Warmup #1

Bill Cheng

http://merlot.usc.edu/cs402-f18

`listtest`

- **Use provided `listtest.c` and `Makefile` to create `listtest`**
  - **`listtest` must run without error and you must not change `listtest.c` and `Makefile`**
  - **They specifies how your code is expected to be used**
- **You should learn how to run `listtest` under `gdb`**

gdb listtest Exercise

- Do the following gdb exercise with listtest
  - IMPORTANT: draw picture on a piece of paper!
  - first, change "num_items=64" in DoTest() to "num_items=3"

```
make
gdb listtest
(gdb) break DoTest
(gdb) run
(gdb) n                ←— do this 5 times, you are now at call to CreateTestList()
(gdb) print list          ←— does the list look like an empty list?
(gdb) n                ←— returned from CreateTestList()
(gdb) print list          ←— does the list look like a 3-item list?
(gdb) print &(list.anchor)     ←— what's the address of the anchor?
(gdb) print list.anchor        ←— what's in the anchor?
(gdb) print *(list.anchor.next)
(gdb) print *(list.anchor.next->next)
(gdb) print *(list.anchor.next->next->next)   ←┐
                                                 │
                 this should be the last list element, ——┘
          does its next pointer point to the anchor?
```

C File I/O Review: Reading Text Input

- Read in an entire line using `fgets()`
  - especially since we know the maximum line length, according to the spec
- If a filename is given, use `fopen()` to get a file pointer (`FILE*`)

```
    FILE *fp = fopen(..., "r");
```

  - read man pages of `fopen()`
  - if a filename is not given, you will be reading from "standard input" (i.e., file descriptor 0)

```
    FILE *fp = stdin;
```

  - pass the file pointer around so that you run the same code whether you input comes from a file or `stdin`

```
    My420List list;
    if (!My402ListInit(&list)) { /* error */ }
    if (!ReadInput(fp, &list)) { /* error */ }
    if (fp != stdin) fclose(fp);
    SortInput(&list);
    PrintStatement(&list);
```

C File I/O Review: Parsing Text Input

- Read a line

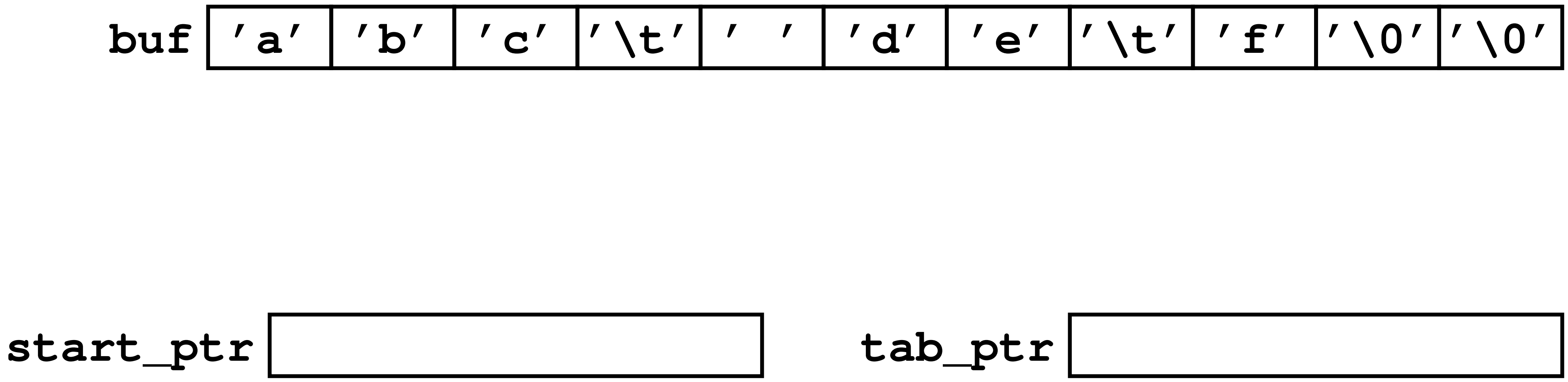
```
char buf[1026];
if (fgets(buf, sizeof(buf), fp) == NULL) {
  /* end of file */
} else {
  /* parse it */
}
```

