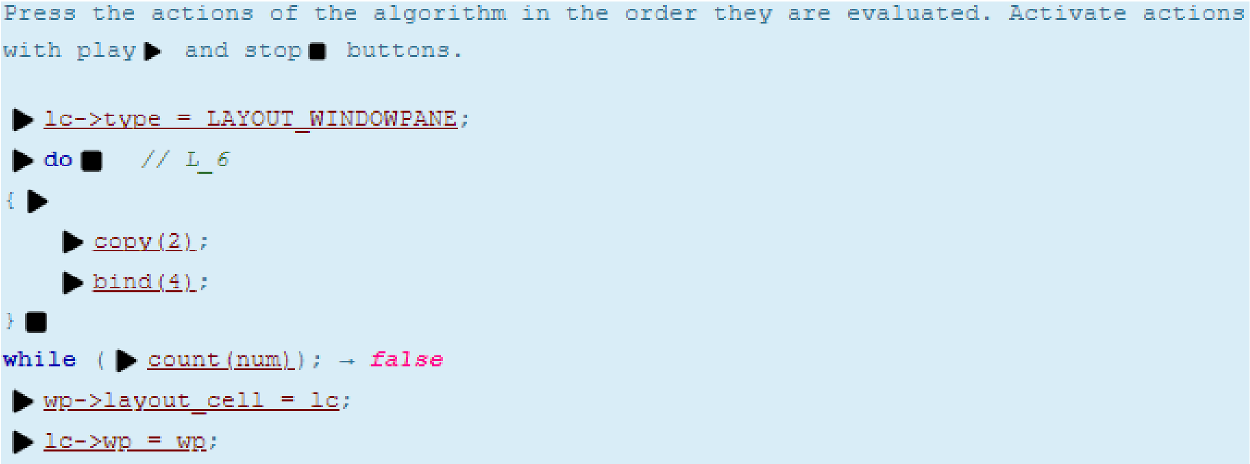
Supplementary material.

Examples of generated exercises for Control-flow domain.

Target concepts:

- else-if;
- DO-WHILE loop;
- break.

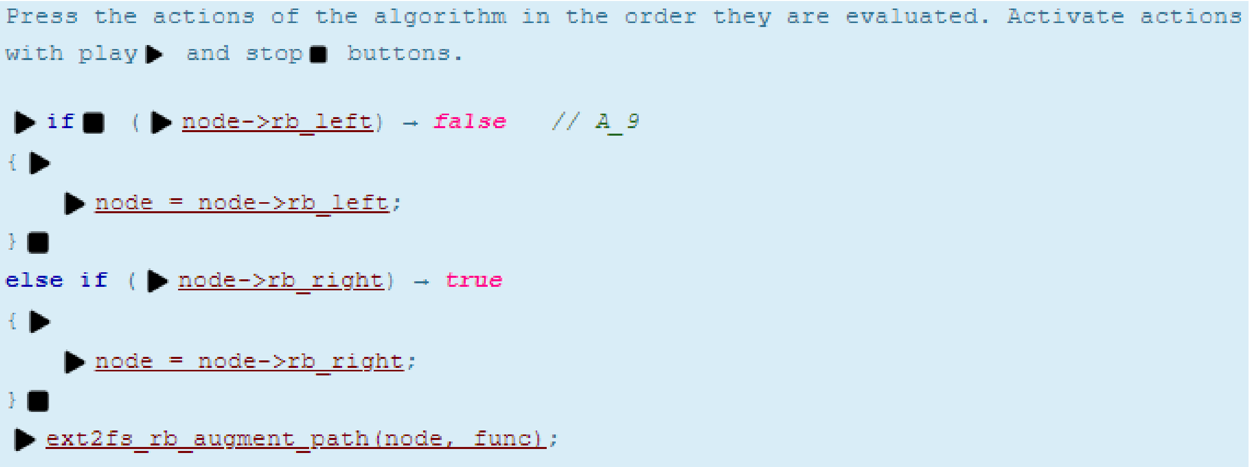
1. An exercise of low (L) difficulty.



Press the actions of the algorithm in the order they are evaluated. Activate actions with play ▶ and stop ■ buttons.

```
▶ lc->type = LAYOUT_WINDOWPANE;
▶ do ■   // L_6
{ ▶
    ▶ copy(2);
    ▶ bind(4);
} ■
while ( ▶ count(num)); → false
▶ wp->layout_cell = lc;
▶ lc->wp = wp;
```

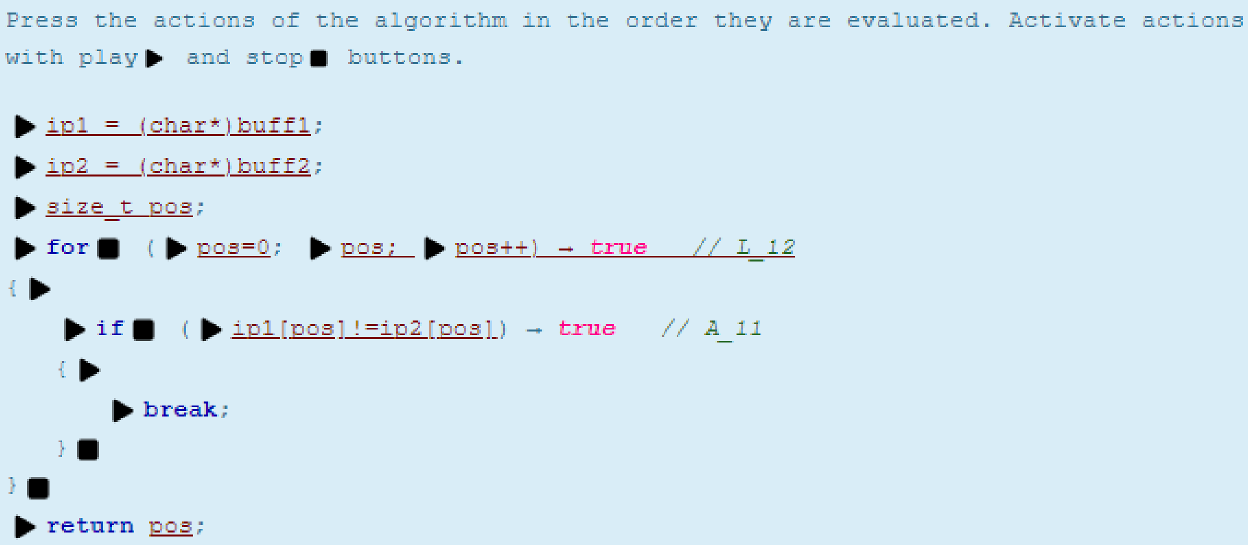
Problem L-1 (difficulty: 0.1643)



Press the actions of the algorithm in the order they are evaluated. Activate actions with play ▶ and stop ■ buttons.

```
▶ if ■  ( ▶ node->rb_left) → false   // A_9
{ ▶
    ▶ node = node->rb_left;
} ■
else if ( ▶ node->rb_right) → true
{ ▶
    ▶ node = node->rb_right;
} ■
▶ ext2fs_rb_augment_path(node, func);
```

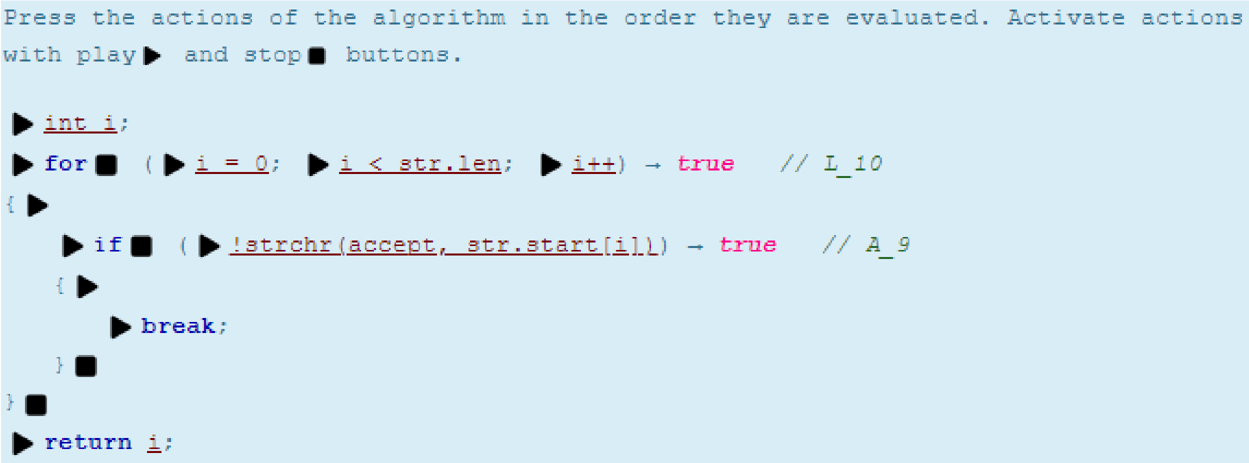
Problem L-2 (difficulty: 0.3408)



Press the actions of the algorithm in the order they are evaluated. Activate actions with play ▶ and stop ■ buttons.

```
▶ ip1 = (char*)buff1;
▶ ip2 = (char*)buff2;
▶ size_t pos;
▶ for ■  ( ▶ pos=0;  ▶ pos;  ▶ pos++) → true    // L_12
{ ▶
    ▶ if ■  ( ▶ ip1[pos]!=ip2[pos]) → true    // A_11
    { ▶
        ▶ break;
    } ■
} ■
▶ return pos;
```

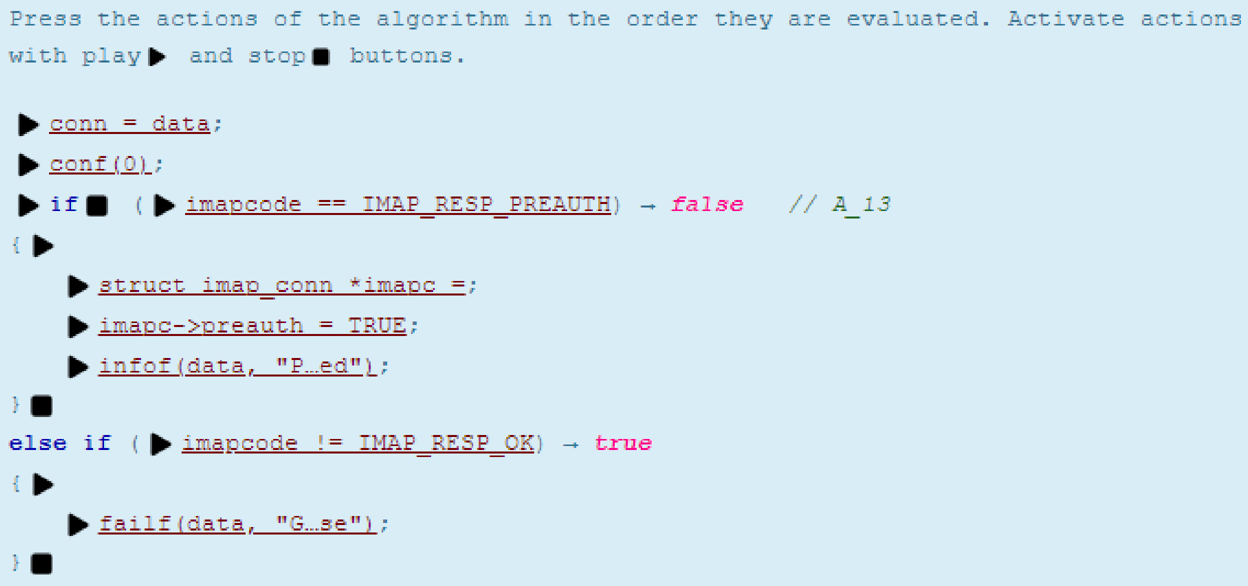
Problem L-3 (difficulty: 0.8024)



Press the actions of the algorithm in the order they are evaluated. Activate actions with play ▶ and stop ■ buttons.

```
▶ int i;
▶ for ■  ( ▶ i = 0;  ▶ i < str.len;  ▶ i++) → true    // L_10
{ ▶
    ▶ if ■  ( ▶ !strchr(accept, str.start[i])) → true    // A_9
    { ▶
        ▶ break;
    } ■
} ■
▶ return i;
```

