## 7.20.2 Pseudo-random sequence generation functions

### 7.20.2.1 The `rand` function

**Synopsis**

1
```
#include <stdlib.h>
int rand(void);
```

**Description**

2 The **rand** function computes a sequence of pseudo-random integers in the range 0 to **RAND_MAX**.

3 The implementation shall behave as if no library function calls the **rand** function.

**Returns**

4 The **rand** function returns a pseudo-random integer.

**Environmental limits**

5 The value of the **RAND_MAX** macro shall be at least 32767.

### 7.20.2.2 The `srand` function

**Synopsis**

1
```
#include <stdlib.h>
void srand(unsigned int seed);
```

**Description**

2 The **srand** function uses the argument as a seed for a new sequence of pseudo-random numbers to be returned by subsequent calls to **rand**. If **srand** is then called with the same seed value, the sequence of pseudo-random numbers shall be repeated. If **rand** is called before any calls to **srand** have been made, the same sequence shall be generated as when **srand** is first called with a seed value of 1.

3 The implementation shall behave as if no library function calls the **srand** function.

**Returns**

4 The **srand** function returns no value.

5 EXAMPLE The following functions define a portable implementation of **rand** and **srand**.

```
static unsigned long int next = 1;

int rand(void)   // RAND_MAX assumed to be 32767
{
      next = next * 1103515245 + 12345;
      return (unsigned int)(next/65536) % 32768;
}
```

```
void srand(unsigned int seed)
{
      next = seed;
}
```

### 7.20.3 Memory management functions

1 The order and contiguity of storage allocated by successive calls to the **calloc**, **malloc**, and **realloc** functions is unspecified. The pointer returned if the allocation succeeds is suitably aligned so that it may be assigned to a pointer to any type of object and then used to access such an object or an array of such objects in the space allocated (until the space is explicitly deallocated). The lifetime of an allocated object extends from the allocation until the deallocation. Each such allocation shall yield a pointer to an object disjoint from any other object. The pointer returned points to the start (lowest byte address) of the allocated space. If the space cannot be allocated, a null pointer is returned. If the size of the space requested is zero, the behavior is implementation-defined: either a null pointer is returned, or the behavior is as if the size were some nonzero value, except that the returned pointer shall not be used to access an object.

#### 7.20.3.1 The `calloc` function

**Synopsis**

1
```
#include <stdlib.h>
void *calloc(size_t nmemb, size_t size);
```

**Description**

2 The **calloc** function allocates space for an array of **nmemb** objects, each of whose size is **size**. The space is initialized to all bits zero.[261)]

**Returns**

3 The **calloc** function returns either a null pointer or a pointer to the allocated space.

#### 7.20.3.2 The `free` function

**Synopsis**

1
```
#include <stdlib.h>
void free(void *ptr);
```

**Description**

2 The **free** function causes the space pointed to by **ptr** to be deallocated, that is, made available for further allocation. If **ptr** is a null pointer, no action occurs. Otherwise, if the argument does not match a pointer earlier returned by the **calloc**, **malloc**, or

261) Note that this need not be the same as the representation of floating-point zero or a null pointer constant.