_68);
              *(undefined2 *)((long)&local_68 + sVar1) = 0x44;
            }
            else {
              if (local_10 < 0x44e) {
                if (local_10 == 0x44c) {
                  sVar1 = strlen((char *)&local_68);
                  *(undefined2 *)((long)&local_68 + sVar1) = 0x43;
                }
                else {
                  if (local_10 < 0x44d) {
                    if (local_10 == 0x3f3) {
                      sVar1 = strlen((char *)&local_68);
                      *(undefined2 *)((long)&local_68 + sVar1) = 0x42;
                    }
                    else {
                      if (local_10 < 0x3f4) {
                        if (local_10 == 0x3f2) {
                          sVar1 = strlen((char *)&local_68);
                          *(undefined2 *)((long)&local_68 + sVar1) = 0x41;
                        }
                        else {
                          if (local_10 < 0x3f3) {
                            if (local_10 == 0x3e9) {
                              sVar1 = strlen((char *)&local_68);
                              *(undefined2 *)((long)&local_68 + sVar1) = 0x39;
                            }
                            else {
```

```
                            if (local_10 < 0x3ea) {
                              if (local_10 == 1000) {
                                sVar1 = strlen((char *)&local_68);
                                *(undefined2 *)((long)&local_68 + sVar1) = 0x38;
                              }
                              else {
                                if (local_10 < 0x3e9) {
                                  if (local_10 == 0x6f) {
                                    sVar1 = strlen((char *)&local_68);
                                    *(undefined2 *)((long)&local_68 + sVar1) = 0x37;
                                  }
                                  else {
                                    if (local_10 < 0x70) {
                                      if (local_10 == 0x6e) {
                                        sVar1 = strlen((char *)&local_68);
                                        *(undefined2 *)((long)&local_68 + sVar1) = 0x36;
                                      }
                                      else {
                                        if (local_10 < 0x6f) {
                                          if (local_10 == 0x65) {
                                            sVar1 = strlen((char *)&local_68);
                                            *(undefined2 *)((long)&local_68 + sVar1) = 0x35;
                                          }
                                          else {
                                            if (local_10 < 0x66) {
                                              if (local_10 == 100) {
                                                sVar1 = strlen((char *)&local_68);
                                                *(undefined2 *)((long)&local_68 + sVar1) = 0x34;
                                              }
                                              else {
                                                if (local_10 < 0x65) {
                                                  if (local_10 == 0xb) {
                                                    sVar1 = strlen((char *)&local_68);
                                                    *(undefined2 *)((long)&local_68 + sVar1) =
                                                         0x33;
                                                  }
                                                  else {
                                                    if (local_10 < 0xc) {
                                                      if (local_10 == 10) {
                                                        sVar1 = strlen((char *)&local_68);
                                                       *(undefined2 *)((long)&local_68 + sVar1) =
                                                            0x32;
                                                      }
                                                      else {
                                                        if (local_10 < 0xb) {
                                                          if (local_10 == 0) {
                                                            sVar1 = strlen((char *)&local_68);
                                                            *(undefined2 *)
                                                             ((long)&local_68 + sVar1) = 0x30;
                                                          }
                                                          else {
                                                            if (local_10 == 1) {
                                                              sVar1 = strlen((char *)&local_68);
                                                              *(undefined2 *)
                                                               ((long)&local_68 + sVar1) = 0x31;
                                                            }
                                                          }
                                                        }
                                                      }
                                                    }
                                                  }
                                                }
                                              }
                                            }
                                          }
                                        }
                                      }
                                    }
                                  }
                                }
                              }
```

```
                                    }
                                  }
                                }
                              }
                            }
                          }
                        }
                      }
                    }
                  }
                }
              }
            }
          }
        }
      }
    }
  }
  printf("Binary number: %lld\\n",(ulong)local_c);
  printf("Hexadecimal number: %s",&local_68);
  return 0;
}
```

Chilli (dewolf):

