Imperative programming

10. Standard library

Zoltán Porkoláb

The C standard library

- Input/output library <stdio.h>
- Strings library <string.h>
- Numerics library <math.h> <stdlib.h> <complex.h>
- Algorithms library <stdlib.h>
- Dynamic memory management <stdlib.h>
- Date and time library <time.h>
- Program utilities <stddef.f> <stdbool.h> <stdalign.h> <stdnoreturn.h>
- Type support library <stddef.h> <stdbool.h> <stdalign.h> <stdnoreturn.h>
- Diagnostics (error handling) library <errno.h> <assert.h> <stdlib.h>
- Concurrency support library <threads.h> <stdatomic.h>
- Variadic functions <stdarg.h>
- Localization library <locale.h>

File handling

```
#include <errno.h>   /* extern int errno    */
#include <string.h>  /* strerror            */
#include <stdio.h>   /* io functions, stdin,stdout,stderr  */

void f()
{
  FILE *ifp = fopen("inputfname", "r");  /* open the file for read  */
  if ( NULL == ifp )
  {
    fprintf(stderr, "can't open file %s for read\n", "inputfname");
    fprintf(stderr, "reason: %s\n", strerror(errno)); /* perror(NULL) */
    return;
  }
  FILE *ofp;
  if ( NULL == (ofp = fopen("outputfname","w")) ) /* open the file for write */
  {
    perror("can't open outputfname");  /* strerror(errno) written to stderr */
    fclose(ifp);   /* close the open file */
    return;
  }
  int cnt = 0;
  int ch;
  while( EOF != (ch = fgetc(ifp)) )
  {
    fputc(ch, ofp);  /* print ch to ofp */
    ++cnt;
  }
  fprintf(stderr, "copied %d characters\n", cnt);
  fclose(ifp);  /* close the open files */
  fclose(ofp);
}
```

File descriptor

```c
#include <errno.h>   /* extern int errno    */
#include <string.h>  /* strerror            */
#include <stdio.h>   /* io functions, stdin,stdout,stderr  */

void f()
{
  FILE *ifp = fopen("inputfname", "r");  /* open the file for read  */
  if ( NULL == ifp )
  {
    fprintf(stderr, "can't open file %s for read\n", "inputfname");
    fprintf(stderr, "reason: %s\n", strerror(errno)); /* perror(NULL) */
    return;
  }
  FILE *ofp;
  if ( NULL == (ofp = fopen("outputfname","w")) ) /* open the file for write */
  {
    perror("can't open outputfname");  /* strerror(errno) written to stderr */
    fclose(ifp);   /* close the open file */
    return;
  }
  int cnt = 0;
  int ch;
  while( EOF != (ch = fgetc(ifp)) )
  {
    fputc(ch, ofp);  /* print ch to ofp */
    ++cnt;
  }
  fprintf(stderr, "copied %d characters\n", cnt);
  fclose(ifp);  /* close the open files */
  fclose(ofp);
}
```

File descriptor

```
#include <errno.h>   /* extern int errno    */
#include <string.h>  /* strerror            */
#include <stdio.h>   /* io functions, stdin,stdout,stderr  */

void f(char *ifname, char *ofname)   /* copy input file to output file */
{
  FILE *ifp = fopen(ifname, "r");  /* open the file for read  */
```

- stdin -- standard input, bufferelt
- stdout -- standard output, bufferelt
- stderr -- standard error, nem bufferelt

File open mode

```c
#include <errno.h>   /* extern int errno    */
#include <string.h>  /* strerror            */
#include <stdio.h>   /* io functions, stdin,stdout,stderr  */

void f(char *ifname, char *ofname)   /* copy input file to output file */
{
  FILE *ifp = fopen(ifname, "r" );  /* open the file for read  */
```

		if file exists	if file does not exist
• "r"	-- open for read,	from start,	error
• "r+"	-- open for read/write,	from start,	error
• "w"	-- open for write,	destroy content,	create empty new file
• "w+"	-- open for read/write,	destroy content,	create empty new file
• "a"	-- open for append,	write to end,	create empty new file
• "a+"	-- open for read/append,	write to end,	create empty new file

File descriptor

```c
#include <errno.h>   /* extern int errno    */
#include <string.h>  /* strerror            */
#include <stdio.h>   /* io functions, stdin,stdout,stderr  */

void f(char *ifname, char *ofname)   /* copy input file to output file */
{
  FILE *ifp = fopen(ifname, "r");  /* open the file for read  */
  if ( NULL == ifp )
  {
    fprintf(stderr, "can't open file %s for read\n", ifname);
    fprintf(stderr, "reason: %s\n", strerror(errno)); /* perror(NULL) */
    return;
  }
  FILE *ofp;
  if ( NULL == (ofp = fopen(ofname,"w")) ) /* open the file for write */
  {
    perror(ofname); /* ofname+": "+strerror(errno) written to stderr */
    fclose(ifp);    /* close the open file */
    return;
  }
  int cnt = 0;
  int ch;
  while( EOF != (ch = fgetc(ifp)) )
  {
    fputc(ch, ofp);  /* print ch to ofp */
    ++cnt;
  }
  fprintf(stderr, "copied %d characters\n", cnt);
  fclose(ifp);  /* close the open files */
  fclose(ofp);
}
```