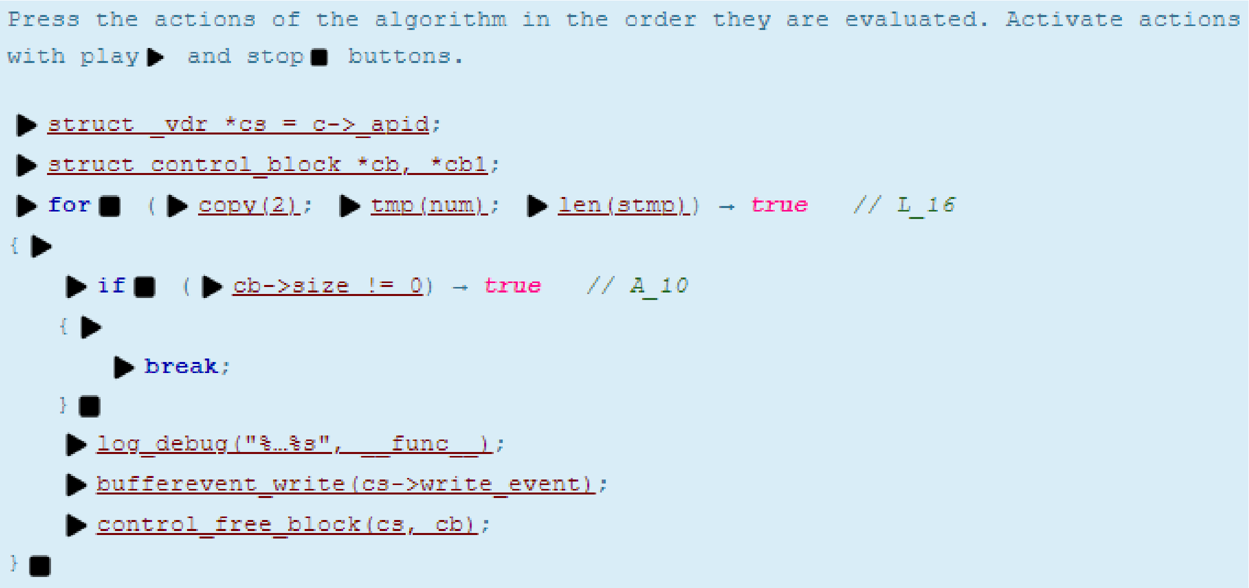
Problem L-4 (difficulty: 0.7794)



Press the actions of the algorithm in the order they are evaluated. Activate actions with play ▶ and stop ■ buttons.

```
▶ conn = data;
▶ conf(0);
▶ if ■  ( ▶ imapcode == IMAP_RESP_PREAUTH) → false    // A_13
{ ▶
    ▶ struct imap_conn *imapc =;
    ▶ imapc->preauth = TRUE;
    ▶ infof(data, "P…ed");
} ■
else if ( ▶ imapcode != IMAP_RESP_OK) → true
{ ▶
    ▶ failf(data, "G…se");
} ■
```

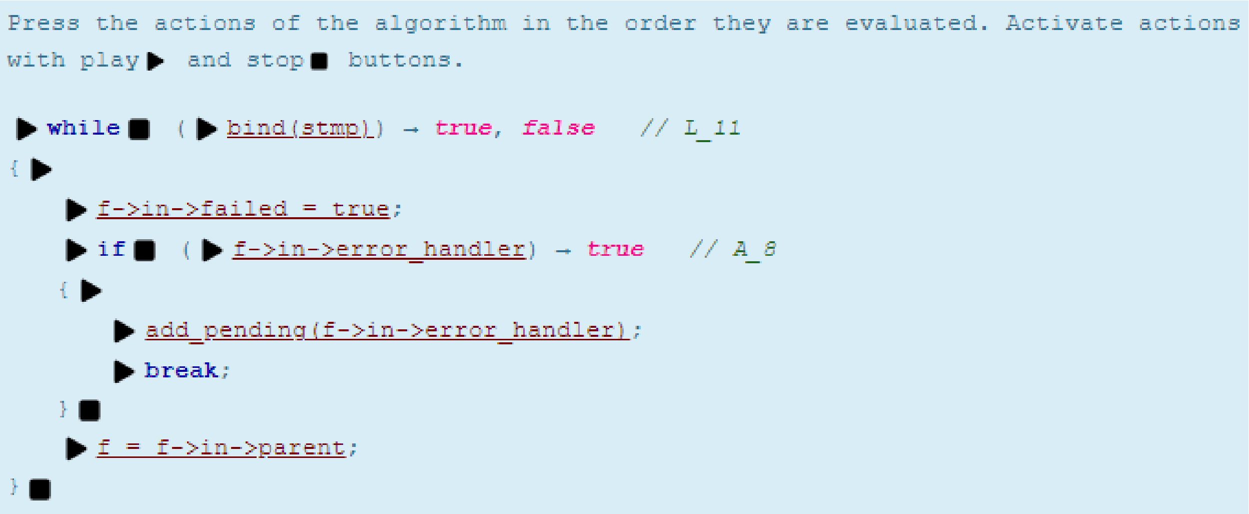
Problem L-5 (difficulty: 0.3566)



```
Press the actions of the algorithm in the order they are evaluated. Activate actions
with play ▶  and stop ■  buttons.

▶ struct _vdr *cs = c->_apid;
▶ struct control_block *cb, *cb1;
▶ for ■  ( ▶ copy(2); ▶ tmp(num); ▶ len(stmp)) → true   // L_16
{ ▶
    ▶ if ■  ( ▶ cb->size != 0) → true   // A_10
    { ▶
        ▶ break;
    } ■
    ▶ log_debug("%…%s", __func__);
    ▶ bufferevent_write(cs->write_event);
    ▶ control_free_block(cs, cb);
} ■
```

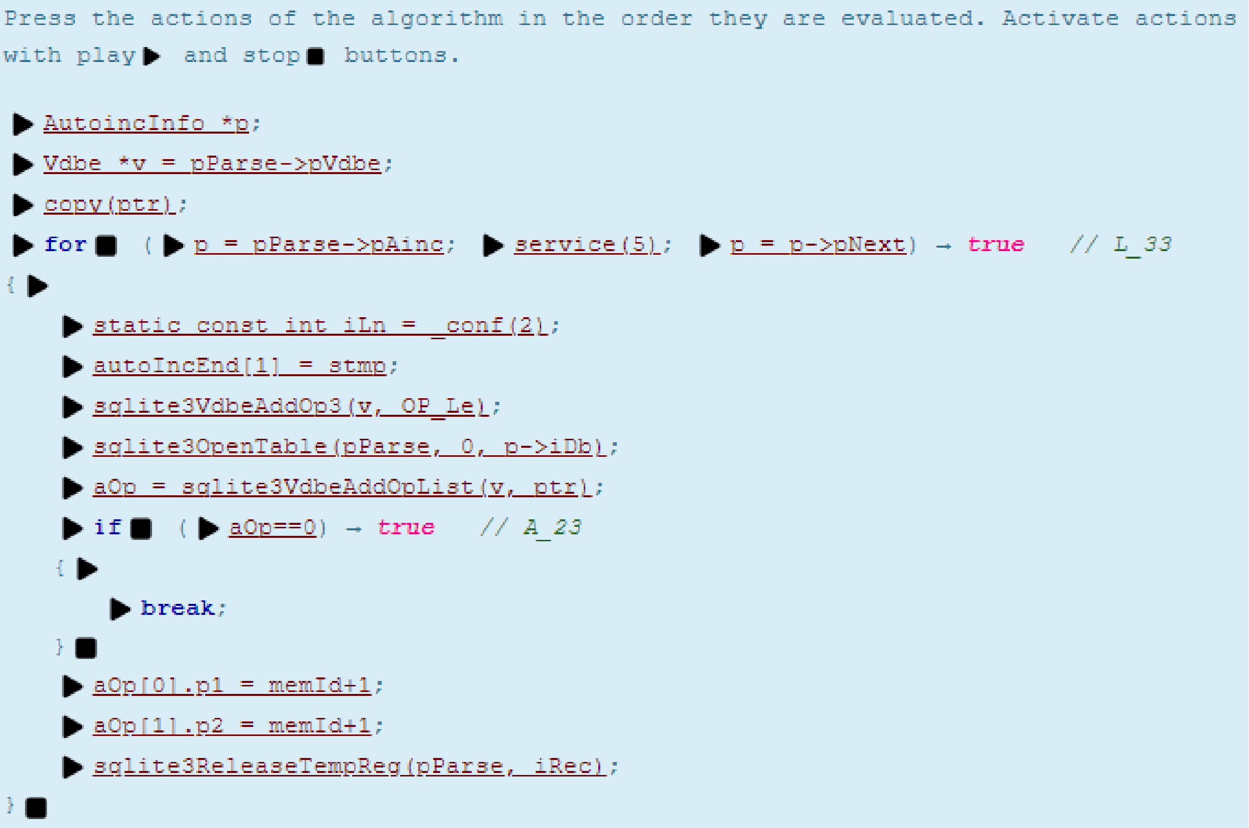
Problem L-6 (difficulty: 0.6410)



```
Press the actions of the algorithm in the order they are evaluated. Activate actions
with play ▶  and stop ■  buttons.

▶ while ■  ( ▶ bind(stmp)) → true, false   // L_11
{ ▶
    ▶ f->in->failed = true;
    ▶ if ■  ( ▶ f->in->error_handler) → true   // A_8
    { ▶
        ▶ add_pending(f->in->error_handler);
        ▶ break;
    } ■
    ▶ f = f->in->parent;
} ■
```

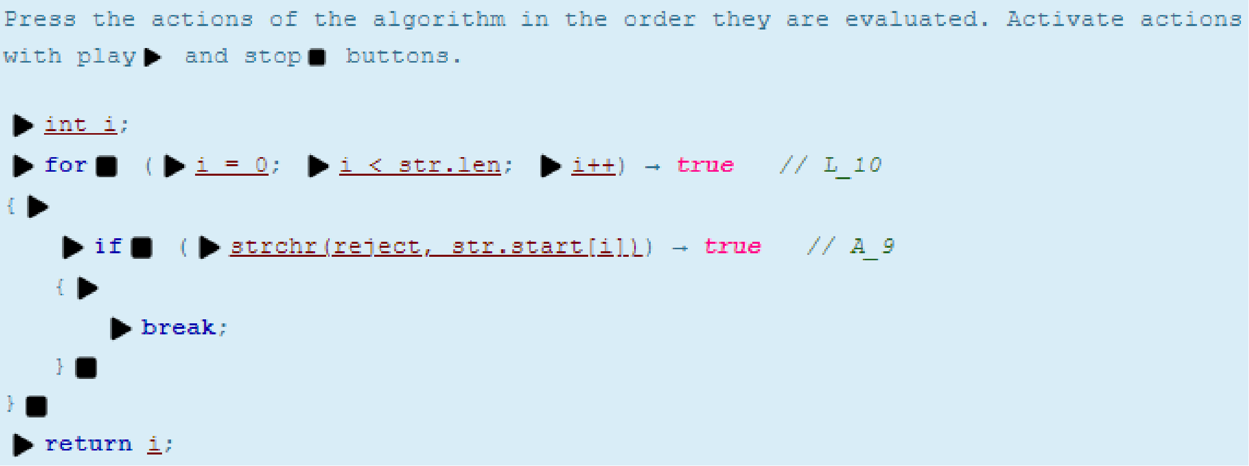
Problem L-7 (difficulty: 0.7178)



Press the actions of the algorithm in the order they are evaluated. Activate actions with play ▶ and stop ■ buttons.

```
▶ AutoincInfo *p;
▶ Vdbe *v = pParse->pVdbe;
▶ copy(ptr);
▶ for ■ ( ▶ p = pParse->pAinc; ▶ service(5); ▶ p = p->pNext) → true   // L_33
{ ▶
    ▶ static const int iLn = conf(2);
    ▶ autoIncEnd[1] = stmp;
    ▶ sqlite3VdbeAddOp3(v, OP_Le);
    ▶ sqlite3OpenTable(pParse, 0, p->iDb);
    ▶ aOp = sqlite3VdbeAddOpList(v, ptr);
    ▶ if ■ ( ▶ aOp==0) → true   // A_23
    { ▶
        ▶ break;
    } ■
    ▶ aOp[0].p1 = memId+1;
    ▶ aOp[1].p2 = memId+1;
    ▶ sqlite3ReleaseTempReg(pParse, iRec);
} ■
```