- Parse a line according to the spec
  - find an *easy* and *correct* way to parse the line
    - according to the spec, each line must have exactly 3 `<TAB>` characters
    - I think it's easy and correct to go after this

```
char *start_ptr = buf;
char *tab_ptr = strchr(start_ptr, '\t');
if (tab_ptr != NULL) {
  *tab_ptr++ = '\0';
}
/* start_ptr now contains a
   "null-terminated string" */
```

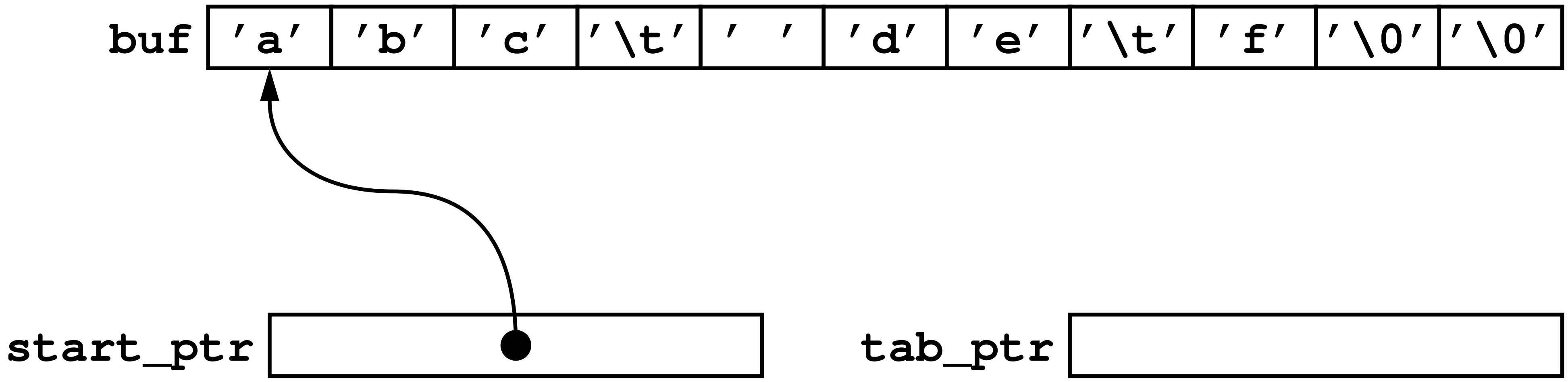
C File I/O Review: Parsing Text Input

```
char *start_ptr = buf;
char *tab_ptr = strchr(start_ptr, '\t');
if (tab_ptr != NULL) {
  *tab_ptr++ = '\0';
}
/* start_ptr now contains a
   "null-terminated string" */
```



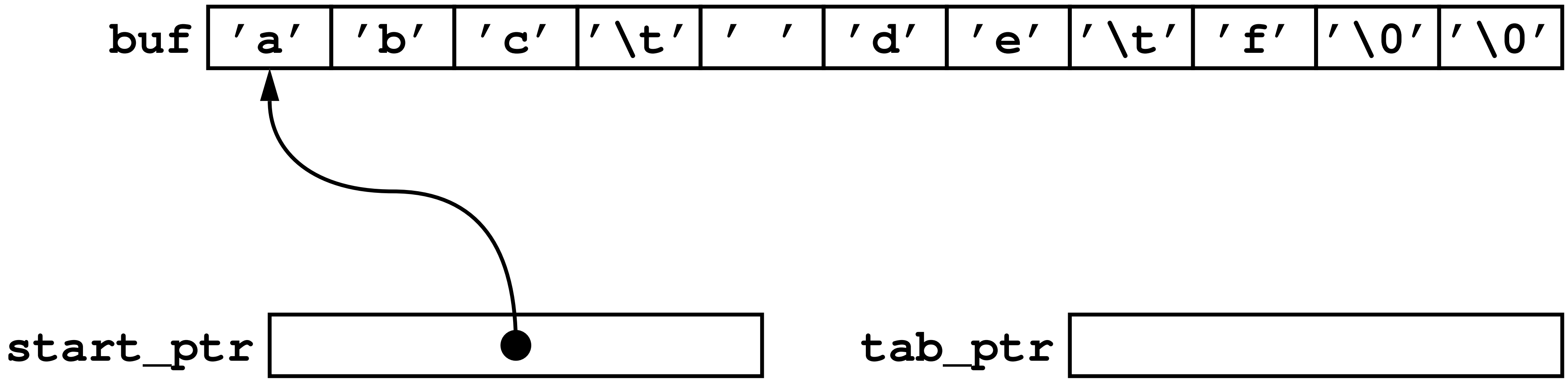
C File I/O Review: Parsing Text Input

```
char *start_ptr = buf;
char *tab_ptr = strchr(start_ptr, '\t');
if (tab_ptr != NULL) {
  *tab_ptr++ = '\0';
}
/* start_ptr now contains a
   "null-terminated string" */
```



