```
/* This file was generated by the Hex-Rays decompiler version 7.7.0.220118.
   Copyright (c) 2007-2021 Hex-Rays <info@hex-rays.com>

   Detected compiler: GNU C++
*/

#include <math.h>
#include <defs.h>


//-------------------------------------------------------------------------
// Function declarations

__int64 (**init_proc())(void);
__int64 __fastcall sub_400740(); // weak
// void free(void *ptr);
// char *strcpy(char *dest, const char *src);
// int puts(const char *s);
// double atof(const char *nptr);
// size_t strlen(const char *s);
// int printf(const char *format, ...);
// double pow(double x, double y);
// void srand(unsigned int seed);
// int strcmp(const char *s1, const char *s2);
// void *malloc(size_t size);
// int atoi(const char *nptr);
// __int64 __isoc99_scanf(const char *, ...); weak
// int rand(void);
// const unsigned __int16 **__ctype_b_loc(void);
void __fastcall __noreturn start(__int64 a1, __int64 a2, void (*a3)(void));
char *deregister_tm_clones();
__int64 __fastcall register_tm_clones(); // weak
char *_do_global_dtors_aux();
__int64 __fastcall frame_dummy(_QWORD, _QWORD, _QWORD); // weak
__int64 __fastcall is_number(const char *a1);
_BOOL8 __fastcall isOctal(__int64 a1);
__int64 __fastcall isValidOctal(const char *a1);
__int64 __fastcall is_operator_char(char a1);
__int64 __fastcall is_operator(const char *a1);
double __fastcall check_limit(double result);
double __fastcall convertOctalToDecimal(int a1);
__int64 __fastcall op_precidence(const char *a1);
__int64 __fastcall processToken(__int64 a1, __int64 a2);
__int64 __fastcall processLine(char *a1, __int64 a2);
void shunt();
__int64 __fastcall processCommand(const char *a1, __int64 a2);
int __fastcall processNumber(const char *a1, __int64 a2);
int __fastcall processOperator(const char *a1, __int64 a2);
__int64 run_srpn();
int __cdecl main(int argc, const char **argv, const char **envp);
__int64 __fastcall push(__int64 a1, double a2);
__int64 __fastcall pop(double *a1);
double __fastcall popfromstack(double *a1, __int64 a2);
double __fastcall peek(_QWORD *a1);
int __fastcall print_stack(__int64 a1);
void _libc_csu_fini(void); // idb
void term_proc();
// int puts(const char *s);
```

```
// double atof(const char *nptr);
// int __fastcall _libc_start_main(int (__fastcall *main)(int, char **, char **),
int argc, char **ubp_av, void (*init)(void), void (*fini)(void), void
(*rtld_fini)(void), void *stack_end);
// int strcmp(const char *s1, const char *s2);
// int atoi(const char *nptr);
// int rand(void);
// const unsigned __int16 **_ctype_b_loc(void);
// __int64 _gmon_start__(void); weak

//-------------------------------------------------------------------------
// Data declarations

_UNKNOWN _libc_csu_init;
const char asc_401DA8[] = "-\x00+\x00*\x00/\x00%\x00^"; // idb
__int64 (__fastcall *_frame_dummy_init_array_entry)() = &frame_dummy; // weak
__int64 (__fastcall *_do_global_dtors_aux_fini_array_entry)() =
&_do_global_dtors_aux; // weak
__int64 (*qword_603010)(void) = NULL; // weak
char *OPERATORS = "-\x00+\x00*\x00/\x00%\x00^"; // idb
char _bss_start; // weak
int rand_count; // weak
int stack_count; // weak


//----- (0000000000400720) ----------------------------------------------------
__int64 (**init_proc())(void)
{
  __int64 (**result)(void); // rax

  result = &_gmon_start__;
  if ( &_gmon_start__ )
    return (__int64 (**)(void))_gmon_start__();
  return result;
}
// 603180: using guessed type __int64 _gmon_start__(void);

//----- (0000000000400740) ----------------------------------------------------
__int64 sub_400740()
{
  return qword_603010();
}
// 400740: using guessed type __int64 __fastcall sub_400740();
// 603010: using guessed type __int64 (*qword_603010)(void);

//----- (0000000000400830) ----------------------------------------------------
// positive sp value has been detected, the output may be wrong!