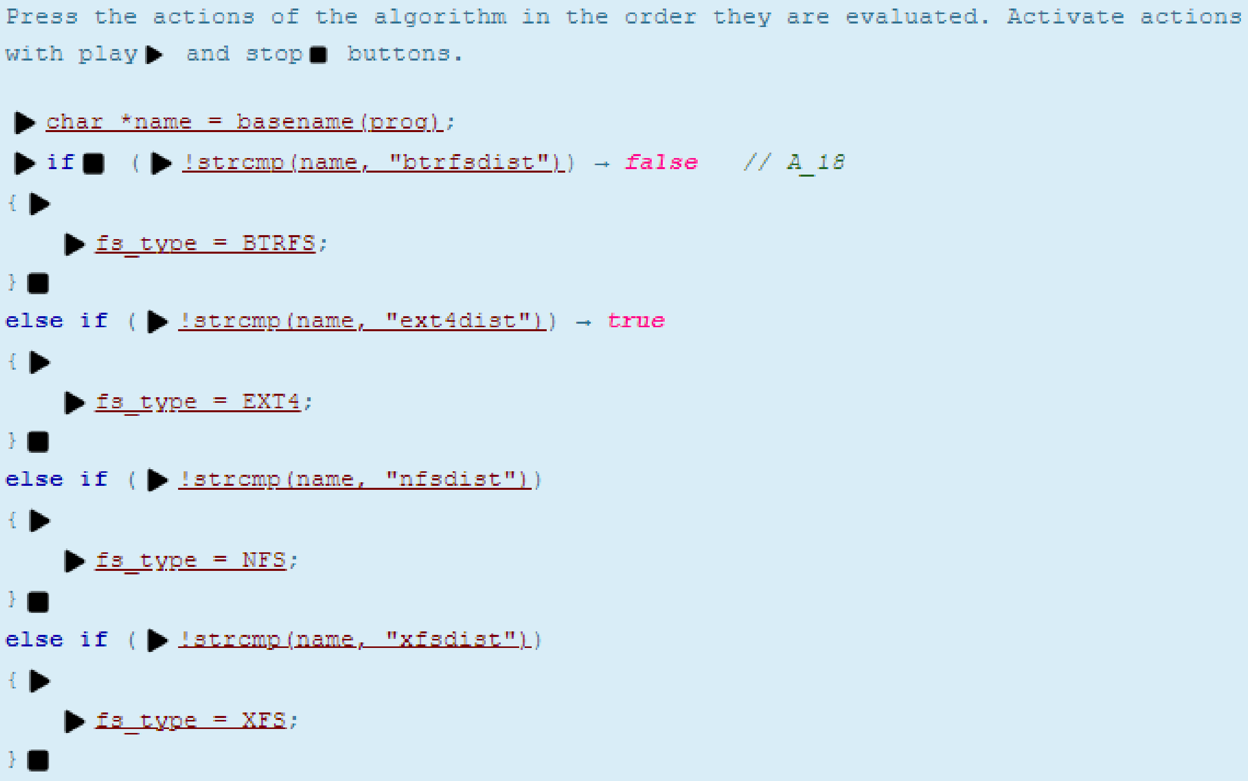
Problem L-8 (difficulty: 0.7305)



Press the actions of the algorithm in the order they are evaluated. Activate actions with play ▶ and stop ■ buttons.

```
▶ int i;
▶ for ■ ( ▶ i = 0; ▶ i < str.len; ▶ i++) → true   // L_10
{ ▶
    ▶ if ■ ( ▶ strchr(reject, str.start[i])) → true   // A_9
    { ▶
        ▶ break;
    } ■
} ■
▶ return i;
```

Problem L-9 (difficulty: 0.7794)

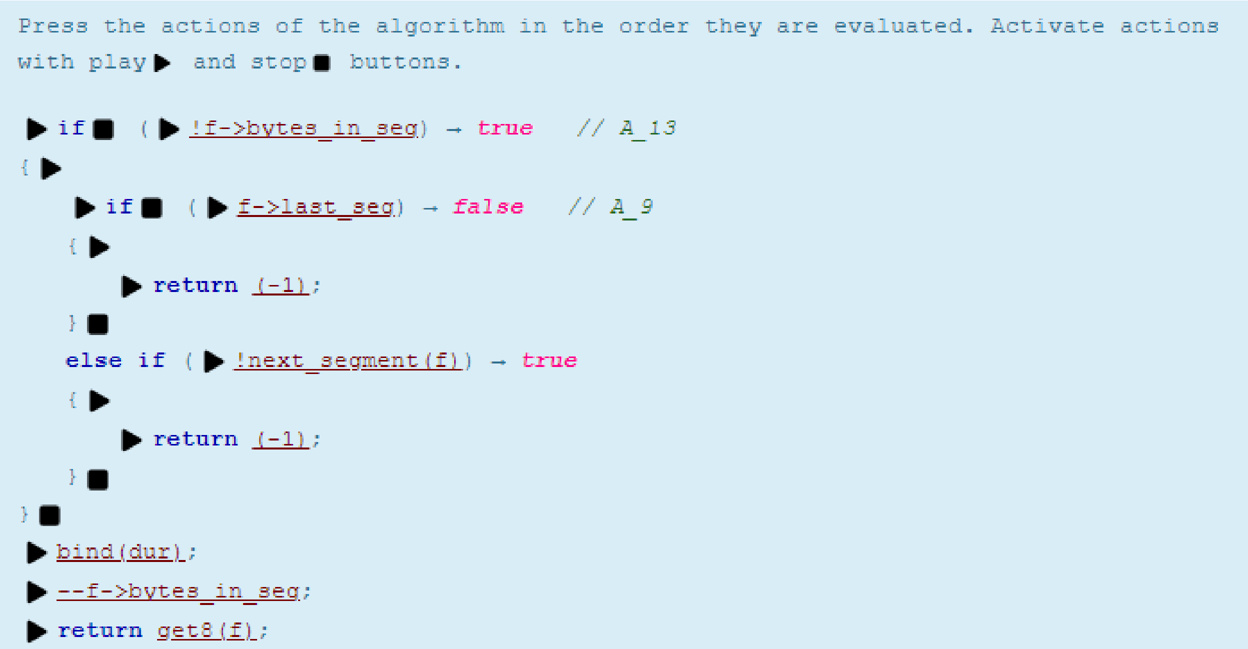


Press the actions of the algorithm in the order they are evaluated. Activate actions with play ▶ and stop ■ buttons.

```
▶ char *name = basename(prog);
▶ if ■  ( ▶ !strcmp(name, "btrfsdist")) → false   // A_18
{ ▶
    ▶ fs_type = BTRFS;
} ■
else if ( ▶ !strcmp(name, "ext4dist")) → true
{ ▶
    ▶ fs_type = EXT4;
} ■
else if ( ▶ !strcmp(name, "nfsdist"))
{ ▶
    ▶ fs_type = NFS;
} ■
else if ( ▶ !strcmp(name, "xfsdist"))
{ ▶
    ▶ fs_type = XFS;
} ■
```

Problem L-10 (difficulty: 0.3408)

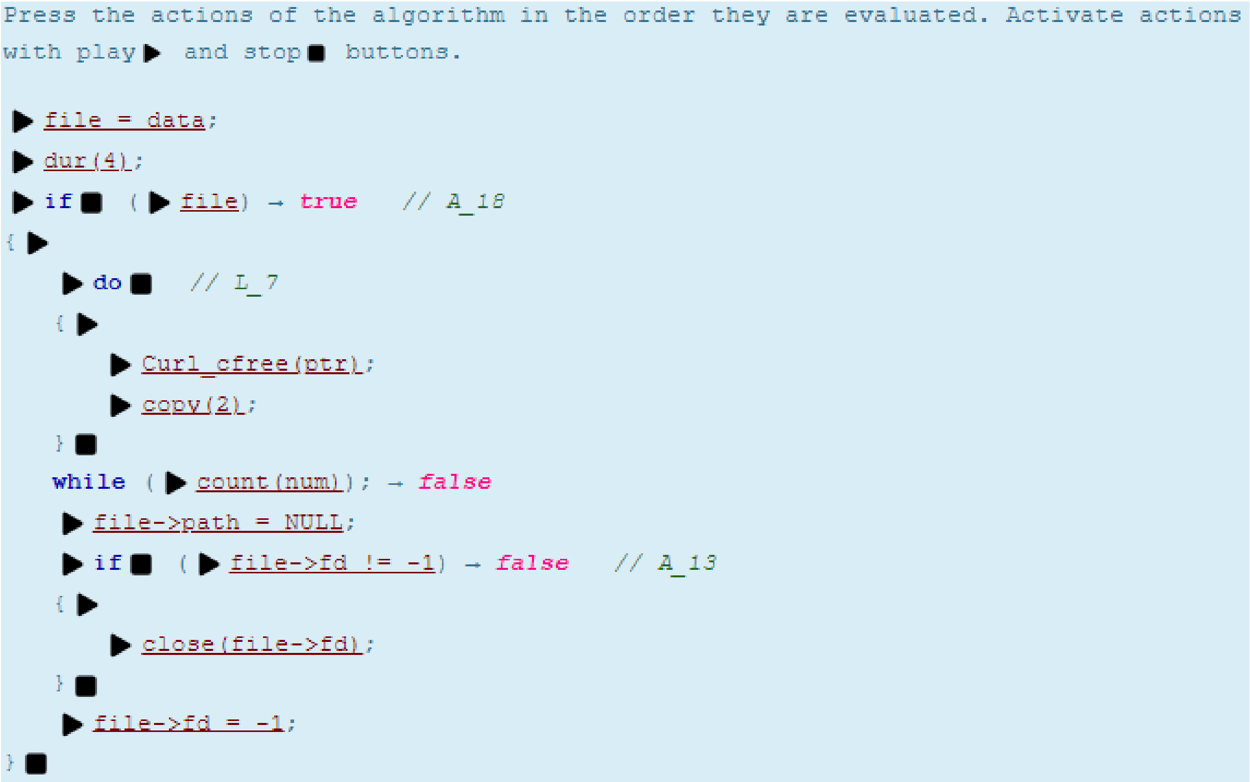
2. An exercise of medium (M) difficulty.



Press the actions of the algorithm in the order they are evaluated. Activate actions with play ▶ and stop ■ buttons.

```
▶ if ■  ( ▶ !f->bytes_in_seg) → true   // A_13
{ ▶
    ▶ if ■  ( ▶ f->last_seg) → false   // A_9
    { ▶
        ▶ return (-1);
    } ■
    else if ( ▶ !next_segment(f)) → true
    { ▶
        ▶ return (-1);
    } ■
} ■
▶ bind(dur);
▶ --f->bytes_in_seg;
▶ return get8(f);
```

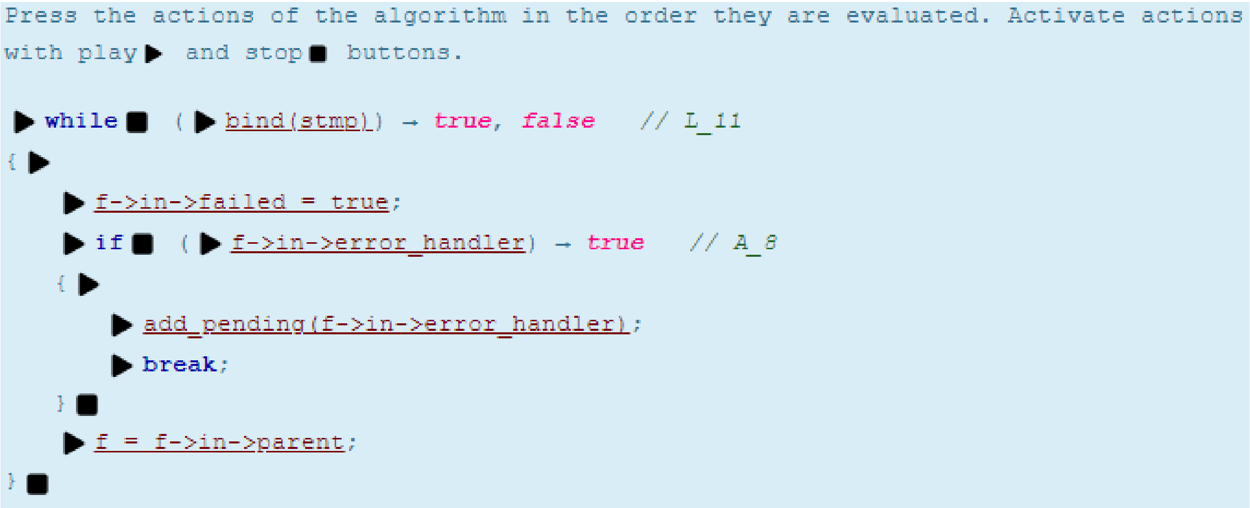
Problem M-1 (difficulty: 0.5279)