Error handling

```
#include <errno.h>   /* extern int errno    */
#include <string.h>  /* strerror            */
#include <stdio.h>   /* io functions, stdin,stdout,stderr  */

void f(char *ifname, char *ofname)   /* copy input file to output file */
{
  FILE *ifp = fopen(ifname, "r");  /* open the file for read  */
  if ( NULL == ifp )
  {
    fprintf(stderr, "can't open file %s for read\n", ifname);
    fprintf(stderr, "reason: %s\n", strerror( errno )); /* perror(NULL) */
    return;
  }
  FILE *ofp;
  if ( NULL == (ofp = fopen(ofname,"w")) ) /* open the file for write */
  {
    perror(ofname); /* ofname+": "+strerror(errno) written to stderr */
    fclose(ifp);    /* close the open file */
    return;
  }
  int cnt = 0;
  int ch;
  while( EOF != (ch = fgetc(ifp)) )
  {
    fputc(ch, ofp);  /* print ch to ofp */
    ++cnt;
  }
  fprintf(stderr, "copied %d characters\n", cnt);
  fclose(ifp);  /* close the open files */
  fclose(ofp);
}
```

Error handling

```
#include <errno.h>   /* extern int errno    */
#include <string.h>  /* strerror            */
#include <stdio.h>   /* io functions, stdin,stdout,stderr  */

void f(char *ifname, char *ofname)   /* copy input file to output file */
{
  FILE *ifp = fopen(ifname, "r");  /* open the file for read  */
  if ( NULL == ifp )
  {
    fprintf(stderr, "can't open file %s for read\n", ifname);
    fprintf(stderr, "reason: %s\n", strerror( errno )); /* perror(NULL) */
    return;
  }
  FILE *ofp;
  if ( NULL == (ofp = fopen(ofname,"w")) ) /* open the file for write */
  {
    perror(ofname); /* ofname+": "+strerror(errno) written to stderr */
    fclose(ifp);    /* close the open file */
    return;
  }
  int cnt = 0;
  int ch;
  while( EOF != (ch = fgetc(ifp)) )
  {
    fputc(ch, ofp);  /* print ch to ofp */
    ++cnt;
  }
  fprintf(stderr, "copied %d characters\n", cnt);
  fclose(ifp);  /* close the open files */
  fclose(ofp);
}
```

End Of File

```
#include <errno.h>   /* extern int errno    */
#include <string.h>  /* strerror            */
#include <stdio.h>   /* io functions, stdin,stdout,stderr  */

void f(char *ifname, char *ofname)   /* copy input file to output file */
{
  FILE *ifp = fopen(ifname, "r");  /* open the file for read  */


  int cnt = 0;
  int ch;

  while( EOF != (ch = fgetc(ifp)) )
  {
    fputc(ch, ofp);  /* print ch to ofp */
    ++cnt;

  }


}
```

End Of File

```
#include <errno.h>   /* extern int errno    */
#include <string.h>  /* strerror            */
#include <stdio.h>   /* io functions, stdin,stdout,stderr  */

void f(char *ifname, char *ofname)   /* copy input file to output file */
{
  FILE *ifp = fopen(ifname, "r");  /* open the file for read  */



  int cnt = 0;
  int ch = fgetc(ifp);

  while( !feof(ifp) )
  {
    fputc(ch, ofp);  /* print ch to output file */
    ++cnt;
    ch = fgetc(ifp);
  }



}
```

Output

```
#include <errno.h>   /* extern int errno    */
#include <string.h>  /* strerror            */
#include <stdio.h>   /* io functions, stdin,stdout,stderr  */

void f(char *ifname, char *ofname)   /* copy input file to output file */
{
  FILE *ifp = fopen(ifname, "r");  /* open the file for read  */


  int cnt = 0;
  int ch;

  while( EOF != (ch = fgetc(ifp)) )
  {
    fputc(ch, ofp);  /* print to output file */
    ++cnt;

  }
  fprintf(stderr, "copied %d characters\n", cnt);
  fclose(ifp);
  fclose(ofp);
}
```

Buffered input and output

```c
#include <errno.h>   /* extern int errno    */
#include <string.h>  /* strerror            */
#include <stdio.h>   /* io functions, stdin,stdout,stderr  */

void f(char *ifname, char *ofname)   /* copy input file to output file */
{
  FILE *ifp = fopen(ifname, "r");  /* open the file for read  */



  int cnt = 0;
  int ch;

  while( EOF != (ch = fgetc(ifp)) )
  {
    fputc(ch, ofp);  /* print ch to output buffer */
    ++cnt;

  }
  fprintf(stderr, "copied %d characters\n", cnt);
  fclose(ifp);
  fclose(ofp);    /* flush output buffer before close file */
}
```