void __fastcall __noreturn start(__int64 a1, __int64 a2, void (*a3)(void))
{
  __int64 v3; // rax
  int v4; // esi
  __int64 v5; // [rsp-8h] [rbp-8h] BYREF
  char *retaddr; // [rsp+0h] [rbp+0h] BYREF

  v4 = v5;
  v5 = v3;
  _libc_start_main(
    (int (__fastcall *)(int, char **, char **))main,
```

```
    v4,
    &retaddr,
    (void (*)(void))_libc_csu_init,
    _libc_csu_fini,
    a3,
    &v5);
  __halt();
}
// 400836: positive sp value 8 has been found
// 40083D: variable 'v3' is possibly undefined

//----- (0000000000400860) ----------------------------------------------------
char *deregister_tm_clones()
{
  return &_bss_start;
}
// 6030F0: using guessed type char _bss_start;

//----- (0000000000400890) ----------------------------------------------------
__int64 register_tm_clones()
{
  return 0LL;
}
// 400890: using guessed type __int64 __fastcall register_tm_clones();

//----- (00000000004008D0) ----------------------------------------------------
char *_do_global_dtors_aux()
{
  char *result; // rax

  if ( !_bss_start )
  {
    result = deregister_tm_clones();
    _bss_start = 1;
  }
  return result;
}
// 6030F0: using guessed type char _bss_start;

//----- (0000000000400902) ----------------------------------------------------
__int64 __fastcall is_number(const char *a1)
{
  int v2; // [rsp+10h] [rbp-10h]
  int i; // [rsp+14h] [rbp-Ch]
  unsigned __int8 v4; // [rsp+1Fh] [rbp-1h]

  v4 = 1;
  v2 = strlen(a1);
  if ( v2 <= 1 )
  {
    if ( ((*__ctype_b_loc())[*a1] & 0x800) == 0 )
      return 0;
  }
  else
  {
    for ( i = *a1 == 45; i < v2; ++i )
    {
      if ( ((*__ctype_b_loc())[a1[i]] & 0x800) == 0 )
```

```
        v4 = 0;
    }
  }
  return v4;
}

//----- (00000000004009BD) ----------------------------------------------------
_BOOL8 __fastcall isOctal(__int64 a1)
{
  if ( (int)strlen((const char *)a1) <= 2 )
    return 0LL;
  if ( *(_BYTE *)a1 == 48 )
    return 1LL;
  return *(_BYTE *)a1 == 45 && *(_BYTE *)(a1 + 1) == 48;
}

//----- (0000000000400A14) ----------------------------------------------------
__int64 __fastcall isValidOctal(const char *a1)
{
  int v2; // [rsp+14h] [rbp-Ch]
  int i; // [rsp+18h] [rbp-8h]

  v2 = strlen(a1);
  for ( i = *a1 == 45; i < v2; ++i )
  {
    if ( a1[i] <= 47 )
      return 0LL;
    if ( a1[i] > 55 )
      return 0LL;
  }
  return 1LL;
}

//----- (0000000000400A9F) ----------------------------------------------------
__int64 __fastcall is_operator_char(char a1)
{
  char v2[2]; // [rsp+1Eh] [rbp-2h] BYREF

  v2[0] = a1;
  v2[1] = 0;
  return is_operator(v2);
}

//----- (0000000000400AC5) ----------------------------------------------------
__int64 __fastcall is_operator(const char *a1)
{
  int i; // [rsp+1Ch] [rbp-4h]

  if ( !strcmp(a1, L"rd=") )
    return 1LL;
  if ( !strcmp(a1, L"d=") )
    return 1LL;
  if ( !strcmp(a1, L"=") )
    return 1LL;
  for ( i = 0; i < 6; ++i )
  {
    if ( !strcmp(a1, (&OPERATORS)[i]) )
      return 1LL;
```

```
  }
  return 0LL;
}

//----- (0000000000400B6A) ----------------------------------------------------
double __fastcall check_limit(double result)
{
  if ( result > 2147483647.0 )
    return 2147483647.0;
  if ( result < -2147483648.0 )
    return -2147483648.0;
  return result;
}

//----- (0000000000400BAC) ----------------------------------------------------