```
long sub_401c3d() {
    size_t var_5;
    long i;
    long var_0;
    long var_2;
    long var_4;
    long * var_3;
    printf(/* format */ "Enter any binary number: ");
    var_3 = &var_0;
    __isoc99_scanf(/* format */ "%lld", var_3);
    var_2 = 0L;
    for (i = var_0; i > 0L; i /= 0x2710) {
        var_4 = i % 0x2710;
        switch(var_4) {
        case 0:
            var_3 = &var_2;
            var_5 = strlen(var_3);
            *(&var_2 + var_5) = 0x30;
            break;
        case 1:
            var_3 = &var_2;
            var_5 = strlen(var_3);
            *(&var_2 + var_5) = 0x31;
            break;
        case 10:
            var_3 = &var_2;
            var_5 = strlen(var_3);
            *(&var_2 + var_5) = 0x32;
            break;
        case 11:
            var_3 = &var_2;
            var_5 = strlen(var_3);
            *(&var_2 + var_5) = 0x33;
            break;
        case 100:
            var_3 = &var_2;
            var_5 = strlen(var_3);
            *(&var_2 + var_5) = 0x34;
            break;
        case 101:
            var_3 = &var_2;
            var_5 = strlen(var_3);
            *(&var_2 + var_5) = 0x35;
            break;
        case 110:
            var_3 = &var_2;
            var_5 = strlen(var_3);
            *(&var_2 + var_5) = 0x36;
            break;
        case 111:
            var_3 = &var_2;
            var_5 = strlen(var_3);
            *(&var_2 + var_5) = 0x37;
            break;
        case 0x3e8:
            var_3 = &var_2;
            var_5 = strlen(var_3);
            *(&var_2 + var_5) = 0x38;
            break;
        case 0x3e9:
            var_3 = &var_2;
            var_5 = strlen(var_3);
            *(&var_2 + var_5) = 0x39;
            break;
        case 0x3f2:
            var_3 = &var_2;
            var_5 = strlen(var_3);
```

```
            *(&var_2 + var_5) = 0x41;
            break;
        case 0x3f3:
            var_3 = &var_2;
            var_5 = strlen(var_3);
            *(&var_2 + var_5) = 0x42;
            break;
        case 0x44c:
            var_3 = &var_2;
            var_5 = strlen(var_3);
            *(&var_2 + var_5) = 0x43;
            break;
        case 0x44d:
            var_3 = &var_2;
            var_5 = strlen(var_3);
            *(&var_2 + var_5) = 0x44;
            break;
        case 0x456:
            var_3 = &var_2;
            var_5 = strlen(var_3);
            *(&var_2 + var_5) = 0x45;
            break;
        case 0x457:
            var_3 = &var_2;
            var_5 = strlen(var_3);
            *(&var_2 + var_5) = 0x46;
            break;
        }
    }
    printf(/* format */ "Binary number: %lld\\n", var_0 & 0xffffffff);
    var_3 = &var_2;
    printf(/* format */ "Hexadecimal number: %s", var_3);
    return 0L;
}
```

Please rank the decompiled code presented.

Clove

Cumin

Chilli

Which aspects in the decompiled codes above is especially favorable to you?

Which aspects of the presented decompiled functions do you deem unhelpful?

1.3 Measurement Part

In this part, we will show you short code snippets that represent the same code but are restructured differently. The goal is to find a way to establish a baseline on user preference between the presented approaches.

1.3.1 Copy Instructions If-Else

Please consider the following two code snippets, which both represent the same semantics with a different structure. In Snippet 1, `/* Block #10 */` is copied to achieve a simplified structure, while in Snippet 2 it occurs only once.

Snippet 1:

```
/* Block #1 */
while(var_1 > 0){
  if(var_2 == 2){
    /* Block #4 */
    if(var_4 == 4){
      /* Block #8 */
      continue;
    }
    /* Block #9 */
  }else{
    /* Block #5 */
  }
  /* Block #10 */ // <----- occurs two times
}
if(var_3 > 3){
  /* Block #6 */
}else{
  /* Block #7 */
}
/* Block #11 */
/* Block #10 */ // <----- occurs two times
```

Snippet 2:

```
/* Block #0 */
while(true){
  if(var_1 <= 0){
    if(var_2 != 2){
      /* Block #5 */
      break;
    }
    /* Block #4 */
    if(var_4 == 4){
      /* Block #8 */
      continue;
    }
    /* Block #9 */
    break;
  }else{
    if(var_3 > 3){
      /* Block #6 */
    }else{
      /* Block #7 */
    }
    /* Block #11 */
    break;
  }
}
/* Block #10 */ // <----- occurs only once
```

Which structure do you prefer in general?

- ○ Snippet 1 (Copying `/* Block #10 */`)
- ○ Snippet 2 (Not copying `/* Block #10 */`)
- ○ No preference

Now, please consider the following possible instruction sequences as replacement for `/* Block #10 */`:

Option 1:

```
return var;
```

Option 2:

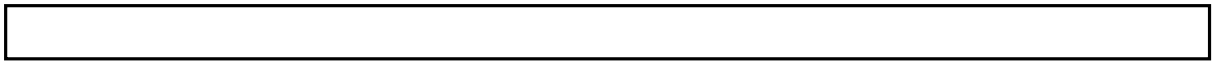
```
y = x + 5;
z = bar(y);
printf("Some text to print");
y += z;
return y;
```

Option 3:

```
y = x + 5;
z = bar(y);
printf("Enter a number larger than %d: \\n", z);
scanf("%d", &numb1);
printf("Enter a number smaller than %d: \\n", z);
scanf("%d", &numb2);
y += z;
printf("Enter a number larger than %d: \\n", y);
scanf("%d", &numb3);
printf("Enter a number smaller than %d: \\n", y);
scanf("%d", &numb4);
diff_1 = numb1 - numb2;
diff_2 = numb3 - numb4;
printf("The two differences are %d and %d: \\n", diff_1, diff_2);
return diff_1 + diff_2;
```

	Snippet 1 (Copying `/* Block #10 */`)	Snippet 2 (Not copying `/* Block #10 */`)	No preference
Option 1 (1 Instruction)	○	○	○
Option 2 (5 Instructions)	○	○	○
Option 3 (15 Instructions)	○	○	○

Please explain your choices. Which criteria guide your decision?



Now, please assume that `/* Block #10 */` contains a more complex structure, for example the following:

```
while(var > 1){
    printf("Enter a number \\n");
    scanf("%d", &numb1);
    if(var % numb == 0){
      var /= numb;
    }else{
      var -= numb;
    }
    printf("The new number is %d, \\n", var);
  }
  return var;
```

Snippet 1 and Snippet 2 are structured as before. In Snippet 3, `/* Block #10 */` is extracted as a function.

Snippet 1:

```
/* Block #0 */
while(var_1 > 0){
  if(var_2 == 2){
    /* Block #4 */
    if(var_4 == 4){
      /* Block #8 */
      continue;
    }
    /* Block #9 */
  }else{
    /* Block #5 */
  }
  while(var > 1){
    printf("Enter a number \\n");
    scanf("%d", &numb1);
    if(var % numb == 0){
      var /= numb;
    }else{
      var -= numb;
    }
    printf("The new number is %d, \\n", var);
  }
  return var;
}
if(var_3 > 3){
  /* Block #6 */
}else{
  /* Block #7 */
}
/* Block #11 */
while(var > 1){
  printf("Enter a number \\n");
  scanf("%d", &numb1);
  if(var % numb == 0){
    var /= numb;
  }else{
    var -= numb;
  }
```

```
  printf("The new number is %d, \\n", var);
}
return var;
```

Snippet 2:

```
/* Block #0 */
while(true){
  if(var_1 < 0){
    if(var_2 != 2){
      /* Block #5 */
      break;
    }
    /* Block #4 */
    if(var_4 == 4){
      /* Block #8 */
      continue;
    }
    /* Block #9 */
    break;
  }else{
    if(var_3 > 3){
      /* Block #6 */
    }else{
      /* Block #7 */
    }
    /* Block #11 */
    break;
  }
}
while(var > 1){
  printf("Enter a number \\n");
  scanf("%d", &numb1);
  if(var % numb == 0){
    var /= numb;
  }else{
    var -= numb;
  }
  printf("The new number is %d, \\n", var);
}
return var;
```

Snippet 3:

```
/* Block #0 */
while(var_1 > 0){
  if(var_2 == 2){
    /* Block #4 */
    if(var_4 == 4){
      /* Block #8 */
      continue;
    }
    /* Block #9 */
  }else{
    /* Block #5 */
  }
```