Buffered input and output

```
#include <errno.h>   /* extern int errno    */
#include <string.h>  /* strerror            */
#include <stdio.h>   /* io functions, stdin,stdout,stderr  */

void f(char *ifname, char *ofname)   /* copy input file to output file */
{
  FILE *ifp = fopen(ifname, "r");  /* open the file for read  */



  int cnt = 0;
  int ch;

  while( EOF != (ch = fgetc(ifp)) )
  {
    fputc(ch, ofp);  /* print ch to output buffer */
    ++cnt;
    fflush(ofp);     /* flush output buffer now   */
  }
  fprintf(stderr, "copied %d characters\n", cnt);
  fclose(ifp);
  fclose(ofp);    /* flush output buffer before close file */
}
```

Standard error

```
#include <errno.h>   /* extern int errno    */
#include <string.h>  /* strerror            */
#include <stdio.h>   /* io functions, stdin,stdout,stderr  */

void f(char *ifname, char *ofname)   /* copy input file to output file */
{
  FILE *ifp = fopen(ifname, "r");  /* open the file for read  */



  int cnt = 0;
  int ch;

  while( EOF != (ch = fgetc(ifp)) )
  {
    fputc(ch, ofp);  /* print ch to output buffer */
    ++cnt;
    fflush(ofp);     /* flush output buffer now   */
  }
  fprintf(stderr, "copied %d characters\n", cnt); /* stderr does not use buffer */
  fclose(ifp);
  fclose(ofp);
}
```

File handling

```
#include <errno.h>   /* extern int errno    */
#include <string.h>  /* strerror            */
#include <stdio.h>   /* io functions, stdin,stdout,stderr  */

void f()
{
  FILE *ifp = fopen("inputfname", "r");  /* open the file for read  */
  if ( NULL == ifp )
  {
    fprintf(stderr, "can't open file %s for read\n", "inputfname");
    fprintf(stderr, "reason: %s\n", strerror(errno)); /* perror(NULL) */
    return;
  }
  FILE *ofp;
  if ( NULL == (ofp = fopen("outputfname","w")) ) /* open the file for write */
  {
    fclose(ifp);   /* close the open file */
    return;
  }
  int cnt = 0;
  int ch;

  while( EOF != (ch = fgetc(ifp)) )
  {
    fputc(ch, ofp);  /* print ch to ofp */
    ++cnt;
  }
  fprintf(stderr, "copied %d characters\n", cnt);
  fclose(ifp);  /* close the open files */
  fclose(ofp);
}
```

Line-based input

```
#include <errno.h>   /* extern int errno    */
#include <string.h>  /* strerror            */
#include <stdio.h>   /* io functions, stdin,stdout,stderr  */

void f()
{
  FILE *ifp = fopen("inputfname", "r");  /* open the file for read  */
  if ( NULL == ifp )
  {
    fprintf(stderr, "can't open file %s for read\n", "inputfname");
    fprintf(stderr, "reason: %s\n", strerror(errno)); /* perror(NULL) */
    return;
  }
  FILE *ofp;
  if ( NULL == (ofp = fopen("outputfname","w")) ) /* open the file for write */
  {
    fclose(ifp);   /* close the open file */
    return;
  }
  int cnt = 0;
  char buffer[1024];

  while( NULL != fgets(buffer, sizeof(buffer), ifp) )  /* place '\0' at end */
  {
    fprintf(ofp, "%s", buffer);  /* print ch to ofp */
    cnt += strlen(buffer);
  }
  fprintf(stderr, "copied %d characters\n", cnt);
  fclose(ifp);  /* close the open files */
  fclose(ofp);
}
```

Direct input/output

```c
#include <errno.h>   /* extern int errno    */
#include <string.h>  /* strerror            */
#include <stdio.h>   /* io functions, stdin,stdout,stderr  */

void f()
{
  FILE *ifp = fopen("inputfname", "r");  /* open the file for read  */
  if ( NULL == ifp )
  {
    fprintf(stderr, "can't open file %s for read\n", "inputfname");
    return;
  }
  FILE *ofp;
  if ( NULL == (ofp = fopen("outputfname","w")) ) /* open the file for write */
  {
    fclose(ifp);   /* close the open file */
    return;
  }
  int cnt = 0;
  char buffer[1024];
  int ret;

  while( (ret = fread(buffer, sizeof(char), 1024, ifp)) > 0 ) /* read max 1024 items */
  {
    fwrite(buffer, sizeof(char), ret, ofp);  /* actually read ret items */
    cnt += ret;
  }
  fprintf(stderr, "copied %d characters\n", cnt);
  fclose(ifp);  /* close the open files */
  fclose(ofp);
}
```