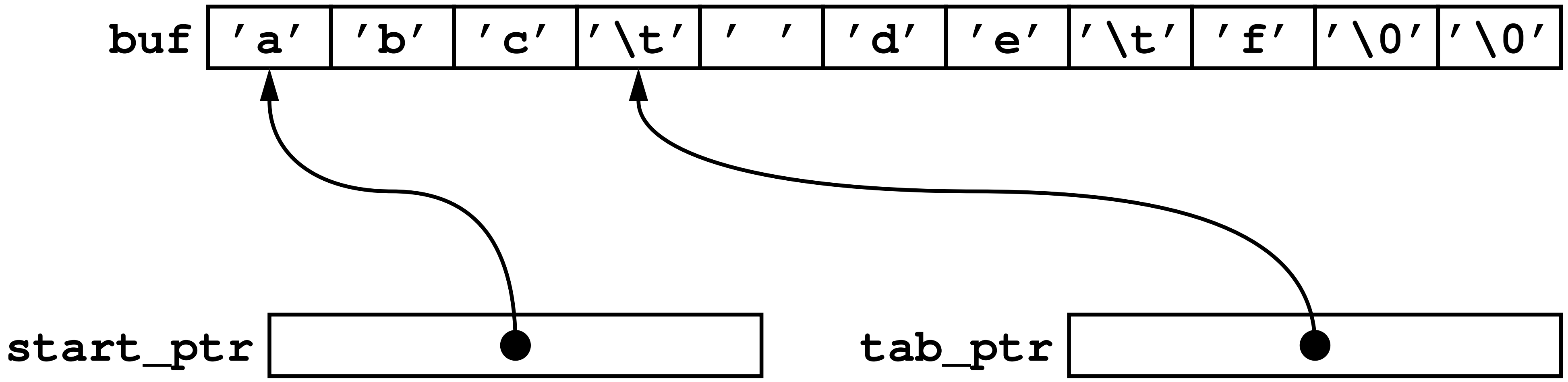
C File I/O Review: Parsing Text Input

```
char *start_ptr = buf;
char *tab_ptr = strchr(start_ptr, '\t');
if (tab_ptr != NULL) {
  *tab_ptr++ = '\0';
}
/* start_ptr now contains a
   "null-terminated string" */
```



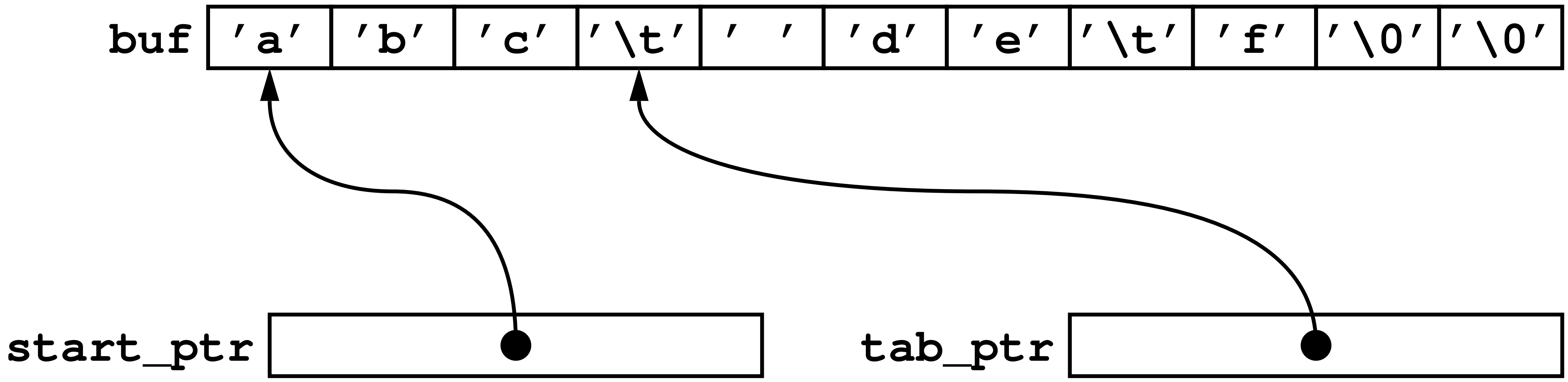
C File I/O Review: Parsing Text Input

```
char *start_ptr = buf;
char *tab_ptr = strchr(start_ptr, '\t');
if (tab_ptr != NULL) {
  *tab_ptr++ = '\0';
}
/* start_ptr now contains a
   "null-terminated string" */
```