double __fastcall convertOctalToDecimal(int a1)
{
  int v3; // [rsp+14h] [rbp-Ch]
  double v4; // [rsp+18h] [rbp-8h]

  v4 = 0.0;
  v3 = 0;
  while ( a1 )
  {
    v4 = pow(8.0, (double)v3++) * (double)(a1 % 10) + v4;
    a1 /= 10;
  }
  return v4;
}

//----- (0000000000400C4D) ----------------------------------------------------
__int64 __fastcall op_precidence(const char *a1)
{
  int i; // [rsp+1Ch] [rbp-4h]

  for ( i = 0; i < 6; ++i )
  {
    if ( !strcmp(a1, (&OPERATORS)[i]) )
      return (unsigned int)i;
  }
  return 0xFFFFFFFFLL;
}

//----- (0000000000400C9C) ----------------------------------------------------
__int64 __fastcall processToken(__int64 a1, __int64 a2)
{
  if ( (unsigned __int8)is_number(a1) || (unsignedrun)sr __int8)is_operator(a1) )
    return processCommand(a1, a2);
  else
    return processLine(a1, a2);
}
// 400902: using guessed type __int64 __fastcall is_number(_QWORD);
// 400AC5: using guessed type __int64 __fastcall is_operator(_QWORD);
// 400D0C: using guessed type __int64 __fastcall processLine(_QWORD, _QWORD);
// 4014DD: using guessed type __int64 __fastcall processCommand(_QWORD, _QWORD);

//----- (0000000000400D0C) ----------------------------------------------------
__int64 __fastcall processLine(char *a1, __int64 a2)
```

```
{
  __int64 v2; // r12
  void *v3; // rsp
  void *v4; // rsp
  void *v5; // rsp
  __int64 result; // rax
  __int64 v7[16]; // [rsp+0h] [rbp-130h] BYREF
  __int64 v8; // [rsp+80h] [rbp-B0h]
  char *s; // [rsp+88h] [rbp-A8h]
  char v10[2]; // [rsp+90h] [rbp-A0h] BYREF
  char v11[2]; // [rsp+92h] [rbp-9Eh] BYREF
  int v12; // [rsp+94h] [rbp-9Ch]
  __int64 *v13; // [rsp+98h] [rbp-98h]
  __int64 v14; // [rsp+A0h] [rbp-90h]
  char *src; // [rsp+A8h] [rbp-88h]
  __int64 v16; // [rsp+B0h] [rbp-80h]
  __int64 *v17; // [rsp+B8h] [rbp-78h]
  __int64 v18; // [rsp+C0h] [rbp-70h]
  __int64 v19; // [rsp+C8h] [rbp-68h]
  int v20; // [rsp+D4h] [rbp-5Ch]
  int n; // [rsp+D8h] [rbp-58h]
  int m; // [rsp+DCh] [rbp-54h]
  int k; // [rsp+E0h] [rbp-50h]
  int j; // [rsp+E4h] [rbp-4Ch]
  int v25; // [rsp+E8h] [rbp-48h]
  int v26; // [rsp+ECh] [rbp-44h]
  int i; // [rsp+F0h] [rbp-40h]
  char v28; // [rsp+F7h] [rbp-39h]
  int v29; // [rsp+F8h] [rbp-38h]
  int v30; // [rsp+FCh] [rbp-34h]

  s = a1;
  v8 = a2;
  v20 = strlen(a1);
  v19 = v20 - 1LL;
  v7[8] = v20;
  v7[9] = 0LL;
  v2 = v20;
  v18 = v19;
  v7[14] = v20;
  v7[15] = 0LL;
  v7[12] = v20;
  v7[13] = 0LL;
  v7[10] = v20;
  v7[11] = 0LL;
  v3 = alloca(16 * ((v20 * (__int64)v20 + 15) / 0x10uLL));
  v17 = v7;
  v25 = 0;
  v16 = v19;
  v7[6] = v20;
  v7[7] = 0LL;
  v7[4] = v20;
  v7[5] = 0LL;
  v4 = alloca(16 * ((v20 + 15LL) / 0x10uLL));
  src = (char *)v7;
  v26 = 0;
  for ( i = 0; i < v20; ++i )
  {
```

```
    if ( ((*__ctype_b_loc())[s[i]] & 0x800) != 0 )
    {
      src[v26++] = s[i];
    }
    else if ( (unsigned __int8)is_operator_char((unsigned int)s[i]) )
    {
      v28 = 0;
      if ( s[i] == 45 && !v26 )
      {
        *src = s[i];
        ++v26;
        v28 = 1;
      }
      if ( v28 != 1 )
      {
        if ( v26 > 0 )
        {
          src[v26] = 0;
          strcpy((char *)v17 + v2 * v25, src);
          v26 = 0;
          *src = 0;
          ++v25;
        }
        v11[0] = s[i];