Direct input/output

```
#include <errno.h>   /* extern int errno    */
#include <string.h>  /* strerror            */
#include <stdio.h>   /* io functions, stdin,stdout,stderr  */

void f()
{
  FILE *ifp = fopen("inputfname", "r");  /* open the file for read  */
  if ( NULL == ifp )
  {
    fprintf(stderr, "can't open file %s for read\n", "inputfname");
    return;
  }
  FILE *ofp;
  if ( NULL == (ofp = fopen("outputfname","w")) ) /* open the file for write */
  {
    fclose(ifp);   /* close the open file */
    return;
  }
  int cnt = 0;
  char buffer[1024];
  int ret;

  while( (ret = fread(buffer, sizeof(char), 1024, ifp)) > 0 ) /* no '\0' placed */
  {
    fwrite(buffer, sizeof(char), ret, ofp);  /* actually read ret items */
    cnt += ret;
  }
  fprintf(stderr, "copied %d characters\n", cnt);
  fclose(ifp);  /* close the open files */
  fclose(ofp);
}
```

Direct input/output

```
#include <string.h>  /* strcmp       */
#include <stdio.h>   /* io functions */

struct employee
{
  char name[40];  // terminated by '\0'
  long salary;
};

void increase_salary(FILE *fp, char *name, long incr)
{
  struct employee emp;

  while( 1 == fread(&emp, sizeof(struct employee), 1, fp) )  // read next employee
  {
    if ( 0 == strcmp(name, emp.name) )  // found employee
    {
      long pos = ftell(fp);  // current position in file (is the next employee)
      emp.salary += incr;
      fseek(fp, pos-sizeof(struct employee), SEEK_SET);  // go back to current
      fwrite(&emp, sizeof(char), ret, fp);  /* (re-)write employee
      break;
    }
  }
}
```

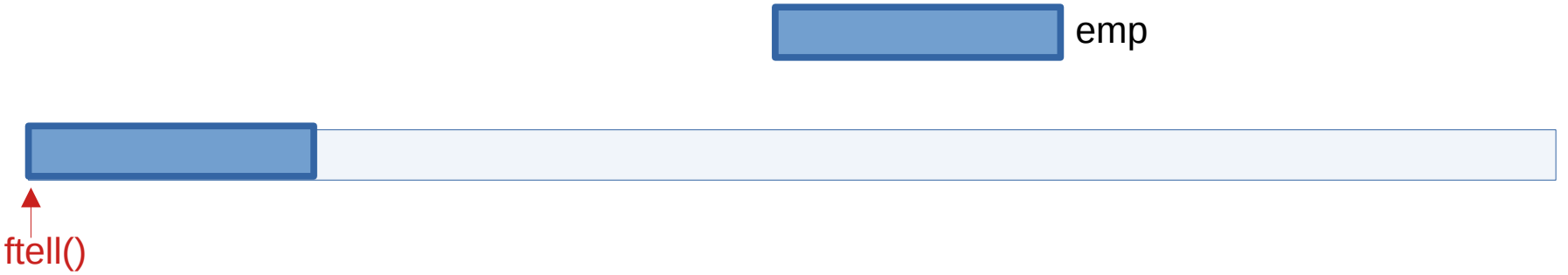
Direct input/output

```
void increase_salary(FILE *fp, char *name, long incr)
{
  struct employee emp;

  while( 1 == fread(&emp, sizeof(struct employee), 1, fp) )  // read next employee
  {
    if ( 0 == strcmp(name, emp.name) )  // found employee
    {
      long pos = ftell(fp);  // current position in file (is the next employee)
      emp.salary += incr;
      fseek(fp, pos-sizeof(struct employee), SEEK_SET);  // go back to current
      fwrite(&emp, sizeof(char), ret, fp);  /* (re-)write employee
      break;
    }
  }
}
```

Direct input/output


```
void increase_salary(FILE *fp, char *name, long incr)
{
  struct employee emp;

  while( 1 == fread(&emp, sizeof(struct employee), 1, fp) )  // read next employee
  {
    if ( 0 == strcmp(name, emp.name) )  // found employee
    {
      long pos = ftell(fp);  // current position in file (is the next employee)
      emp.salary += incr;
      fseek(fp, pos-sizeof(struct employee), SEEK_SET);  // go back to current
      fwrite(&emp, sizeof(char), ret, fp);  /* (re-)write employee
      break;
    }
  }
}
```

Direct input/output

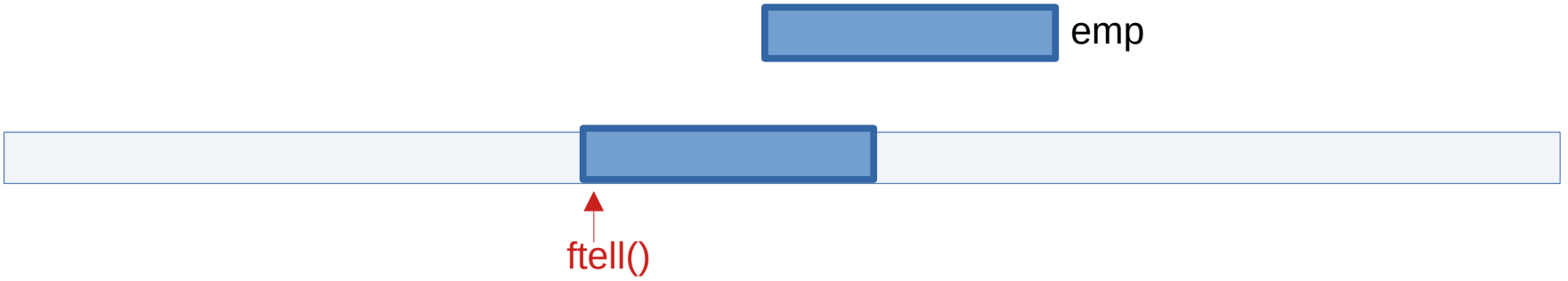



```
void increase_salary(FILE *fp, char *name, long incr)
{
  struct employee emp;

  while( 1 == fread(&emp, sizeof(struct employee), 1, fp) )  // read next employee
  {
    if ( 0 == strcmp(name, emp.name) )  // found employee
    {
      long pos = ftell(fp);  // current position in file (is the next employee)
      emp.salary += incr;
      fseek(fp, pos-sizeof(struct employee), SEEK_SET);  // go back to current
      fwrite(&emp, sizeof(char), ret, fp);  /* (re-)write employee
      break;
    }
  }
}
```