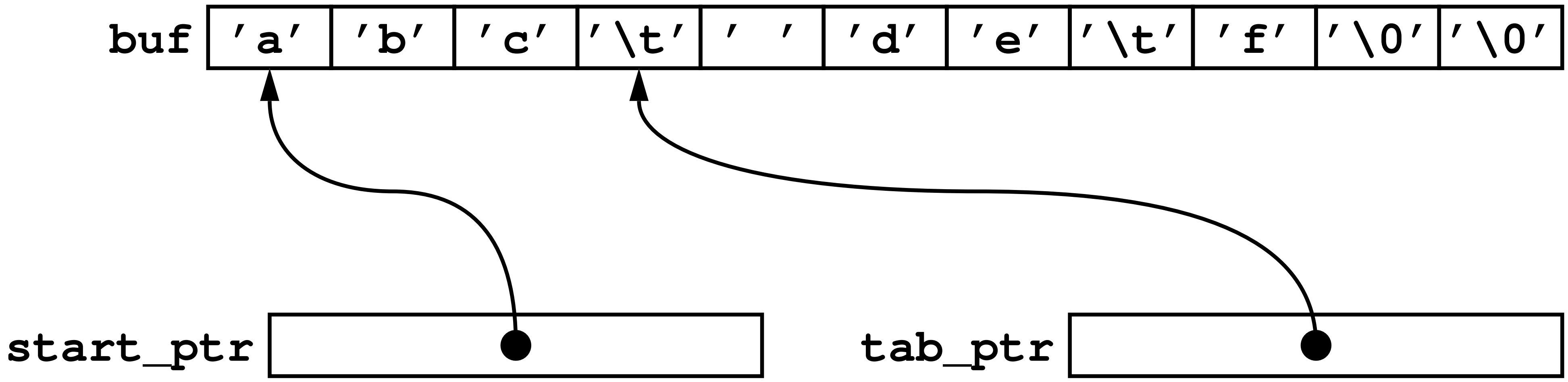
C File I/O Review: Parsing Text Input

```
char *start_ptr = buf;
char *tab_ptr = strchr(start_ptr, '\t');
if (tab_ptr != NULL) {
  *tab_ptr++ = '\0';
}
/* start_ptr now contains a
   "null-terminated string" */
```



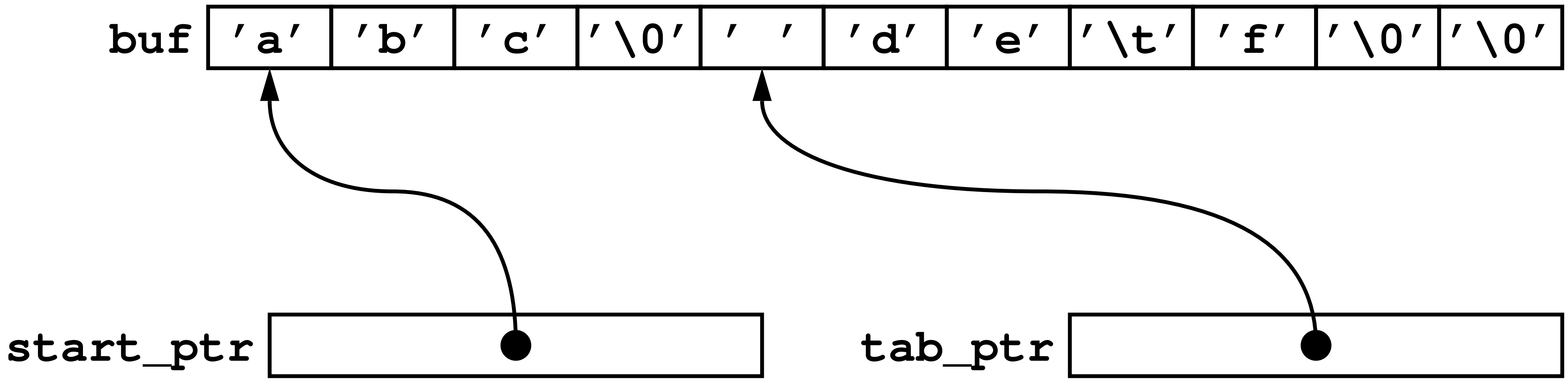
C File I/O Review: Parsing Text Input

```
char *start_ptr = buf;
char *tab_ptr = strchr(start_ptr, '\t');
if (tab_ptr != NULL) {
  *tab_ptr++ = '\0';
}
/* start_ptr now contains a
   "null-terminated string" */
```