Press the actions of the algorithm in the order they are evaluated. Activate actions with play ▶ and stop ■ buttons.

```
▶ file = data;
▶ dur(4);
▶ if ■  (▶ file) → true   // A_18
{ ▶
    ▶ do ■   // L_7
    { ▶
        ▶ Curl_cfree(ptr);
        ▶ copy(2);
    } ■
    while (▶ count(num)); → false
    ▶ file->path = NULL;
    ▶ if ■  (▶ file->fd != -1) → false   // A_13
    { ▶
        ▶ close(file->fd);
    } ■
    ▶ file->fd = -1;
} ■
```

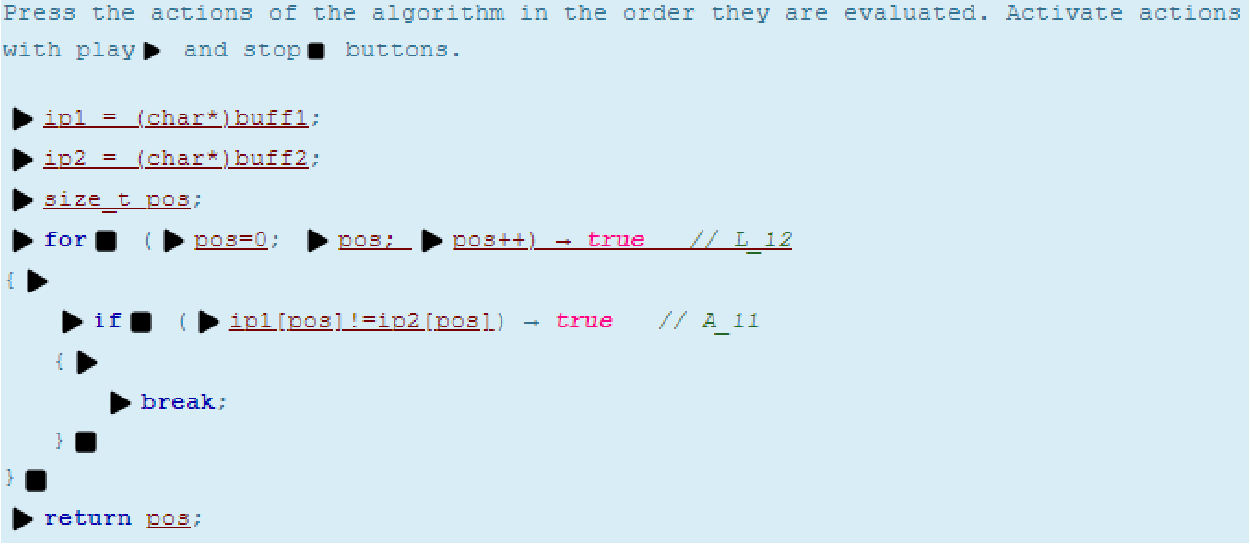
Problem M-2 (difficulty: 0.5963)



Press the actions of the algorithm in the order they are evaluated. Activate actions with play ▶ and stop ■ buttons.

```
▶ while ■  (▶ bind(stmp)) → true, false   // L_11
{ ▶
    ▶ f->in->failed = true;
    ▶ if ■  (▶ f->in->error_handler) → true   // A_8
    { ▶
        ▶ add_pending(f->in->error_handler);
        ▶ break;
    } ■
    ▶ f = f->in->parent;
} ■
```

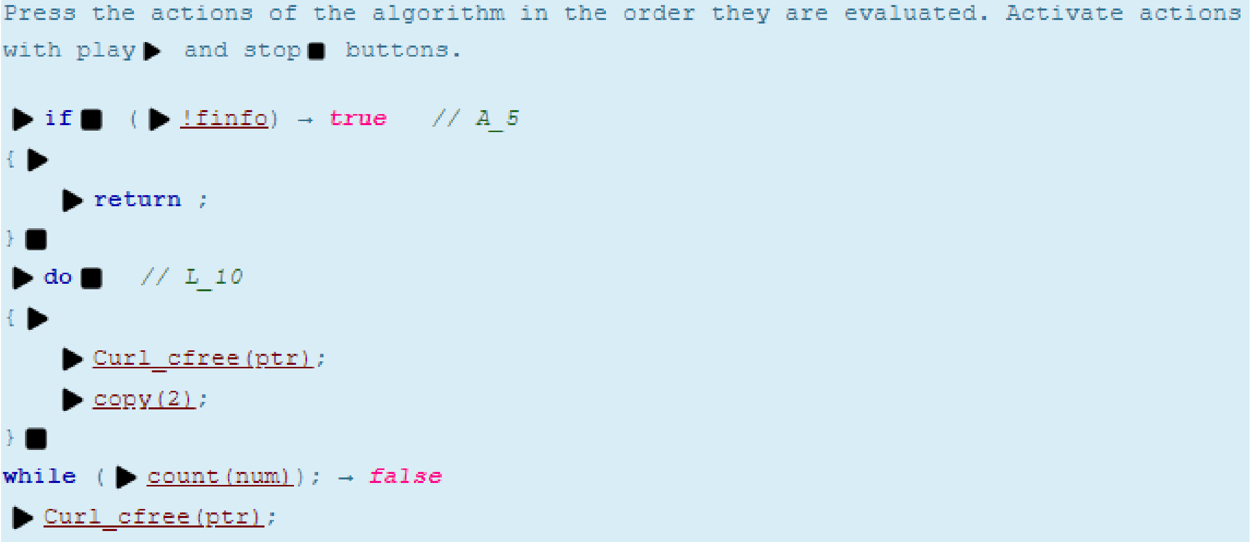
Problem M-3 (difficulty: 0.7178)



Press the actions of the algorithm in the order they are evaluated. Activate actions with play ▶ and stop ■ buttons.

```
▶ ip1 = (char*)buff1;
▶ ip2 = (char*)buff2;
▶ size_t pos;
▶ for ■  ( ▶ pos=0;  ▶ pos;  ▶ pos++) → true   // L_12
{ ▶
    ▶ if ■  ( ▶ ip1[pos]!=ip2[pos]) → true   // A_11
    { ▶
        ▶ break;
    } ■
} ■
▶ return pos;
```

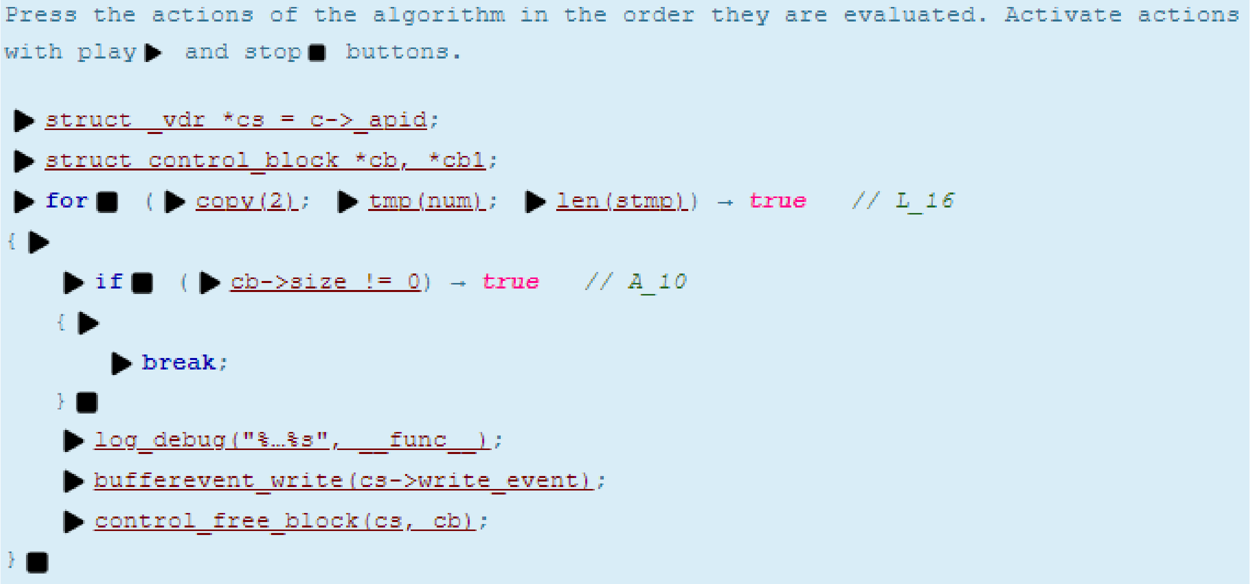
Problem M-4 (difficulty: 0.8692)



Press the actions of the algorithm in the order they are evaluated. Activate actions with play ▶ and stop ■ buttons.

```
▶ if ■  ( ▶ !finfo) → true   // A_5
{ ▶
    ▶ return ;
} ■
▶ do ■   // L_10
{ ▶
    ▶ Curl_cfree(ptr);
    ▶ copy(2);
} ■
while ( ▶ count(num)); → false
▶ Curl_cfree(ptr);
```

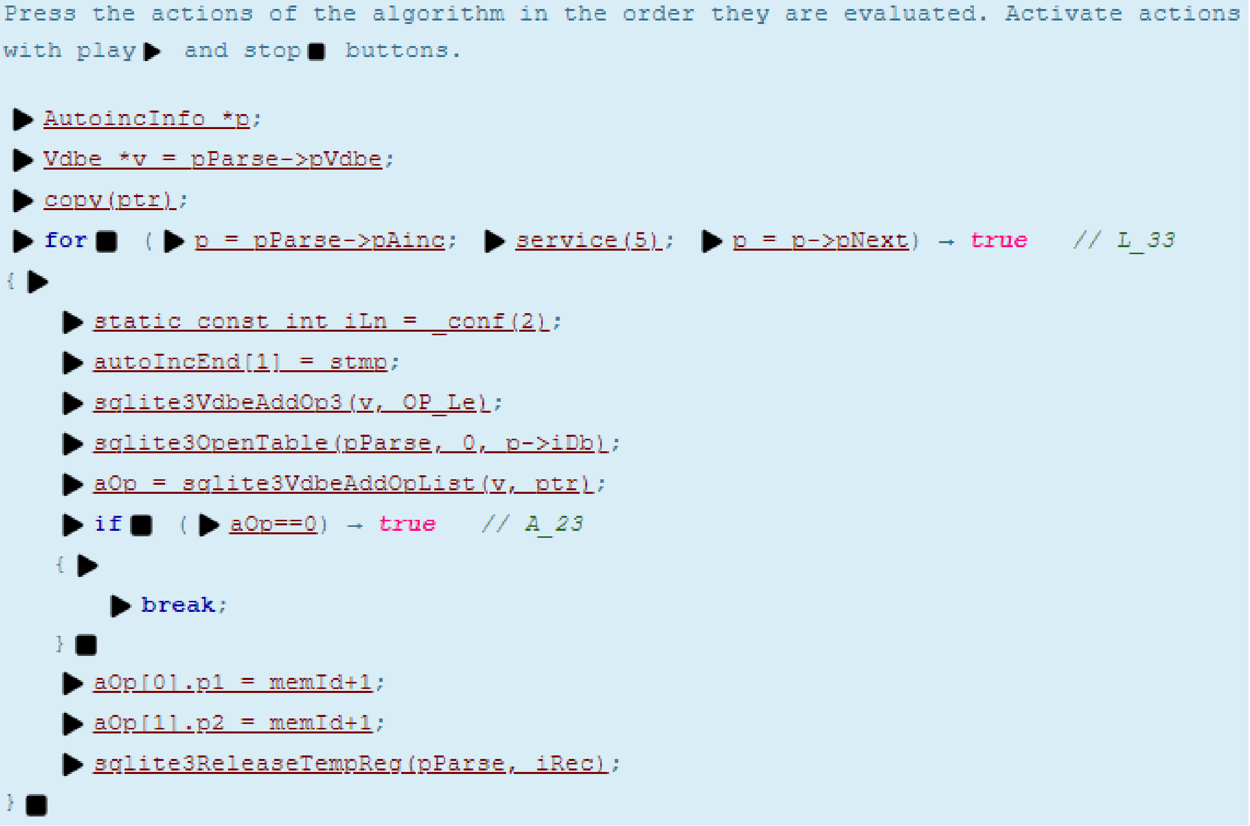
Problem M-5 (difficulty: 0.6568)