Direct input/output

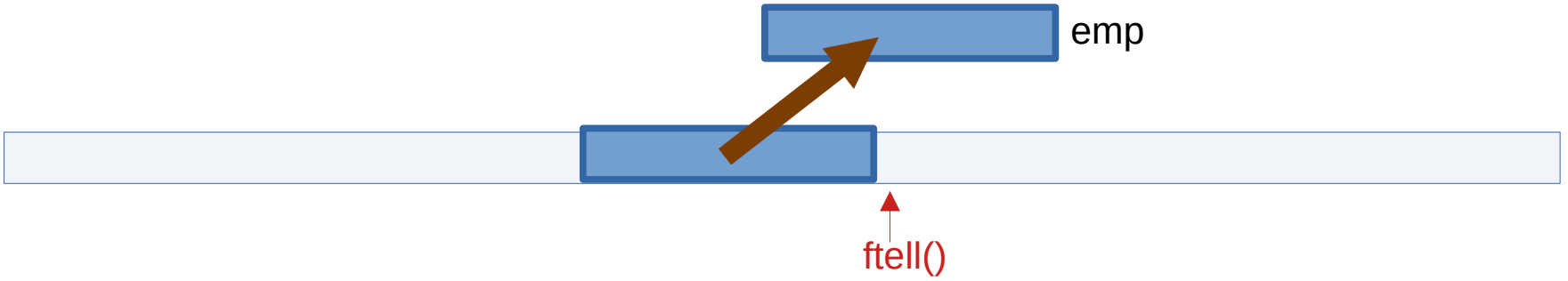

```
void increase_salary(FILE *fp, char *name, long incr)
{
  struct employee emp;

  while( 1 == fread(&emp, sizeof(struct employee), 1, fp) )  // read next employee
  {
    if ( 0 == strcmp(name, emp.name) )  // found employee
    {
      long pos = ftell(fp);  // current position in file (is the next employee)
      emp.salary += incr;
      fseek(fp, pos-sizeof(struct employee), SEEK_SET);  // go back to current
      fwrite(&emp, sizeof(char), ret, fp);  /* (re-)write employee
      break;
    }
  }
}
```

Direct input/output



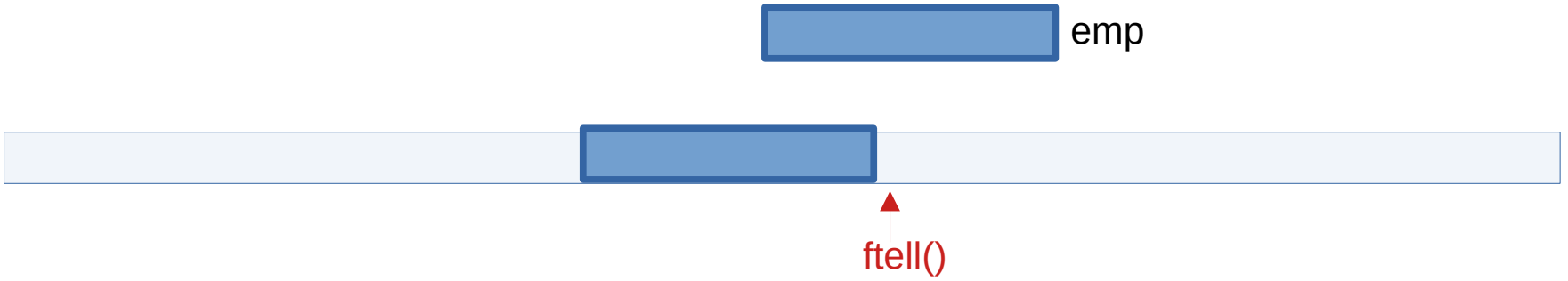

```
void increase_salary(FILE *fp, char *name, long incr)
{
  struct employee emp;

  while( 1 == fread(&emp, sizeof(struct employee), 1, fp) )  // read next employee
  {
    if ( 0 == strcmp(name, emp.name) )  // found employee
    {
      long pos = ftell(fp);  // current position in file (is the next employee)
      emp.salary += incr;
      fseek(fp, pos-sizeof(struct employee), SEEK_SET);  // go back to current
      fwrite(&emp, sizeof(char), ret, fp);  /* (re-)write employee
      break;
    }
  }
}
```