        v11[1] = 0;
        strcpy((char *)v17 + v2 * v25++, v11);
      }
    }
    else
    {
      if ( v26 > 0 )
      {
        src[v26] = 0;
        strcpy((char *)v17 + v2 * v25, src);
        v26 = 0;
        *src = 0;
        ++v25;
      }
      v10[0] = s[i];
      v10[1] = 0;
      strcpy((char *)v17 + v2 * v25++, v10);
    }
  }
  if ( v26 > 0 )
  {
    src[v26] = 0;
    strcpy((char *)v17 + v2 * v25++, src);
  }
  v14 = v25 - 1LL;
  v7[2] = v25;
  v7[3] = 0LL;
  v7[0] = v25;
  v7[1] = 0LL;
  v5 = alloca(16 * ((8LL * v25 + 15) / 0x10uLL));
  v13 = v7;
  v29 = 0;
  v30 = -1;
  for ( j = 0; j < v25; ++j )
```

```
  {
    if ( (unsigned __int8)is_number((const char *)v17 + v2 * j) )
      goto LABEL_29;
    if ( !strcmp((const char *)v17 + v2 * j, L"d=") )
    {
      for ( k = v29 - 1; k >= 0; --k )
        processCommand((const char *)v13[k], v8);
      v29 = 0;
      processCommand((const char *)v17 + v2 * j, v8);
    }
    else
    {
      if ( !strcmp((const char *)v17 + v2 * j, L"rd=") || !strcmp((const char *)v17
+ v2 * j, L"=") )
      {
LABEL_29:
        processCommand((const char *)v17 + v2 * j, v8);
        continue;
      }
      if ( (unsigned __int8)is_operator((const char *)v17 + v2 * j) )
      {
        v12 = op_precidence((char *)v17 + v2 * j);
        if ( v12 >= v30 )
        {
          v13[v29++] = (__int64)v17 + v2 * j;
          v30 = v12;
        }
        else
        {
          for ( m = v29 - 1; m >= 0; --m )
            processCommand((const char *)v13[m], v8);
          v29 = 0;
          v30 = v12;
          *v13 = (__int64)v17 + v2 * j;
          ++v29;
        }
      }
      else
      {
        printf("Unrecognised operator or operand \"%s\".\n", (const char *)v17 + v2
* j);
      }
    }
  }
  result = (unsigned int)(v29 - 1);
  for ( n = v29 - 1; n >= 0; --n )
    result = processCommand((const char *)v13[n], v8);
  return result;
}
// 400A9F: using guessed type __int64 __fastcall is_operator_char(_QWORD);
// 400C4D: using guessed type __int64 __fastcall op_precidence(_QWORD);

//----- (00000000004014CF) ----------------------------------------------------
void shunt()
{
  ;
}
```

```
//----- (00000000004014DD) ----------------------------------------------------
__int64 __fastcall processCommand(const char *a1, __int64 a2)
{
  if ( !(unsigned __int8)is_operator(a1) || !strcmp(a1, L"rd=") )
    return processNumber(a1, a2);
  else
    return processOperator(a1, a2);
}
// 401552: using guessed type __int64 __fastcall processNumber(_QWORD, _QWORD);
// 401678: using guessed type __int64 __fastcall processOperator(_QWORD, _QWORD);

//----- (0000000000401552) ----------------------------------------------------
int __fastcall processNumber(const char *a1, __int64 a2)
{
  int result; // eax
  unsigned int v3; // eax
  double v4; // xmm0_8
  double v5; // xmm0_8

  if ( stack_count == 23 )
    return puts("Stack overflow.");
  if ( !strcmp(a1, L"rd=") )
  {
    if ( rand_count == 22 )
      srand(0);
    rand();
    push(a2);
    ++stack_count;
    return ++rand_count;
  }
  else if ( (unsigned __int8)isOctal(a1) )
  {
    result = isValidOctal(a1);
    if ( (_BYTE)result )
    {
      v3 = atoi(a1);
      v4 = convertOctalToDecimal(v3);
      check_limit(v4);
      push(a2);