Press the actions of the algorithm in the order they are evaluated. Activate actions with play ▶ and stop ■ buttons.

```
▶ struct _vdr *cs = c->_apid;
▶ struct control_block *cb, *cb1;
▶ for ■  ( ▶ copy(2);  ▶ tmp(num);  ▶ len(stmp)) → true   // L_16
{ ▶
    ▶ if ■  ( ▶ cb->size != 0) → true   // A_10
    { ▶
        ▶ break;
    } ■
    ▶ log_debug("%…%s", __func__);
    ▶ bufferevent_write(cs->write_event);
    ▶ control_free_block(cs, cb);
} ■
```

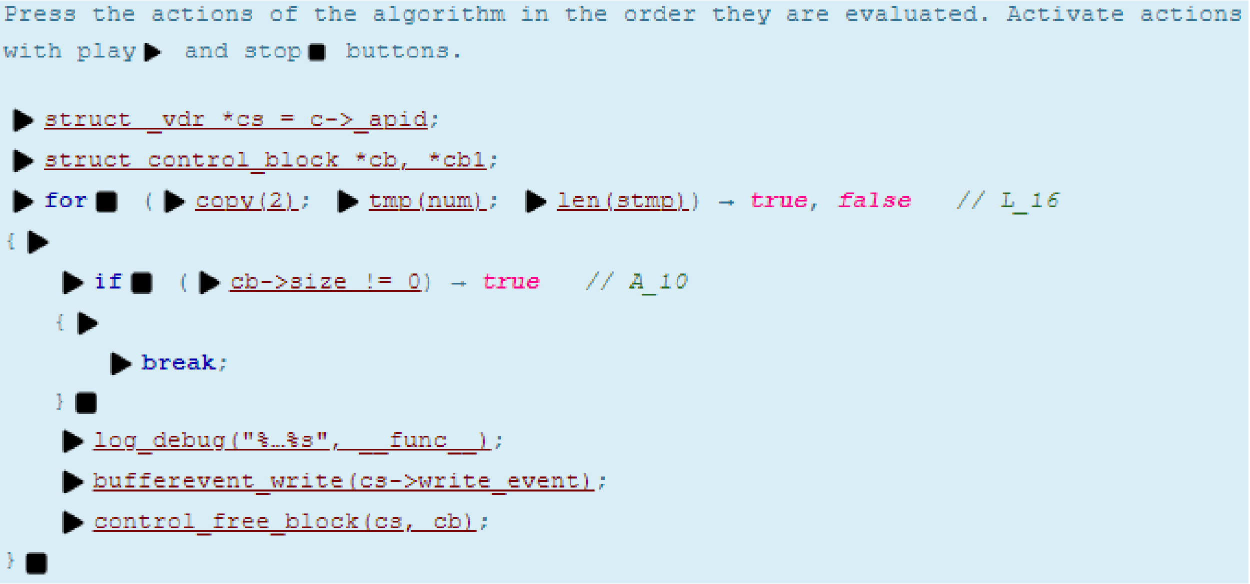
Problem M-6 (difficulty: 0.7706)



Press the actions of the algorithm in the order they are evaluated. Activate actions with play ▶ and stop ■ buttons.

```
▶ AutoincInfo *p;
▶ Vdbe *v = pParse->pVdbe;
▶ copy(ptr);
▶ for ■  ( ▶ p = pParse->pAinc;  ▶ service(5);  ▶ p = p->pNext) → true   // L_33
{ ▶
    ▶ static const int iLn = _conf(2);
    ▶ autoIncEnd[1] = stmp;
    ▶ sqlite3VdbeAddOp3(v, OP_Le);
    ▶ sqlite3OpenTable(pParse, 0, p->iDb);
    ▶ aOp = sqlite3VdbeAddOpList(v, ptr);
    ▶ if ■  ( ▶ aOp==0) → true   // A_23
    { ▶
        ▶ break;
    } ■
    ▶ aOp[0].p1 = memId+1;
    ▶ aOp[1].p2 = memId+1;
    ▶ sqlite3ReleaseTempReg(pParse, iRec);
} ■
```

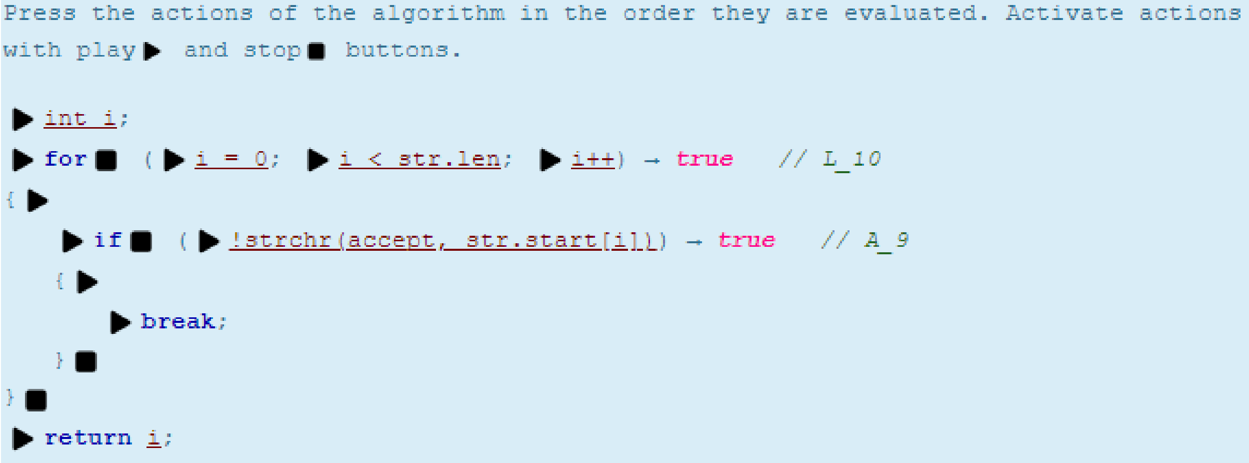
Problem M-7 (difficulty: 0.7305)



Press the actions of the algorithm in the order they are evaluated. Activate actions with play ▶ and stop ■ buttons.

```
▶ struct _vdr *cs = c->_apid;
▶ struct control_block *cb, *cb1;
▶ for ■  ( ▶ copy(2);  ▶ tmp(num);  ▶ len(stmp)) → true, false   // L_16
{ ▶
    ▶ if ■  ( ▶ cb->size != 0) → true   // A_10
    { ▶
        ▶ break;
    } ■
    ▶ log_debug("%…%s", __func__);
    ▶ bufferevent_write(cs->write_event);
    ▶ control_free_block(cs, cb);
} ■
```

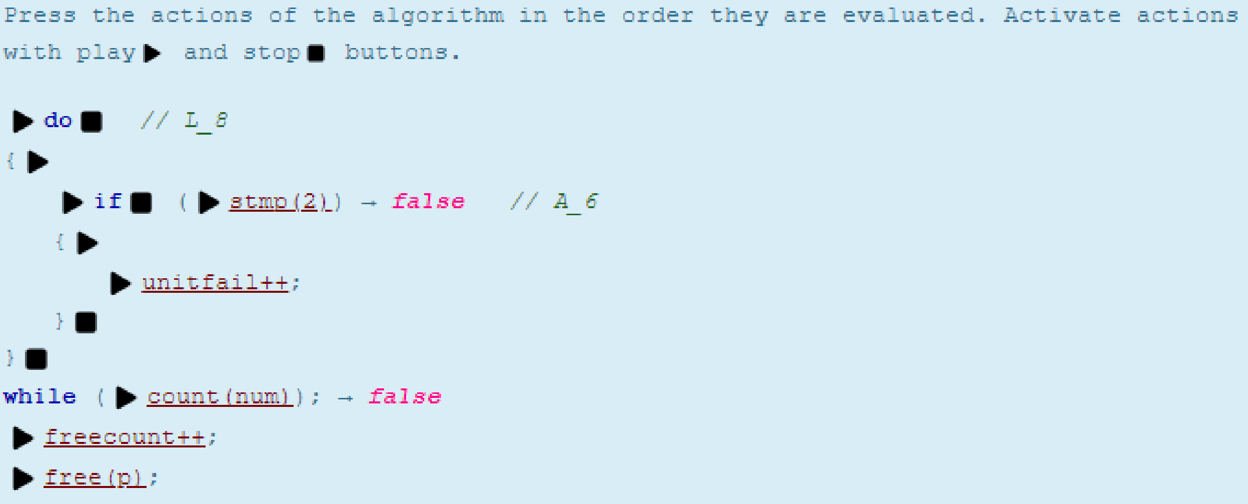
Problem M-8 (difficulty: 0.7706)



Press the actions of the algorithm in the order they are evaluated. Activate actions with play ▶ and stop ■ buttons.

```
▶ int i;
▶ for ■  ( ▶ i = 0;  ▶ i < str.len;  ▶ i++) → true   // L_10
{ ▶
    ▶ if ■  ( ▶ !strchr(accept, str.start[i])) → true   // A_9
    { ▶
        ▶ break;
    } ■
} ■
▶ return i;
```

Problem M-9 (difficulty: 0.9310)

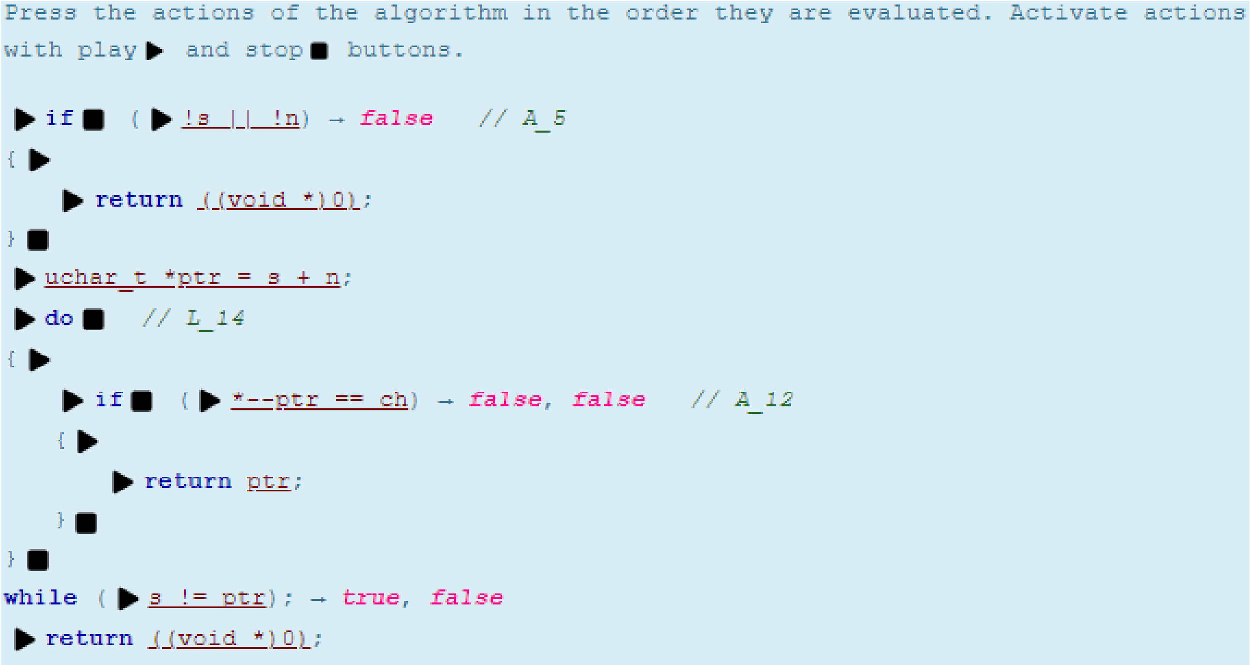


Press the actions of the algorithm in the order they are evaluated. Activate actions with play ▶ and stop ■ buttons.

```
▶ do ■   // L_8
{ ▶
    ▶ if ■  ( ▶ stmp(2)) → false   // A_6
    { ▶
        ▶ unitfail++;
    } ■
} ■
while  ( ▶ count(num)); → false
▶ freecount++;
▶ free(p);
```