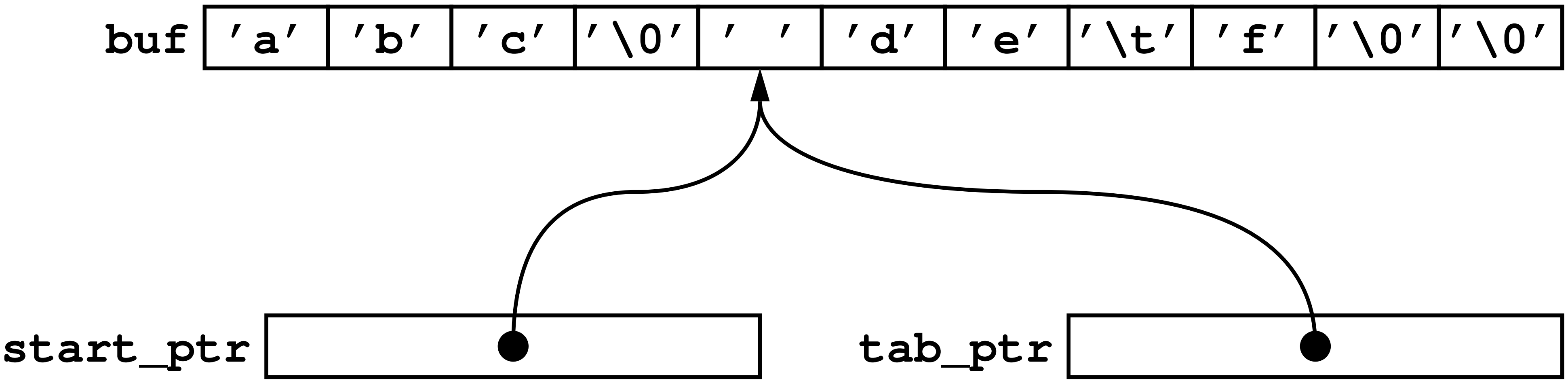
C File I/O Review: Parsing Text Input

```
char *start_ptr = buf;
char *tab_ptr = strchr(start_ptr, '\t');
if (tab_ptr != NULL) {
  *tab_ptr++ = '\0';
}
/* start_ptr now contains a
   "null-terminated string" */
```