Direct input/output



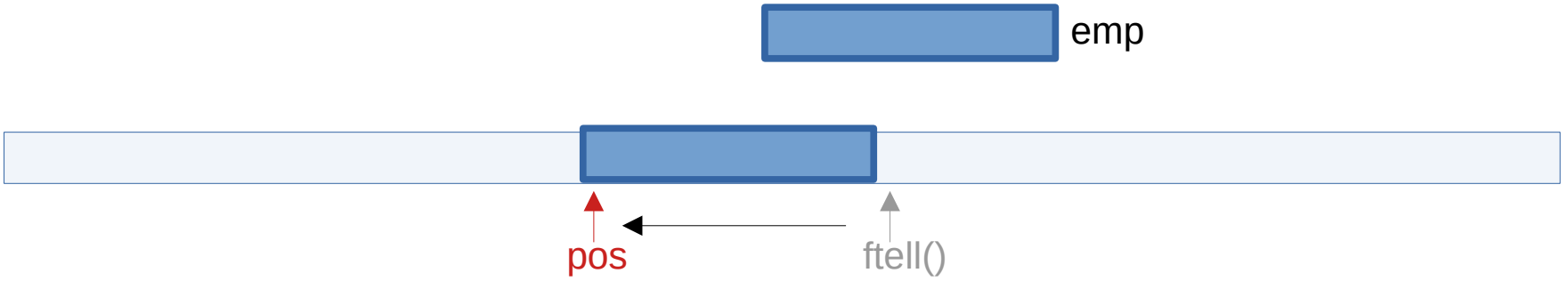

```
void increase_salary(FILE *fp, char *name, long incr)
{
  struct employee emp;

  while( 1 == fread(&emp, sizeof(struct employee), 1, fp) )  // read next employee
  {
    if ( 0 == strcmp(name, emp.name) )  // found employee
    {
      long pos = ftell(fp);  // current position in file (is the next employee)
      emp.salary += incr;
      fseek(fp, pos-sizeof(struct employee), SEEK_SET);  // go back to current
      fwrite(&emp, sizeof(char), ret, fp);  /* (re-)write employee
      break;
    }
  }
}
```

Direct input/output



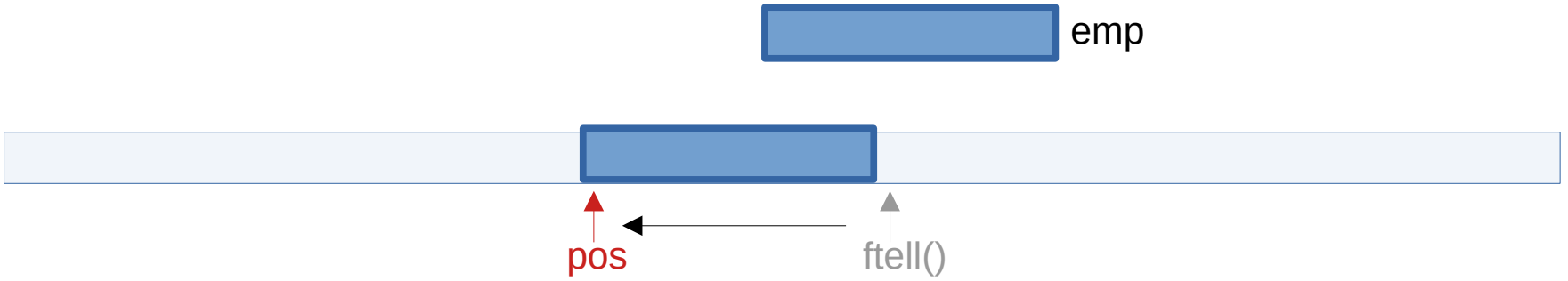

```
void increase_salary(FILE *fp, char *name, long incr)
{
  struct employee emp;

  while( 1 == fread(&emp, sizeof(struct employee), 1, fp) )  // read next employee
  {
    if ( 0 == strcmp(name, emp.name) )  // found employee
    {
      long pos = ftell(fp);  // current position in file (is the next employee)
      emp.salary += incr;
      fseek(fp, pos-sizeof(struct employee), SEEK_SET);  // go back to current
      fwrite(&emp, sizeof(char), ret, fp);  /* (re-)write employee
      break;
    }
  }
}
```