Problem M-10 (difficulty: 0.5747)

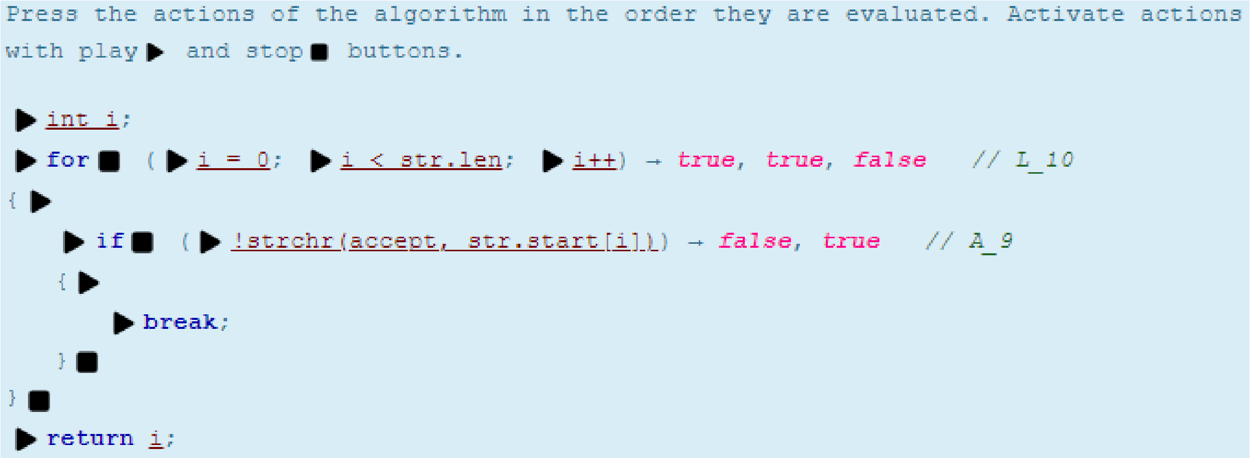
3. An exercise of high (H) difficulty.



Press the actions of the algorithm in the order they are evaluated. Activate actions with play ▶ and stop ■ buttons.

```
▶ if ■  ( ▶ !s || !n) → false   // A_5
{ ▶
    ▶ return ((void *)0);
} ■
▶ uchar_t *ptr = s + n;
▶ do ■   // L_14
{ ▶
    ▶ if ■  ( ▶ *--ptr == ch) → false, false   // A_12
    { ▶
        ▶ return ptr;
    } ■
} ■
while ( ▶ s != ptr); → true, false
▶ return ((void *)0);
```

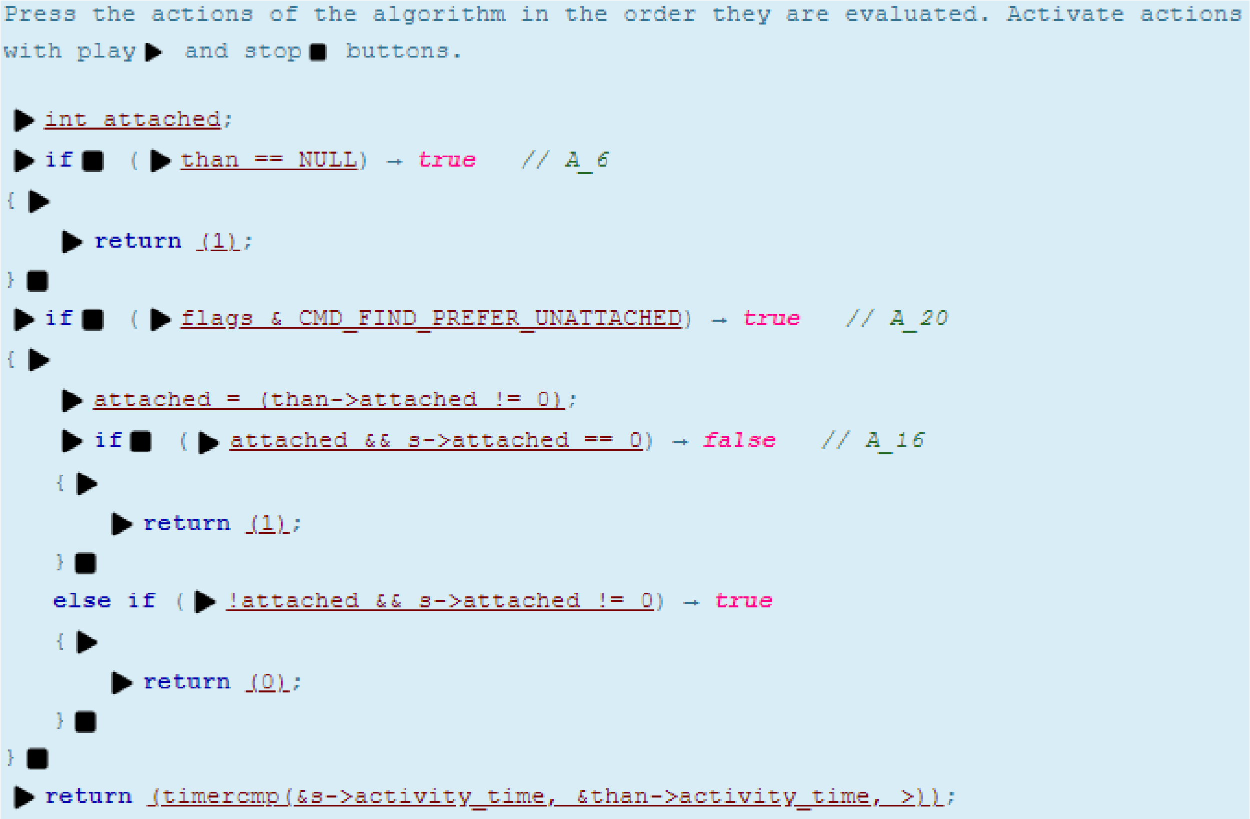
Problem H-1 (difficulty: 0.8557)



Press the actions of the algorithm in the order they are evaluated. Activate actions with play ▶ and stop ■ buttons.

```
▶ int i;
▶ for ■  ( ▶ i = 0;  ▶ i < str.len;  ▶ i++) → true, true, false   // L_10
{ ▶
    ▶ if ■  ( ▶ !strchr(accept, str.start[i])) → false, true   // A_9
    { ▶
        ▶ break;
    } ■
} ■
▶ return i;
```

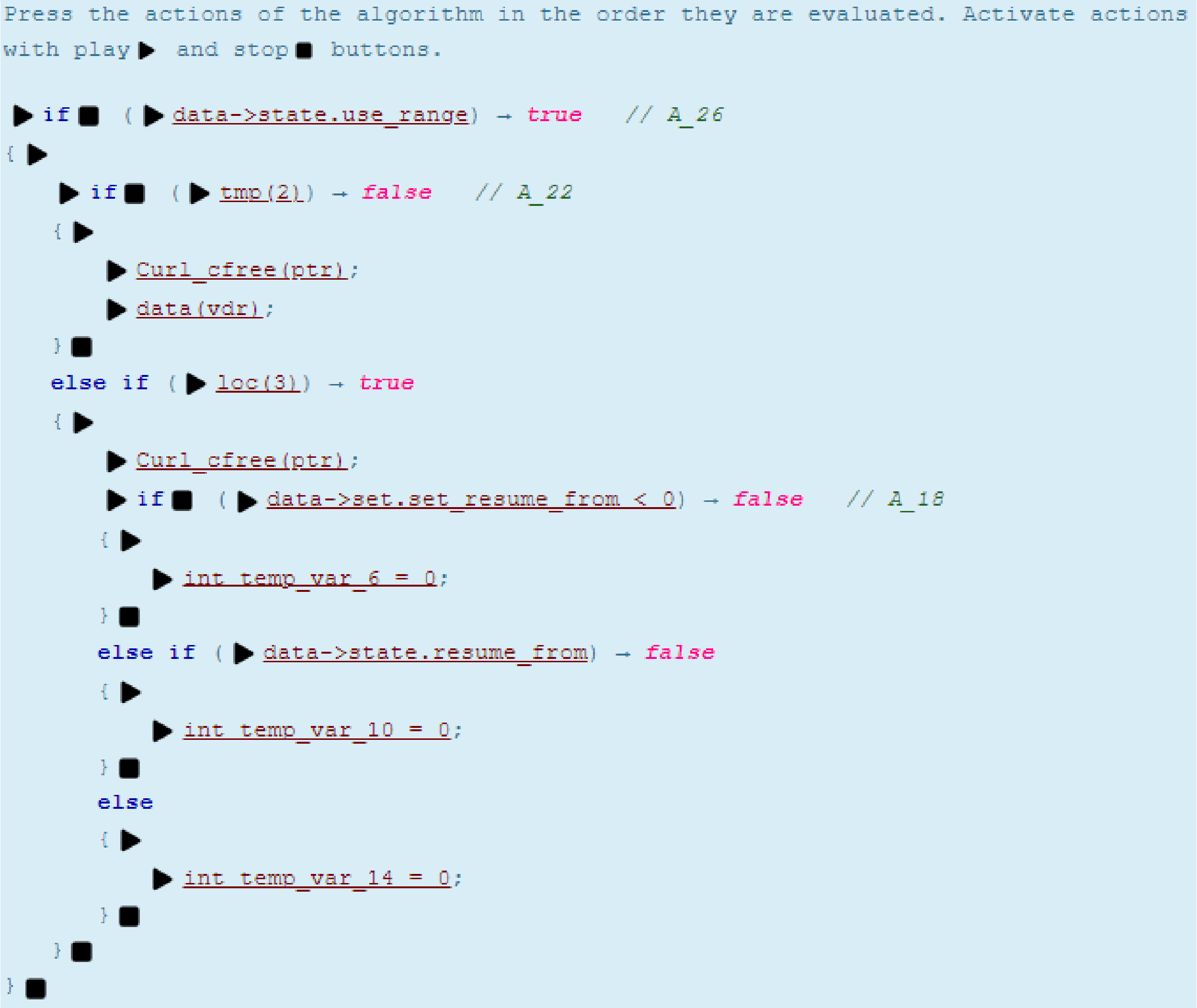
Problem H-2 (difficulty: 0.9310)



Press the actions of the algorithm in the order they are evaluated. Activate actions with play ▶ and stop ■ buttons.

```
▶ int attached;
▶ if ■ ( ▶ than == NULL) → true   // A_6
{ ▶
    ▶ return (1);
} ■
▶ if ■ ( ▶ flags & CMD_FIND_PREFER_UNATTACHED) → true   // A_20
{ ▶
    ▶ attached = (than->attached != 0);
    ▶ if ■ ( ▶ attached && s->attached == 0) → false   // A_16
    { ▶
        ▶ return (1);
    } ■
    else if ( ▶ !attached && s->attached != 0) → true
    { ▶
        ▶ return (0);
    } ■
} ■
▶ return (timercmp(&s->activity_time, &than->activity_time, >));
```

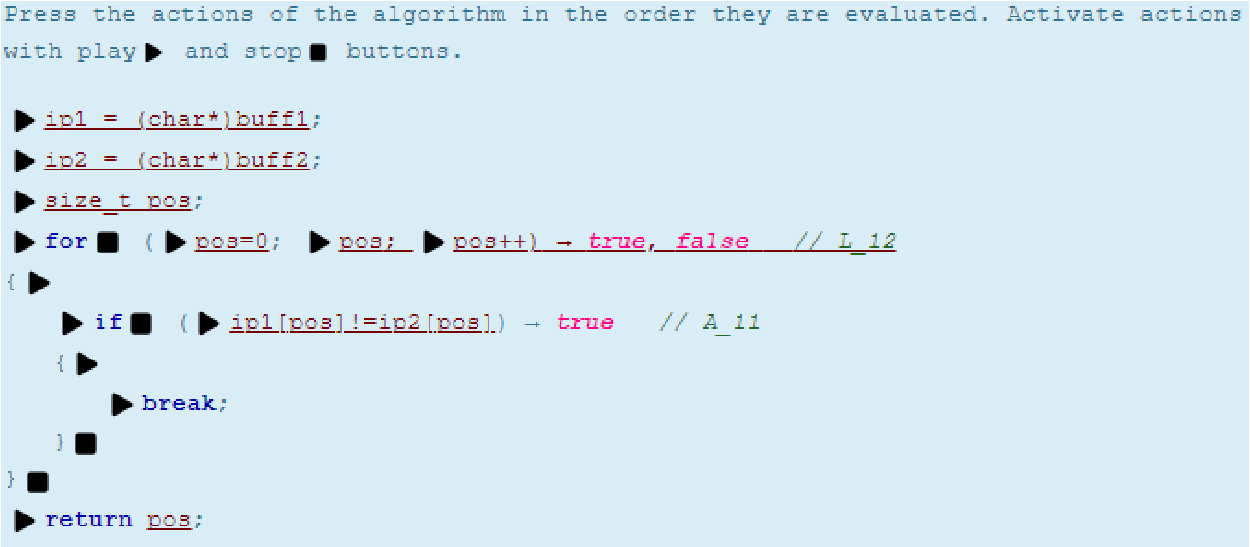
Problem H-3 (difficulty: 0.6779)