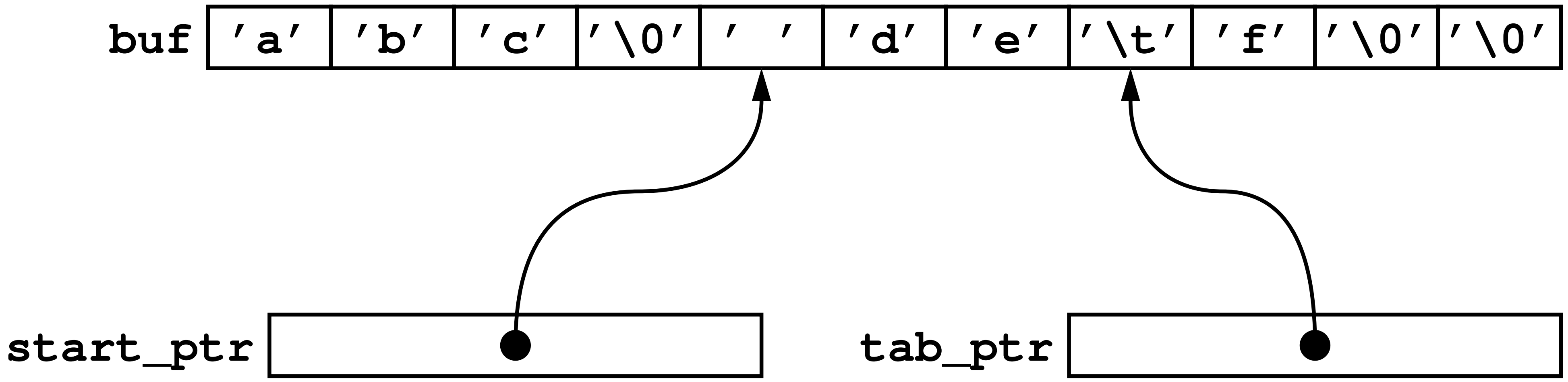
C File I/O Review: Parsing Text Input (2nd Iteration)

```
start_ptr = tab_ptr;
tab_ptr = strchr(start_ptr, '\t');
if (tab_ptr != NULL) {
  *tab_ptr++ = '\0';
}
/* start_ptr now contains a
   "null-terminated string" */
```



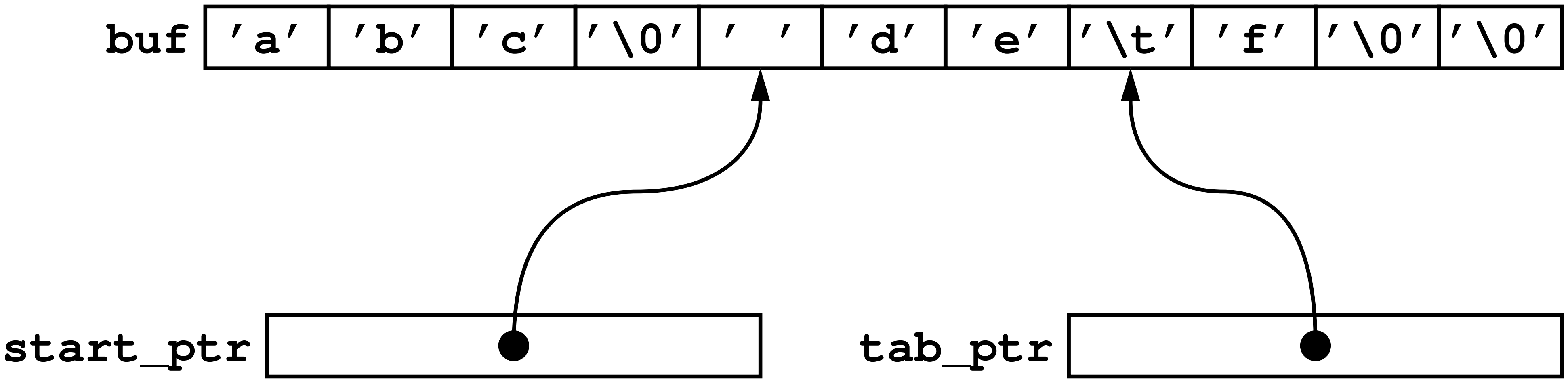
C File I/O Review: Parsing Text Input (2nd Iteration)

```
    start_ptr = tab_ptr;
--> tab_ptr = strchr(start_ptr, '\t');
    if (tab_ptr != NULL) {
      *tab_ptr++ = '\0';
    }
    /* start_ptr now contains a
       "null-terminated string" */
```