Direct input/output



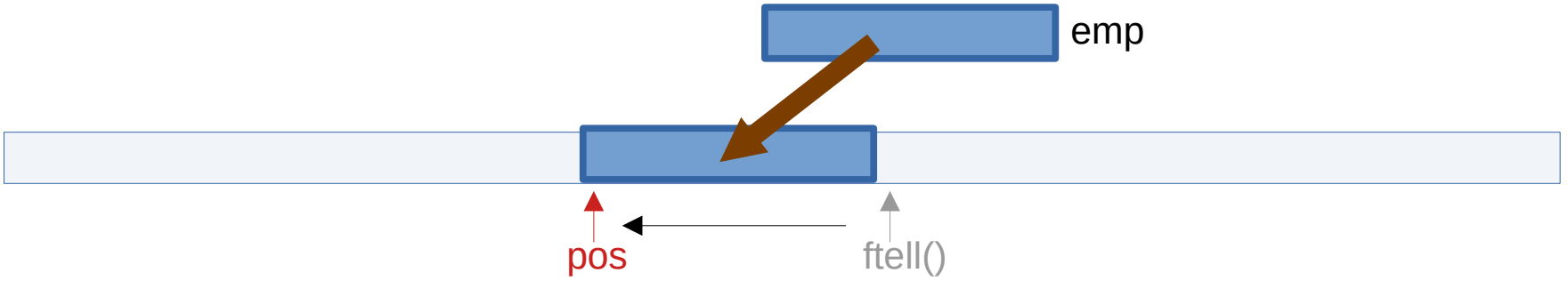

```
void increase_salary(FILE *fp, char *name, long incr)
{
  struct employee emp;

  while( 1 == fread(&emp, sizeof(struct employee), 1, fp) )  // read next employee
  {
    if ( 0 == strcmp(name, emp.name) )  // found employee
    {
      long pos = ftell(fp);  // current position in file (is the next employee)
      emp.salary += incr;
      fseek(fp, pos-sizeof(struct employee), SEEK_SET);  // go back to current
      fwrite(&emp, sizeof(char), ret, fp);  /* (re-)write employee
      break;
    }
  }
}
```

Direct input/output



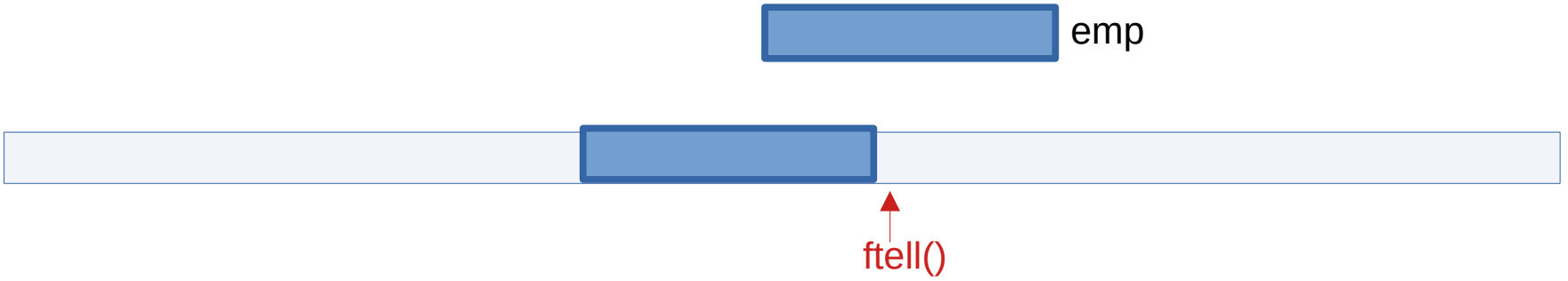

```
void increase_salary(FILE *fp, char *name, long incr)
{
  struct employee emp;

  while( 1 == fread(&emp, sizeof(struct employee), 1, fp) )  // read next employee
  {
    if ( 0 == strcmp(name, emp.name) )  // found employee
    {
      long pos = ftell(fp);  // current position in file (is the next employee)
      emp.salary += incr;
      fseek(fp, -sizeof(struct employee), SEEK_CUR);  // go back to current
      fwrite(&emp, sizeof(char), ret, fp);  /* (re-)write employee
      break;
    }
  }
}
```

Direct input/output



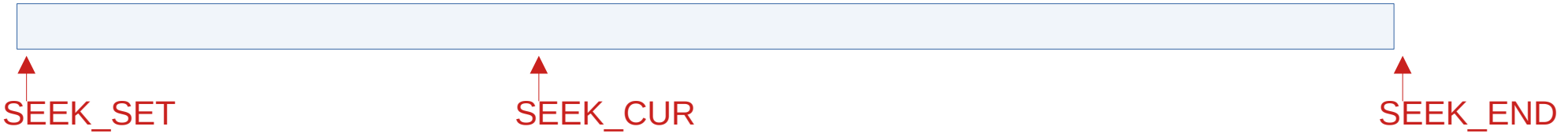

```
void increase_salary(FILE *fp, char *name, long incr)
{
  struct employee emp;

  while( 1 == fread(&emp, sizeof(struct employee), 1, fp) )  // read next employee
  {
    if ( 0 == strcmp(name, emp.name) )  // found employee
    {
      long pos = ftell(fp);  // current position in file (is the next employee)
      emp.salary += incr;
      fseek(fp, -sizeof(struct employee), SEEK_CUR);  // go back to current
      fwrite(&emp, sizeof(char), ret, fp);  /* (re-)write employee
      break;
    }
  }
}
```