Press the actions of the algorithm in the order they are evaluated. Activate actions with play ▶ and stop ■ buttons.

```
▶ if ■ ( ▶ data->state.use_range) → true    // A_26
{ ▶
    ▶ if ■ ( ▶ tmp(2)) → false    // A_22
    { ▶
        ▶ Curl_cfree(ptr);
        ▶ data(vdr);
    } ■
    else if ( ▶ loc(3)) → true
    { ▶
        ▶ Curl_cfree(ptr);
        ▶ if ■ ( ▶ data->set.set_resume_from < 0) → false    // A_18
        { ▶
            ▶ int temp_var_6 = 0;
        } ■
        else if ( ▶ data->state.resume_from) → false
        { ▶
            ▶ int temp_var_10 = 0;
        } ■
        else
        { ▶
            ▶ int temp_var_14 = 0;
        } ■
    } ■
} ■
```

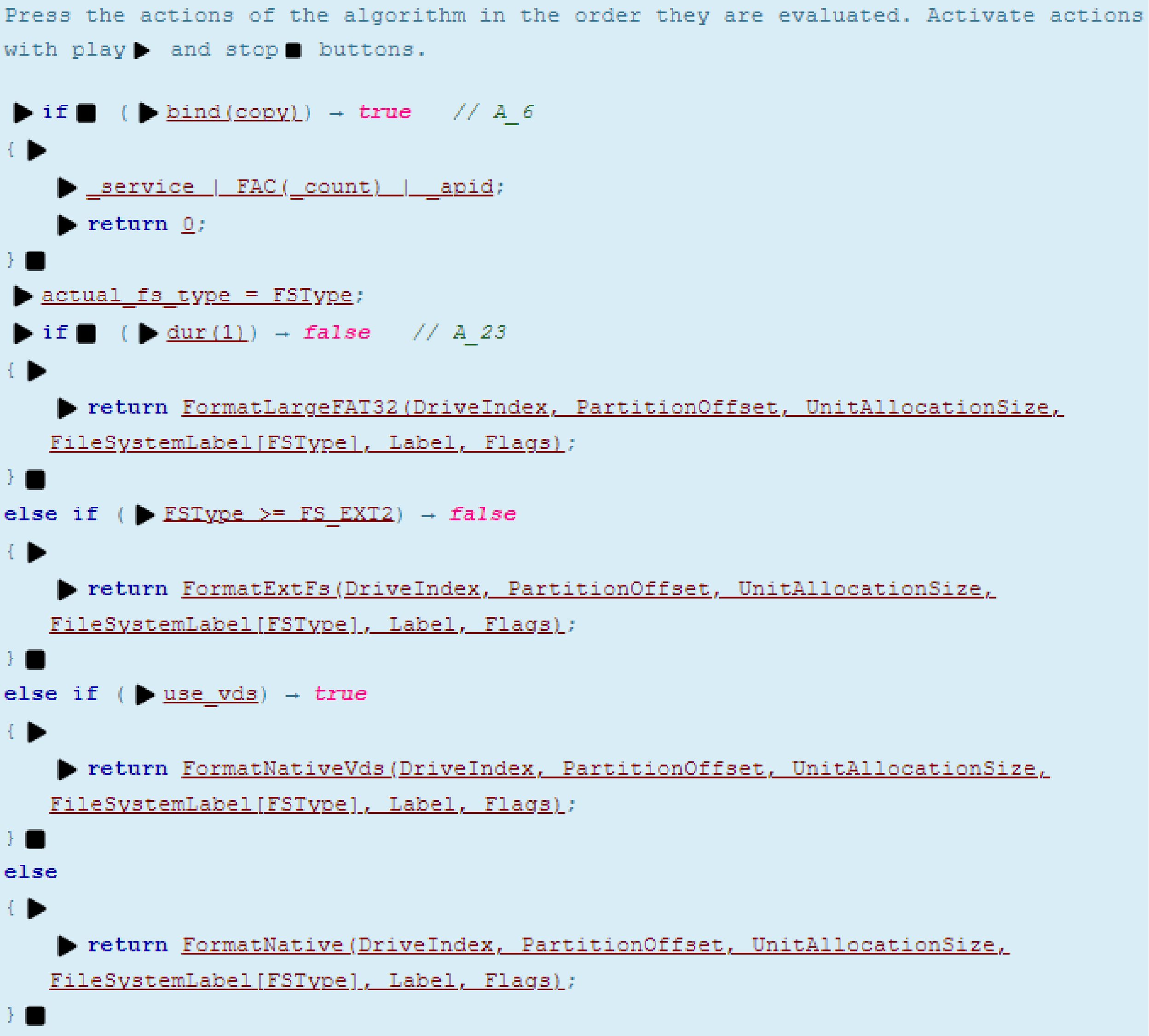
Problem H-4 (difficulty: 0.7229)



Press the actions of the algorithm in the order they are evaluated. Activate actions with play ▶ and stop ■ buttons.

```
▶ ip1 = (char*)buff1;
▶ ip2 = (char*)buff2;
▶ size_t pos;
▶ for ■ ( ▶ pos=0; ▶ pos; ▶ pos++) → true, false    // L_12
{ ▶
    ▶ if ■ ( ▶ ip1[pos]!=ip2[pos]) → true    // A_11
    { ▶
        ▶ break;
    } ■
} ■
▶ return pos;
```

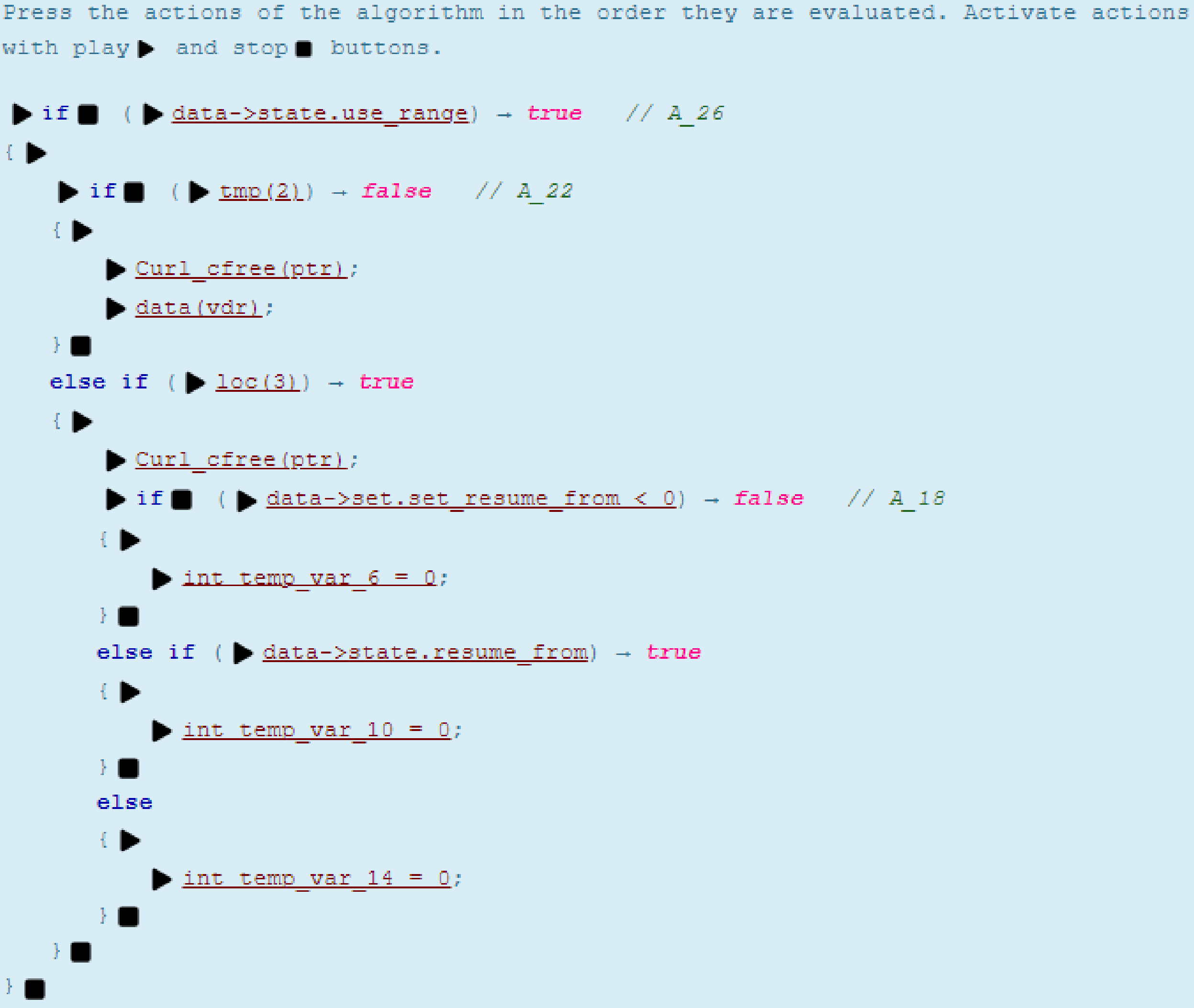
Problem H-5 (difficulty: 0.8692)



Press the actions of the algorithm in the order they are evaluated. Activate actions with play ▶ and stop ■ buttons.

```
▶ if ■  ( ▶ bind(copy)) → true   // A_6
{ ▶
    ▶ _service | FAC(_count) | _apid;
    ▶ return 0;
} ■
▶ actual_fs_type = FSType;
▶ if ■  ( ▶ dur(1)) → false   // A_23
{ ▶
    ▶ return FormatLargeFAT32(DriveIndex, PartitionOffset, UnitAllocationSize,
   FileSystemLabel[FSType], Label, Flags);
} ■
else if ( ▶ FSType >= FS_EXT2) → false
{ ▶
    ▶ return FormatExtFs(DriveIndex, PartitionOffset, UnitAllocationSize,
   FileSystemLabel[FSType], Label, Flags);
} ■
else if ( ▶ use_vds) → true
{ ▶
    ▶ return FormatNativeVds(DriveIndex, PartitionOffset, UnitAllocationSize,
   FileSystemLabel[FSType], Label, Flags);
} ■
else
{ ▶
    ▶ return FormatNative(DriveIndex, PartitionOffset, UnitAllocationSize,
   FileSystemLabel[FSType], Label, Flags);
} ■
```