```
  return call_sub(var);
}
if(var_3 > 3){
  /* Block #6 */
}else{
  /* Block #7 */
}
/* Block #11 */
return call_sub(var);


int call_sub(int var){
  while(var > 1){
    printf("Enter a number \\n");
    scanf("%d", &numb1);
    if(var % numb == 0){
      var /= numb;
    }else{
      var -= numb;
    }
    printf("The new number is %d, \\n", var);
  }
  return var;
}
```

Which Snippets do you like? (Multiple choices are possible)

- ☐ Snippet 1 (Copying `/* Block #10 */`) (Sample 1)
- ☐ Snippet 2 (Not copying `/* Block #10 */`) (Sample 2)
- ☐ Snippet 3 (Extracting `/* Block #10 */` as a function and calling it twice)

Please explain your choice.

1.3.2 Copy Instructions Multiple Exit for Loops

Please consider the following two code snippets, which both represent the same semantics with a different structure. In Snippet 1, `/* Block #10 */` is copied to achieve a simplified structure, while in Snippet 2 it occurs only once.

Snippet 1:

```
/* Block #0 */
if(var_0 > 10){
  while(var_1 > 0){
    if(var_2 == 2){
      /* Block #4 */
      if(var_4 == 4){
        /* Block #8 */
        continue;
      }
      /* Block #9 */
```

```
    }else{
      /* Block #5 */
    }
    /* Block #10 */ // <----- occurs two times
  }
}
/* Block #3 */
if(var_3 > 3){
  /* Block #6 */
}else{
  /* Block #7 */
}
/* Block #11 */
/* Block #10 */ // <----- occurs two times
```

Snippet 2:

```
/* Block #0 */
if(var_0 > 10){
  while(true){
    if(var_1 > 0){
      exit_1 = 0;
      break;
    }
    if(var_2 == 2){
      /* Block #4 */
      if(var_4 == 4){
          /* Block #8 */
          continue;
      }
      /* Block #9 */
    }else{
      /* Block #5 */
    }
      exit = 1;
      break;
    }
}
if(var_0 <= 10 || exit_1 == 0){
  if(var_3 > 3){
    /* Block #6 */
  }else{
    /* Block #7 */
  }
  /* Block #11 */
}
/* Block #10 */ // <----- occurs only once
```

Which structure do you prefer in general?

- ○ Snippet 1 (Copying `/* Block #10 */`)
- ○ Snippet 2 (Not copying `/* Block #10 */`)
- ○ No preference

Now, please consider the following possible instruction sequences as replacement for `/* Block #10 */`:

Option 1:

```
return var;
```

Option 2:

```
y = x + 5;
z = bar(y);
printf("Some text to print");
y += z;
return y;
```

Option 3:

```
y = x + 5;
z = bar(y);
printf("Enter a number larger than %d: \\n", z);
scanf("%d", &numb1);
printf("Enter a number smaller than %d: \\n", z);
scanf("%d", &numb2);
y += z;
printf("Enter a number larger than %d: \\n", y);
scanf("%d", &numb3);
printf("Enter a number smaller than %d: \\n", y);
scanf("%d", &numb4);
diff_1 = numb1 - numb2;
diff_2 = numb3 - numb4;
printf("The two differences are %d and %d: \\n", diff_1, diff_2);
return diff_1 + diff_2;
```

	Snippet 1 (Copying `/* Block #10 */`)	Snippet 2 (Not copying `/* Block #10 */`)	No preference
Option 1 (1 Instruction)	○	○	○
Option 2 (5 Instructions)	○	○	○
Option 3 (15 Instructions)	○	○	○

Please explain your choices. Which criteria guide your decision?

Now, please assume that `/* Block #10 */` contains a more complex structure, like e.g. the following:

```
while(var > 1){
    printf("Enter a number \\n");
    scanf("%d", &numb1);
    if(var % numb == 0){
      var /= numb;
    }else{
      var -= numb;
    }
    printf("The new number is %d, \\n", var);
  }
  return var;
```

Snippet 1 and Snippet 2 are structured as before. In Snippet 3, `/* Block #10 */` is extracted as a function.

Snippet 1:

```
/* Block #0 */
if(var_0 > 10){
  while(var_1 > 0){
    if(var_2 == 2){
      /* Block #4 */
      if(var_4 == 4){
        /* Block #8 */
        continue;
      }
      /* Block #9 */
    }else{
      /* Block #5 */
    }
    while(var > 1){
      printf("Enter a number \\n");
      scanf("%d", &numb1);
      if(var % numb == 0){
        var /= numb;
      }else{
        var -= numb;
      }
      printf("The new number is %d, \\n", var);
    }
    return var;
  }
}
/* Block #3 */
if(var_3 > 3){
  /* Block #6 */
}else{
  /* Block #7 */
}
/* Block #11 */
while(var > 1){
  printf("Enter a number \\n");
  scanf("%d", &numb1);
  if(var % numb == 0){
    var /= numb;
```