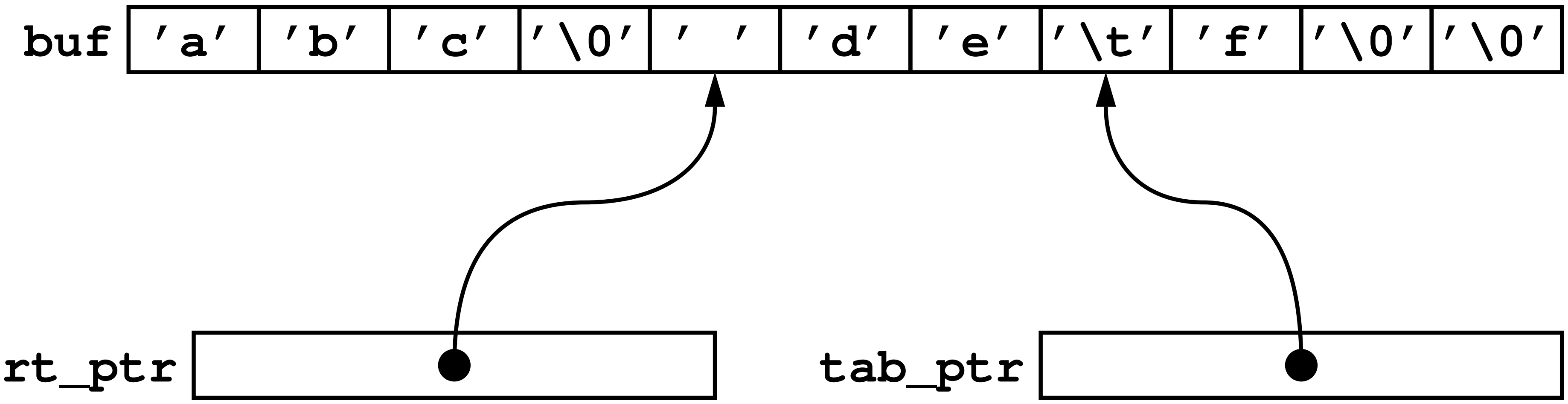
C File I/O Review: Parsing Text Input (2nd Iteration)

```
start_ptr = tab_ptr;
tab_ptr = strchr(start_ptr, '\t');
if (tab_ptr != NULL) {
  *tab_ptr++ = '\0';
}
/* start_ptr now contains a
   "null-terminated string" */
```



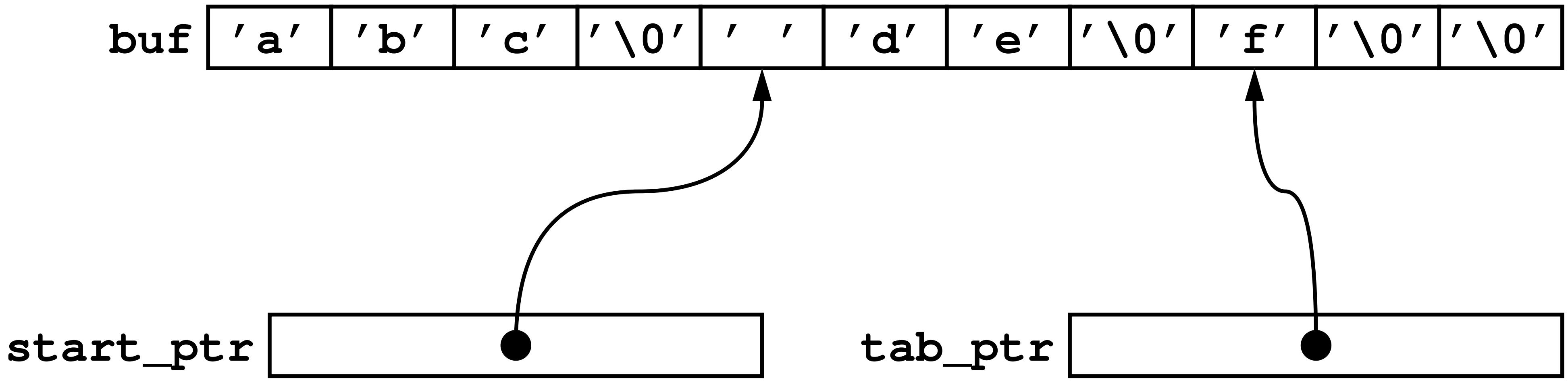
C File I/O Review: Parsing Text Input (2nd Iteration)

```
start_ptr = tab_ptr;
tab_ptr = strchr(start_ptr, '\t');
if (tab_ptr != NULL) {
  *tab_ptr++ = '\0';
}
/* start_ptr now contains a
   "null-terminated string" */
```