Direct input/output




```
void increase_salary(FILE *fp, char *name, long incr)
{
  struct employee emp;

  while( 1 == fread(&emp, sizeof(struct employee), 1, fp) )  // read next employee
  {
    if ( 0 == strcmp(name, emp.name) )  // found employee
    {
      long pos = ftell(fp);  // current position in file (is the next employee)
      emp.salary += incr;
      fseek(fp, pos-sizeof(struct employee), SEEK_SET);  // go back to current
      fwrite(&emp, sizeof(char), ret, fp);  /* (re-)write employee
      break;
    }
  }
}
```

File position




```
rewind(fp);

fseek(fp, 0, SEEK_SET);
```

File operations

```
rewind(fp);

fseek(fp, 0, SEEK_SET);

rename( "oldname", "newname");

remove( "pathname");
```

Buffer operations

```
rewind(fp);

fseek(fp, 0, SEEK_SET);

rename( "oldname", "newname");

remove( "pathname");

char s[20]; int i, double d;
char buffer[80];

sprintf(buffer, "%8d %8.2f, %s", 42, 3.14, "Hello");    // prints to buffer
sscanf( buffer, "%d %f %19s", &i, &d, s);               // reads from buffer
```

String examination

```
size_t strlen(const char *s)  // lenght of a null-terminated string without '\0'
{
  const char *p = s;
  while ( '\0' != *p )
    ++p;
  return p - s;
}

int strcmp(const char *lhs, const char *rhs)  // returns < 0   if lhs < rhs
{                                             // returns == 0  if lhs == rhs
  while ( *lhs == *rhs  &&  '\0' != *lsh )    // returns > 0   if lhs > rhs
  {
    ++lhs;
    ++rhs;
  }
  return *lhs - *rhs;
}

int strncmp(const char *lhs, const char *rhs, int sz)  // compares first n chars
{
  for (int i = 0; i < n; ++i)
  {
    if ( *lhs != *rhs  ||  '\0' == *lsh )
      return *lhs - *rhs;
    ++lhs;
    ++rhs;
  }
  return 0;
}
```

String examination

```
void f()
{
  const char *digits = "0123456789";
  const char *vowels = "aeiou";

  const char *s = "123hello";

  size_t dig_prefix = strspn(s, digits);   // dig_prefix == 3
  size_t no_vowels  = strcspn(s, vowels);  // no_vowels  == 4


  char *pos1 = strstr(s, "3hel");          // pos1 == s+2
  char *pos2 = strstr(s, vowels);          // pos2 == NULL


  char *pos3 = strpbrk(s, vowels);        // pos3 == s+4


  char s2[] = "123hello45world6!";   // should be non-const char buffer

  char *pos4 = strtok(s2, digits);   // pos4 == s2+3,  s2[8]  == '\0', "hello"
  char *pos5 = strtok(NULL,digits);  // pos5 == s2+10, s2[15] == '\0', "world"
  char *pos6 = strtok(NULL,digits);  // pos6 == s2+16, s2[17] == '\0', "!"
  char *pos7 = strtok(NULL,digits);  // pos7 == NULL  strtok is not thread-safe!
}
```

String manipulation

```
char *strcpy(char *tgt, const char *src)  // copy src to tgt
{
  while ( '\0' != (*tgt++ = *src++) )
     ;
  return tgt;
}

char *strncpy(char *tgt, const char *src, size_t n)  // copy src to tgt max n chars
{
  for ( int i = 0; i < n; ++i)
  {
    if ( '\0' == (tgt[i] = src[i]) ) // if there is no '\0' in src[0], ..., src[n-1]
    {                                // then tgt will be no zero terminated
      memset(tgt+i+1, '\0', n-i-1);
      break;
    }
  }
  return *tgt;
}

char *strcat(char *tgt, const char *src);   // concatenates src to tgt

char *strncat(char *tgt, const char *src, size_t n); // concatenates src to tgt

strncpy(tgt, src, n);
tgt[n-1] = '\0';    // safe way: may truncate, but tgt will be zero terminated
```

Character manipulation

```
int isalpha(int ch);
int isalnum(int ch);
int isupper(int ch);
int islower(int ch);
int isdigit(int ch);
int isxdigit(int ch);
int iscntrl(int ch);
int isgraph(int ch);
int isspace(int ch);
int isblank(int ch);
int isprint(int ch);
int ispunct(int ch);

int toupper(int ch);
int tolower(int ch);
```

```
int    atoi(const char *s);
long   atol(const char *s);
double atof(const char *s);

long   strtol(const char *s, char **p, int base);
double strtod(const char *s, char **p, int base);
```

Raw memory

```
void f()
{
  char source[1024];
  char target[1024];

  memset(source, 'x', sizeof(source));     // set source to all 'x'


  memcpy(target, source, sizeof(source));  // copy 1024 bytes from source to target
  memmove(target, source, sizeof(source)); // same as memcpy but can overlap


  void *ptr = memchr( target, 'y', sizeof(target));  // first occurance of 'y' or NULL


  int res = memcmp(target, source, sizeof(source));  // compares source and target
}
```