      return ++stack_count;
    }
  }
  else
  {
    v5 = atof(a1);
    check_limit(v5);
    push(a2);
    return ++stack_count;
  }
  return result;
}
// 4009BD: using guessed type __int64 __fastcall isOctal(_QWORD);
// 400A14: using guessed type __int64 __fastcall isValidOctal(_QWORD);
// 400B6A: using guessed type __int64 __fastcall check_limit(double);
// 400BAC: using guessed type double __fastcall convertOctalToDecimal(_QWORD);
// 401AF9: using guessed type __int64 __fastcall push(_QWORD);
// 6030F4: using guessed type int rand_count;
// 6030F8: using guessed type int stack_count;
```

```
//----- (0000000000401678) ----------------------------------------------------
int __fastcall processOperator(const char *a1, __int64 a2)
{
  int result; // eax
  double v3; // xmm0_8
  double v4; // xmm0_8
  double v5; // xmm0_8
  double v6; // xmm0_8
  double v7; // xmm0_8
  double v8; // xmm0_8
  double v9; // xmm0_8
  double x; // [rsp+18h] [rbp-18h]
  double y; // [rsp+20h] [rbp-10h]
  double v12; // [rsp+28h] [rbp-8h]

  if ( !strcmp(a1, L"d=") )
    return print_stack(a2);
  if ( !strcmp(a1, L"=") )
  {
    v12 = peek(a2);
    if ( v12 == 9.223372036854776e18 )
      return puts("Stack empty.");
    else
      return printf("%d\n", (unsigned int)(int)v12);
  }
  if ( stack_count <= 1 )
    return puts("Stack underflow.");
  y = pop(a2);
  x = pop(a2);
  stack_count -= 2;
  if ( !strcmp(a1, &asc_401DA8[2]) )
  {
    v3 = check_limit(x + y);
    push(a2, v3);
    return ++stack_count;
  }
  if ( !strcmp(a1, asc_401DA8) )
  {
    v4 = check_limit(x - y);
    push(a2, v4);
    return ++stack_count;
  }
  if ( !strcmp(a1, &asc_401DA8[4]) )
  {
    v5 = check_limit(x * y);
    push(a2, v5);
    return ++stack_count;
  }
  if ( !strcmp(a1, &asc_401DA8[6]) )
  {
    if ( y != 0.0 )
    {
      v6 = check_limit(x / y);
      push(a2, v6);
      return ++stack_count;
    }
    goto LABEL_17;
```

```
  }
  if ( !strcmp(a1, &asc_401DA8[8]) )
  {
    if ( x != 0.0 )
    {
      v7 = check_limit((double)((int)x % (int)y));
      push(a2, v7);
      return ++stack_count;
    }
LABEL_17:
    puts("Divide by 0.");
    push(a2, x);
    push(a2, y);
    result = stack_count + 2;
    stack_count += 2;
    return result;
  }
  result = strcmp(a1, &asc_401DA8[10]);
  if ( !result )
  {
    if ( y >= 0.0 )
    {
      v8 = pow(x, y);
      v9 = check_limit(v8);
      push(a2, v9);
      return ++stack_count;
    }
    else
    {
      puts("Negative power.");
      push(a2, x);
      push(a2, y);
      result = stack_count + 2;
      stack_count += 2;
    }
  }
  return result;
}
// 401B9B: using guessed type double __fastcall pop(_QWORD);
// 401C50: using guessed type double __fastcall peek(_QWORD);
// 401CB7: using guessed type __int64 __fastcall print_stack(_QWORD);
// 6030F8: using guessed type int stack_count;

//----- (0000000000401A1F) ----------------------------------------------------
__int64 run_srpn()
{
  __int64 result; // rax
  char s1[112]; // [rsp+0h] [rbp-90h] BYREF
  __int64 v2[3]; // [rsp+70h] [rbp-20h] BYREF
  char v3; // [rsp+8Bh] [rbp-5h]
  int i; // [rsp+8Ch] [rbp-4h]

  v2[0] = 0x43E0000000000000LL;
  v2[1] = 0LL;
  i = 0;
  v3 = 0;
  result = __isoc99_scanf("%s", s1);
  for ( i = result; i >= 0; i = result )
```

```