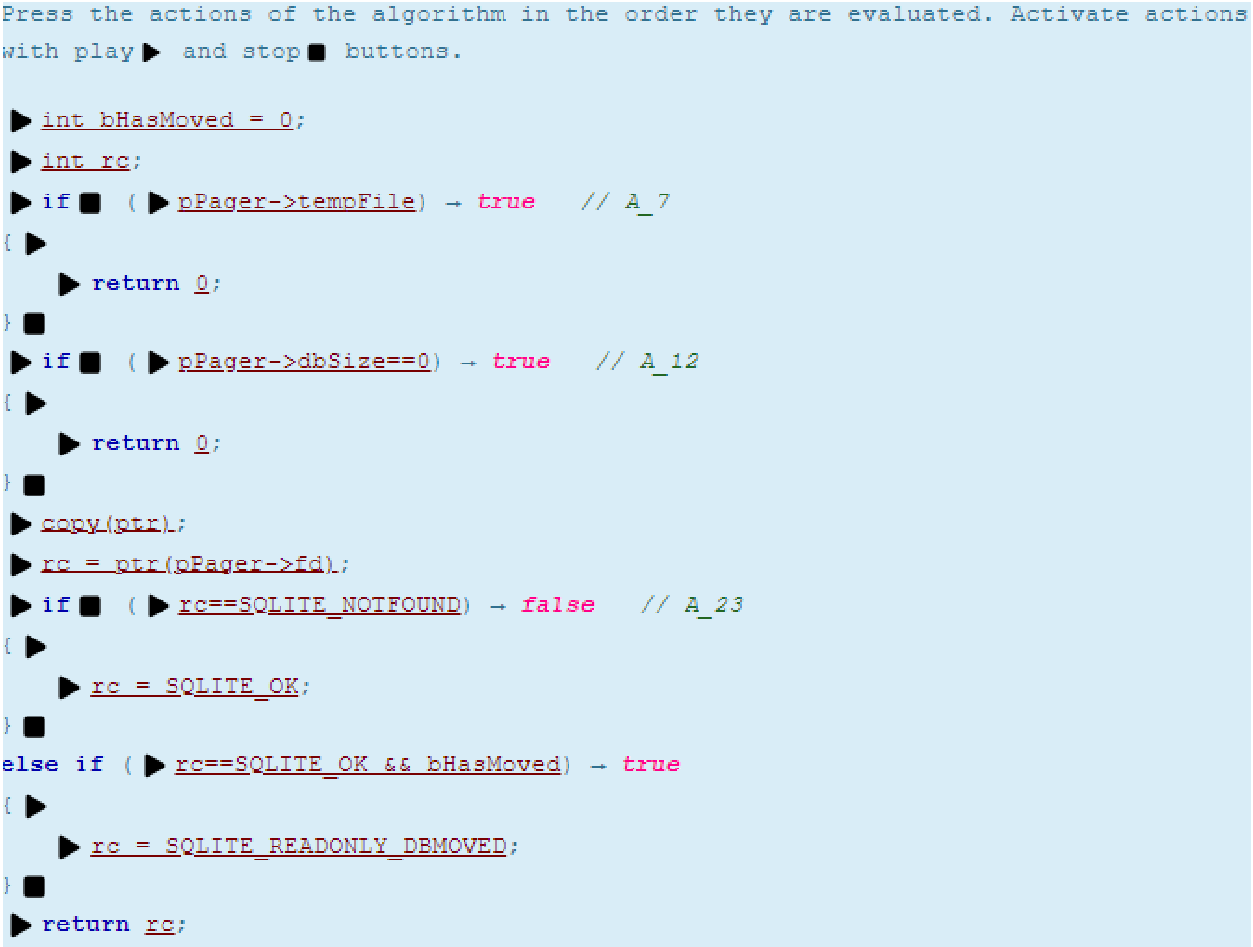
Problem H-6 (difficulty: 0.7837)



Press the actions of the algorithm in the order they are evaluated. Activate actions with play ▶ and stop ■ buttons.

```
▶ if ■  ( ▶ data->state.use_range) → true   // A_26
{ ▶
    ▶ if ■  ( ▶ tmp(2)) → false   // A_22
    { ▶
        ▶ Curl_cfree(ptr);
        ▶ data(vdr);
    } ■
    else if ( ▶ loc(3)) → true
    { ▶
        ▶ Curl_cfree(ptr);
        ▶ if ■  ( ▶ data->set.set_resume_from < 0) → false   // A_18
        { ▶
            ▶ int temp_var_6 = 0;
        } ■
        else if ( ▶ data->state.resume_from) → true
        { ▶
            ▶ int temp_var_10 = 0;
        } ■
        else
        { ▶
            ▶ int temp_var_14 = 0;
        } ■
    } ■
} ■
```

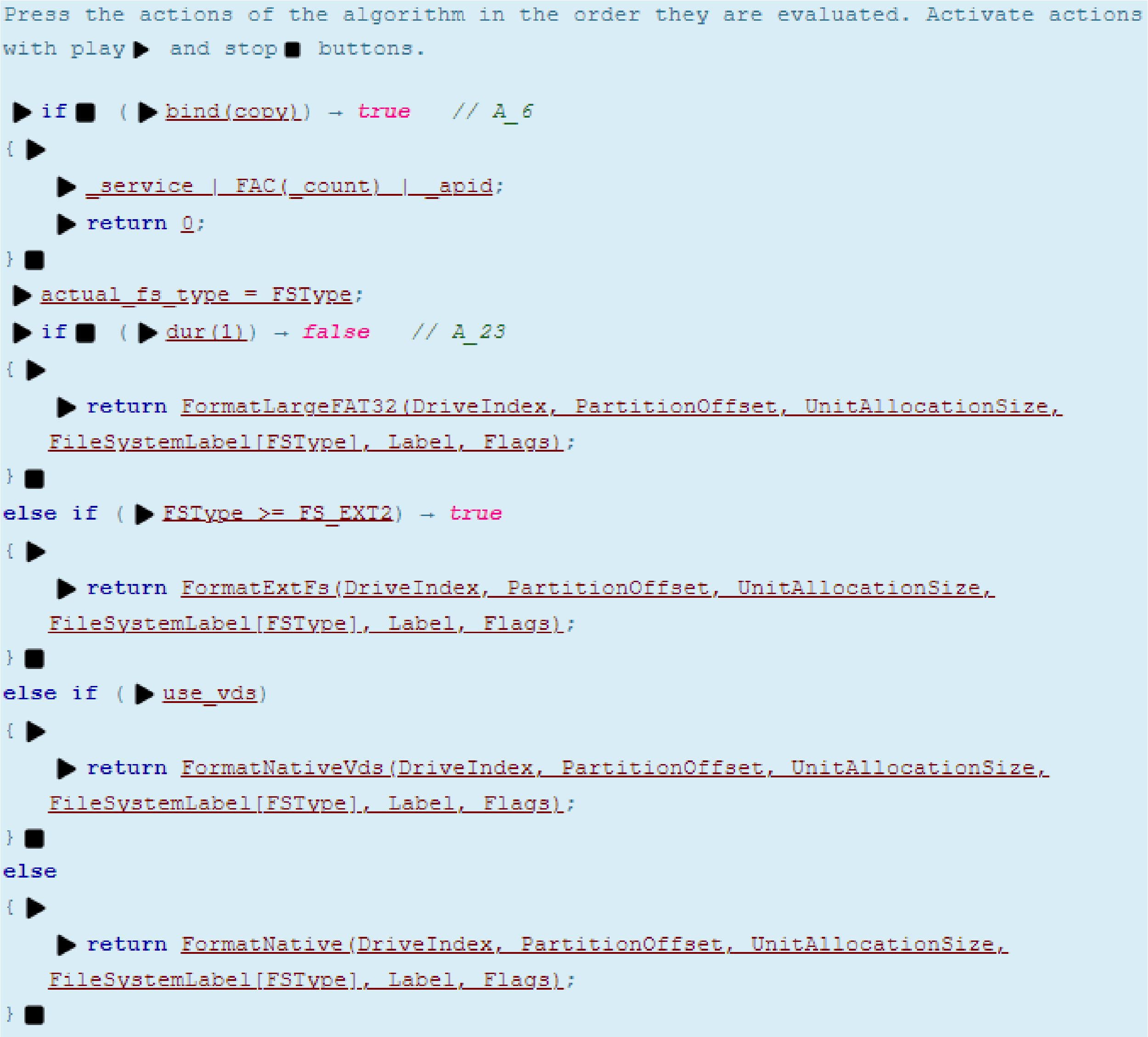
Problem H-7 (difficulty: 0.7229)



Press the actions of the algorithm in the order they are evaluated. Activate actions with play ▶ and stop ■ buttons.

```
▶ int bHasMoved = 0;
▶ int rc;
▶ if ■  ( ▶ pPager->tempFile) → true   // A_7
{ ▶
    ▶ return 0;
} ■
▶ if ■  ( ▶ pPager->dbSize==0) → true   // A_12
{ ▶
    ▶ return 0;
} ■
▶ copy(ptr);
▶ rc = ptr(pPager->fd);
▶ if ■  ( ▶ rc==SQLITE_NOTFOUND) → false   // A_23
{ ▶
    ▶ rc = SQLITE_OK;
} ■
else if ( ▶ rc==SQLITE_OK && bHasMoved) → true
{ ▶
    ▶ rc = SQLITE_READONLY_DBMOVED;
} ■
▶ return rc;
```

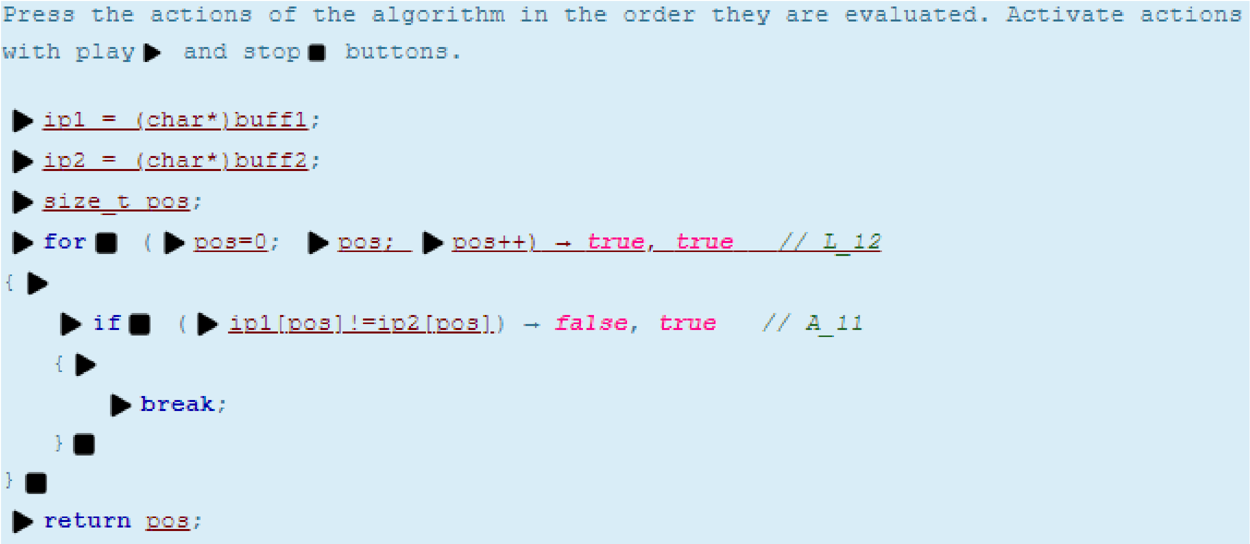
Problem H-8 (difficulty: 0.7487)



Press the actions of the algorithm in the order they are evaluated. Activate actions with play ▶ and stop ■ buttons.

```
▶ if ■ ( ▶ bind(copy)) → true   // A_6
{ ▶
    ▶ _service | FAC(_count) | _apid;
    ▶ return 0;
} ■
▶ actual_fs_type = FSType;
▶ if ■ ( ▶ dur(1)) → false   // A_23
{ ▶
    ▶ return FormatLargeFAT32(DriveIndex, PartitionOffset, UnitAllocationSize,
  FileSystemLabel[FSType], Label, Flags);
} ■
else if ( ▶ FSType >= FS_EXT2) → true
{ ▶
    ▶ return FormatExtFs(DriveIndex, PartitionOffset, UnitAllocationSize,
  FileSystemLabel[FSType], Label, Flags);
} ■
else if ( ▶ use_vds)
{ ▶
    ▶ return FormatNativeVds(DriveIndex, PartitionOffset, UnitAllocationSize,
  FileSystemLabel[FSType], Label, Flags);
} ■
else
{ ▶
    ▶ return FormatNative(DriveIndex, PartitionOffset, UnitAllocationSize,
  FileSystemLabel[FSType], Label, Flags);
} ■
```

Problem H-9 (difficulty: 0.7178)



Press the actions of the algorithm in the order they are evaluated. Activate actions with play ▶ and stop ■ buttons.

```
▶ ip1 = (char*)buff1;
▶ ip2 = (char*)buff2;
▶ size_t pos;
▶ for ■ ( ▶ pos=0; ▶ pos; ▶ pos++) → true, true   // L_12
{ ▶
    ▶ if ■ ( ▶ ip1[pos]!=ip2[pos]) → false, true   // A_11
    { ▶
        ▶ break;
    } ■
} ■
▶ return pos;
```

Problem H-10 (difficulty: 0.9433)