```
  }else{
      var -= numb;
  }
  printf("The new number is %d, \\n", var);
}
return var;
```

Snippet 2:

```
/* Block #0 */
if(var_0 > 10){
  while(true){
    if(var_1 > 0){
      exit_1 = 0;
      break;
    }
    if(var_2 == 2){
      /* Block #4 */
      if(var_4 == 4){
        /* Block #8 */
        continue;
      }
      /* Block #9 */
    }else{
      /* Block #5 */
    }
    exit = 1;
    break;
  }
}
if(var_0 <= 10 || exit_1 == 0){
  if(var_3 > 3){
    /* Block #6 */
  }else{
    /* Block #7 */
  }
  /* Block #11 */
}
while(var > 1){
  printf("Enter a number \\n");
  scanf("%d", &numb1);
  if(var % numb == 0){
      var /= numb;
  }else{
      var -= numb;
  }
  printf("The new number is %d, \\n", var);
}
return var;
```

Snippet 3:

```
/* Block #0 */
if(var_0 > 10){
  while(var_1 > 0){
    if(var_2 == 2){
```

```
      /* Block #4 */
      if(var_4 == 4){
        /* Block #8 */
        continue;
      }
      /* Block #9 */
    }else{
      /* Block #5 */
    }
    return call_sub(var);
  }
}
/* Block #3 */
if(var_3 > 3){
  /* Block #6 */
}else{
  /* Block #7 */
}
/* Block #11 */
return call_sub(var);


int call_sub(int var){
  while(var > 1){
      printf("Enter a number \\n");
      scanf("%d", &numb1);
      if(var % numb == 0){
        var /= numb;
      }else{
        var -= numb;
      }
      printf("The new number is %d, \\n", var);
  }
  return var;
}
```

Which Snippets do you like? (Multiple choices are possible)

- ☐ Snippet 1 (Copying `/* Block #10 */`) (Sample 1)
- ☐ Snippet 2 (Not copying `/* Block #10 */`) (Sample 2)
- ☐ Snippet 3 (Extracting `/* Block #10 */` as a function and calling it twice)

Please explain your choice.

1.3.3 Copy Instructions Multiple Entry for Loops

Please consider the following two code snippets, which both represent the same semantics with a different structure. In Snippet 1, `/* Block #4 */` is copied to achieve a simplified structure, while in Snippet 2 it is not.

Snippet 1:

```
/* Block #0 */
if(var_0 > 10){
  /* Block #1 */
  /* Block #4 */ // <----- occurs two times
}else{
  /* Block #2 */
}
while(var_3 > 0){
  /* Block #4 */ // <----- occurs two times
}
/* Block #5 */
```

Snippet 2:

```
/* Block #0 */
if(var_0 > 10){
  /* Block #1 */
  entry_1 = 1;
}else{
  /* Block #2 */
  entry_1 = 0;
}
while(true){
  if(entry_1 == 0 && var_3 > 0){
    break;
  }
  /* Block #4 */ // <----- occurs only once
  entry_1 = 0;
}
/* Block #5 */
```

Which structure do you prefer in general?

- ○ Snippet 1 (Copying `/* Block #4 */`)
- ○ Snippet 2 (Not copying `/* Block #4 */`)
- ○ No preference

Now, please consider the following possible instruction sequences as replacement for `/* Block #4 */`:

Option 1:

```
return var;
```

Option 2:

```
y = x + 5;
z = bar(y);
printf("Some text to print");
y += z;
return y;
```

Option 3:

```
y = x + 5;
z = bar(y);
printf("Enter a number larger than %d: \\n", z);
scanf("%d", &numb1);
printf("Enter a number smaller than %d: \\n", z);
scanf("%d", &numb2);
y += z;
printf("Enter a number larger than %d: \\n", y);
scanf("%d", &numb3);
printf("Enter a number smaller than %d: \\n", y);
scanf("%d", &numb4);
diff_1 = numb1 - numb2;
diff_2 = numb3 - numb4;
printf("The two differences are %d and %d: \\n", diff_1, diff_2);
return diff_1 + diff_2;
```

	Snippet 1 (Copying `/* Block #4 */`)	Snippet 2 (Not copying `/* Block #4 */`)	No preference
Option 1 (1 Instruction)	○	○	○
Option 2 (5 Instructions)	○	○	○
Option 3 (15 Instructions)	○	○	○

Please explain your choices. Which criteria guide your decision?

Now, please assume that `/* Block #4 */` contains a more complex structure, like e.g. the following:

```
while(var > 1){
    printf("Enter a number \\n");
    scanf("%d", &numb1);
    if(var % numb == 0){
      var /= numb;
    }else{
      var -= numb;
    }
    printf("The new number is %d, \\n", var);
  }
  return var;
```

Snippet 1 and Snippet 2 are structured as before. In Snippet 3, `/* Block #4 */` is extracted as a function.

Snippet 1:

```
/* Block #0 */
if(var_0 > 10){
  /* Block #1 */
  while(var > 1){
    printf("Enter a number \\n");
    scanf("%d", &numb1);
    if(var % numb == 0){
      var /= numb;
    }else{
      var -= numb;
    }
    printf("The new number is %d, \\n", var);
  }
}else{
  /* Block #2 */
}
while(var_3 > 0){
  while(var > 1){
    printf("Enter a number \\n");
    scanf("%d", &numb1);
    if(var % numb == 0){
      var /= numb;
    }else{
      var -= numb;
    }
    printf("The new number is %d, \\n", var);
  }
}
/* Block #5 */
```

Snippet 2:

```
/* Block #0 */
if(var_0 > 10){
  /* Block #1 */
  entry_1 = 1;
```

```
}else{
  /* Block #2 */
  entry_1 = 0;
}
while(true){
  if(entry_1 == 0 && var_3 > 0){
    break;
  }
  while(var > 1){
    printf("Enter a number \\n");
    scanf("%d", &numb1);
    if(var % numb == 0){
      var /= numb;
    }else{
      var -= numb;
    }
    printf("The new number is %d, \\n", var);
  }
  entry_1 = 0;
}
/* Block #5 */
```

Snippet 3:

```
/* Block #0 */
if(var_0 > 10){
  /* Block #1 */
  var = call_sub(var);
}else{
  /* Block #2 */
}
while(var_3 > 0){
  var = call_sub(var);
}
/* Block #5 */

int call_sub(int var){
  while(var > 1){
    printf("Enter a number \\n");
    scanf("%d", &numb1);
    if(var % numb == 0){
      var /= numb;
    }else{
      var -= numb;
    }
    printf("The new number is %d, \\n", var);
  }
  return var;
}
```

Which Snippets do you like? (Multiple choices are possible)

- ☐ Snippet 1 (Copying `/* Block #4 */`) (Sample 1)
- ☐ Snippet 2 (Not copying `/* Block #4 */`) (Sample 2)
- ☐ Snippet 3 (Extracting `/* Block #4 */` as a function and calling it twice)

Please explain your choice.

1.3.4 Switch

Please consider the following two code snippets.

Snippet 1:

```
/* ... */
switch(var){
    case 1:
        /* Block #1 */
        break;
    case 2:
        /* Block #2 */
        break;
}
/* ... */
```

Snippet 2:

```
/* ... */
if(var == 1){
    /* Block #1 */
}
if(var == 2){
    /* Block #2 */
}
/* ... */
```

	Snippet 1	Snippet 2	No preference
Which snippet do you prefer?	○	○	○

Please explain your choice.

Please consider the following two code snippets.

Snippet 1:

```
/* ... */
switch(var){
    case 1:
        /* Block #1 */
        break;
    case 2:
        /* Block #2 */
```

```
        break;
    case 3:
        /* Block #3 */
        break;
    case 4:
        /* Block #4 */
        break;
    case 5:
        /* Block #5 */
        break;
}
/* ... */
```

Snippet 2:

```
/* ... */
if(var == 1){
    /* Block #1 */
}
if(var == 2){
    /* Block #2 */
}
if(var == 3){
    /* Block #3 */
}
if(var == 4){
    /* Block #4 */
}
if(var == 5){
    /* Block #5 */
}
/* ... */
```

	Snippet 1	Snippet 2	No preference
Which snippet do you prefer?	○	○	○

Please explain your choice.

Please consider the following three code snippets.

Snippet 1:

```
/* ... */
switch(var){
    case 1:
        /* Block #1 */
        break;
    case 2:
        /* Block #2 */
        break;
    default:
```