  {
    if ( !strcmp(s1, "#") )
    {
      v3 = v3 == 0;
      v3 &= 1u;
    }
    else if ( v3 != 1 )
    {
      processToken(s1, v2);
    }
    result = __isoc99_scanf("%s", s1);
  }
  return result;
}
// 400800: using guessed type __int64 __isoc99_scanf(const char *, ...);
// 400C9C: using guessed type __int64 __fastcall processToken(_QWORD, _QWORD);

//----- (0000000000401ADE) ----------------------------------------------------
int __cdecl main(int argc, const char **argv, const char **envp)
{
  run_srpn(argc, argv, envp);
  return 0;
}
// 401A1F: using guessed type __int64 __fastcall run_srpn(_QWORD, _QWORD, _QWORD);

//----- (0000000000401AF9) ----------------------------------------------------
__int64 __fastcall push(__int64 a1, double a2)
{
  __int64 result; // rax
  double *v3; // [rsp+18h] [rbp-8h]

  if ( *(double *)a1 == 9.2233720368547776e18 )
  {
    *(double *)a1 = a2;
    result = a1;
    *(_QWORD *)(a1 + 8) = 0LL;
  }
  else if ( *(_QWORD *)(a1 + 8) )
  {
    return push(*(_QWORD *)(a1 + 8));
  }
  else
  {
    v3 = (double *)malloc(0x10uLL);
    *v3 = a2;
    v3[1] = 0.0;
    result = a1;
    *(_QWORD *)(a1 + 8) = v3;
  }
  return result;
}

//----- (0000000000401B9B) ----------------------------------------------------
__int64 __fastcall pop(double *a1)
{
  return popfromstack(a1, 0LL);
}
```

```
//----- (0000000000401BC8) ----------------------------------------------------
double __fastcall popfromstack(double *a1, __int64 a2)
{
  double v4; // [rsp+28h] [rbp-8h]

  if ( *((_QWORD *)a1 + 1) )
    return popfromstack(*((_QWORD *)a1 + 1), a1);
  v4 = *a1;
  if ( a2 )
  {
    *(_QWORD *)(a2 + 8) = 0LL;
    free(a1);
  }
  else
  {
    *a1 = 9.223372036854776e18;
  }
  return v4;
}

//----- (0000000000401C50) ----------------------------------------------------
double __fastcall peek(_QWORD *a1)
{
  if ( *(double *)a1 == 9.223372036854776e18 )
    return 9.223372036854776e18;
  if ( a1[1] )
    return peek(a1[1]);
  return *(double *)a1;
}

//----- (0000000000401CB7) ----------------------------------------------------
int __fastcall print_stack(__int64 a1)
{
  if ( !*(_QWORD *)(a1 + 8) )
    return printf("%d\n", (unsigned int)(int)*(double *)a1);
  printf("%d\n", (unsigned int)(int)*(double *)a1);
  return print_stack(*(_QWORD *)(a1 + 8));
}

//----- (0000000000401D20) ----------------------------------------------------
void __fastcall _libc_csu_init(unsigned int a1, __int64 a2, __int64 a3)
{
  signed __int64 v4; // rbp
  __int64 i; // rbx

  v4 = &_do_global_dtors_aux_fini_array_entry - &_frame_dummy_init_array_entry;
  init_proc();
  if ( v4 )
  {
    for ( i = 0LL; i != v4; ++i )
      ((void (__fastcall *)(_QWORD, __int64,
__int64))*(&_frame_dummy_init_array_entry + i))(a1, a2, a3);
  }
}
// 400900: using guessed type __int64 __fastcall frame_dummy(_QWORD, _QWORD,
_QWORD);
// 602E00: using guessed type __int64 (__fastcall *_frame_dummy_init_array_entry)();
```

```
// 602E08: using guessed type __int64 (__fastcall
*_do_global_dtors_aux_fini_array_entry)();

//----- (0000000000401D94) ----------------------------------------------------
void term_proc()
{
  ;
}

// nfuncs=61 queued=29 decompiled=29 lumina nreq=0 worse=0 better=0
// ALL OK, 29 function(s) have been successfully decompiled
```