```
        /* Block #3 */
        break;
}
/* ... */
```

Snippet 2:

```
/* ... */
if(var == 1){
    /* Block #1 */
}
if(var == 2){
    /* Block #2 */
}
if(var != 1 && var !=2 ){
    /* Block #3 */
}
/* ... */
```

Snippet 3:

```
/* ... */
if(var == 1){
    /* Block #1 */
}else{
    if(var == 2){
        /* Block #2 */
    }else{
        /* Block #3 */
    }
}
/* ... */
```

	Snippet 1	Snippet 2	Snippet 3	Both if-Snippets	No preference
Which snippet do you prefer?	○	○	○	○	○

Please explain your choice.

Please consider the following three code snippets.

Snippet 1:

```
/* ... */
switch(var){
    case 1:
        /* Block #1 */
        break;
```

```
    case 2:
        /* Block #2 */
        break;
    case 3:
        /* Block #3 */
        break;
    case 4:
        /* Block #4 */
        break;
    case 5:
        /* Block #5 */
        break;
    default:
        /* Block #6 */
        break;
}
/* ... */
```

Snippet 2:

```
/* ... */
if(var == 1){
    /* Block #1 */
}
if(var == 2){
    /* Block #2 */
}
if(var == 3){
    /* Block #3 */
}
if(var == 4){
    /* Block #4 */
}
if(var == 5){
    /* Block #5 */
}
if(var != 1 && var !=2 && var != 3 && var != 4 && var != 5){
    /* Block #6 */
}
/* ... */
```

Snippet 3:

```
/* ... */
if(var == 1){
    /* Block #1 */
}else{
    if(var == 2){
        /* Block #2 */
    }else{
        if(var == 3){
            /* Block #3 */
        }else{
            if(var == 4){
                /* Block #4 */
            }else{
```

```
                if(var == 5){
                    /* Block #5 */
                }else{
                    /* Block #6 */
                }
            }
        }
    }
}
/* ... */
```

	Snippet 1	Snippet 2	Snippet 3	Both if-Snippets	No preference
Which snippet do you prefer?	○	○	○	○	○

Please explain your choice.

1.3.5 Conditions

Please consider the following code snippets which all represent the same semantics and rank them from your favourite to least favourite.

Snippet 1:

```
/* ... */
if( (a == 5) && (c <= 3d-4) && ( ((x+y) < (z <<2)) || (u > 20) ) ){
    // do something
}
/* ... */
```

Snippet 2:

```
/* ... */
if(a == 5){
    if(c <= 3d-4){
        if(((x+y) < (z <<2)) || (u > 20) ){
            // do something
        }
    }
}
/* ... */
```

Snippet 3:

```
/* ... */
bool cond1 = a == 5;
bool cond2 = c <= 3d-4;
bool cond3 = ((x+y) < (z <<2)) || (u > 20);
if( cond1 && cond2 && cond3 ){
    // do something
}
/* ... */
```

Snippet 1 ☐

Snippet 2 ☐

Snippet 3 ☐

Please explain your choice.

☐

1.3.6 For-loops

Please consider the following code snippets which all represent the same code and rank them from your favourite to least favourite.

Snippet 1:

```
/* ... */
for(i = var_0; i > 10; i = (i * 3257 >> 5 >> 2) - (i >> 32)){
    /* Block #1 */
}
/* ... */
```

Snippet 2:

```
/* ... */
c = car_0;
while(c > 10){
    /* Block #1 */
    c = (c * 3257 >> 5 >> 2) - (c >> 32);
}
/* ... */
```

Snippet 3:

```
/* ... */
for(i = var_0; i > 10; i = x - y){
    /* Block #1 */
    x = i * 3257 >> 5 >> 2;
    y = i >> 32;
```

```
}
/* ... */
```

Snippet 4:

```
/* ... */
for(i = var_0; i > 10; i = x - (i >> 32)){
    /* Block #1 */
    x = i * 3257 >> 5 >> 2;
}
/* ... */
```

Snippet 1 (Sample 1) ☐

Snippet 2 (Sample 2) ☐

Snippet 3 (Sample 3) ☐

Snippet 4 (Sample 4) ☐

Please explain your choice.

☐

1.4 Feedback

Thank you very much for getting this far and also for participating in any previous surveys! This is very likely to be our last decompiler survey for now.

If you would like to leave us any feedback about the survey, please use the lines below to help us improve ourselves.

☐