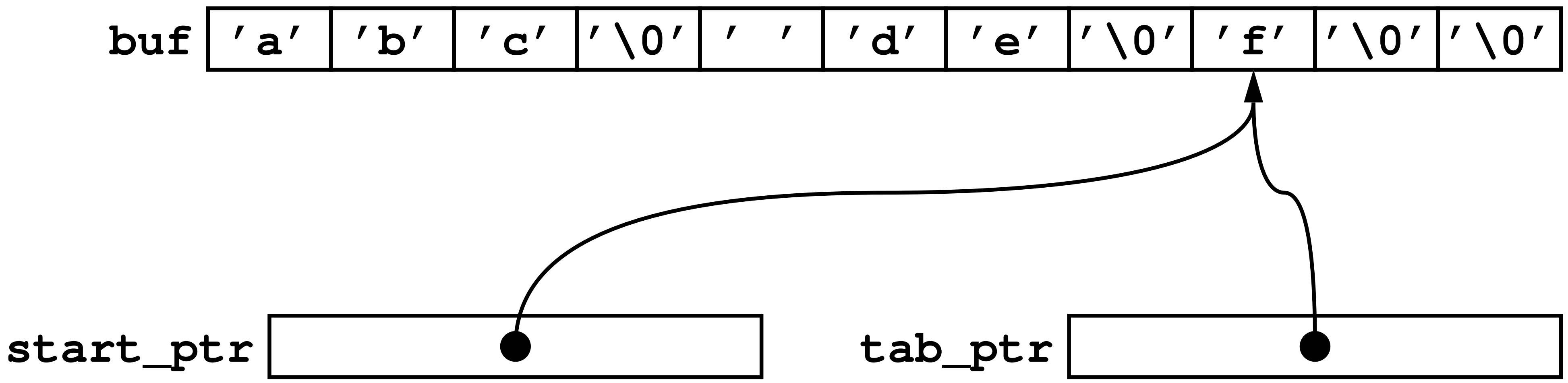
C File I/O Review: Parsing Text Input (2nd Iteration)

```
start_ptr = tab_ptr;
tab_ptr = strchr(start_ptr, '\t');
if (tab_ptr != NULL) {
  *tab_ptr++ = '\0';
}
/* start_ptr now contains a
   "null-terminated string" */
```



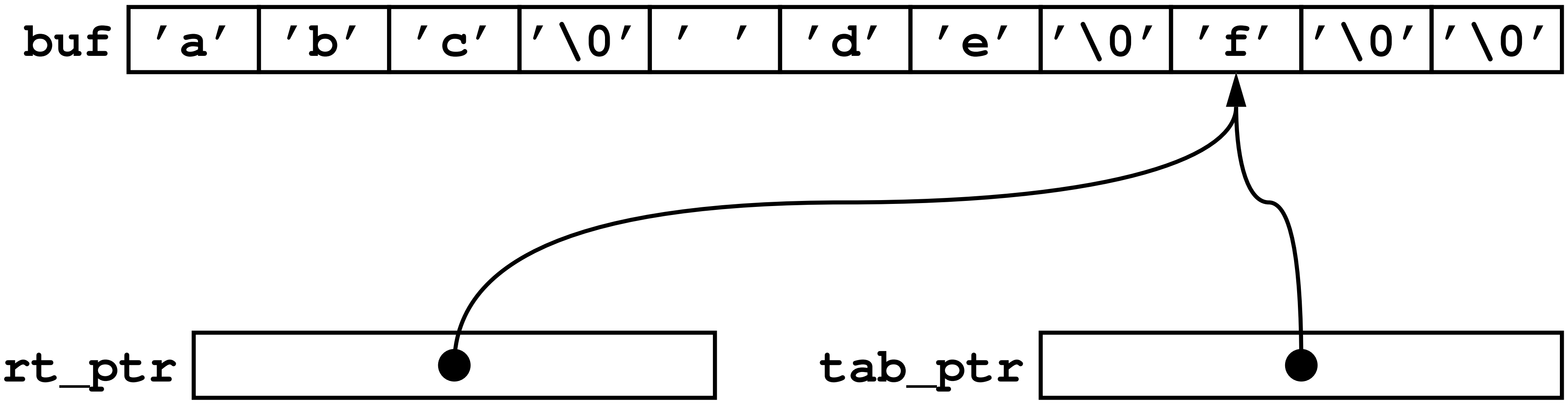
C File I/O Review: Parsing Text Input (3rd Iteration)

```
start_ptr = tab_ptr;
tab_ptr = strchr(start_ptr, '\t');
if (tab_ptr != NULL) {
  *tab_ptr++ = '\0';
}
/* start_ptr now contains a
   "null-terminated string" */
```



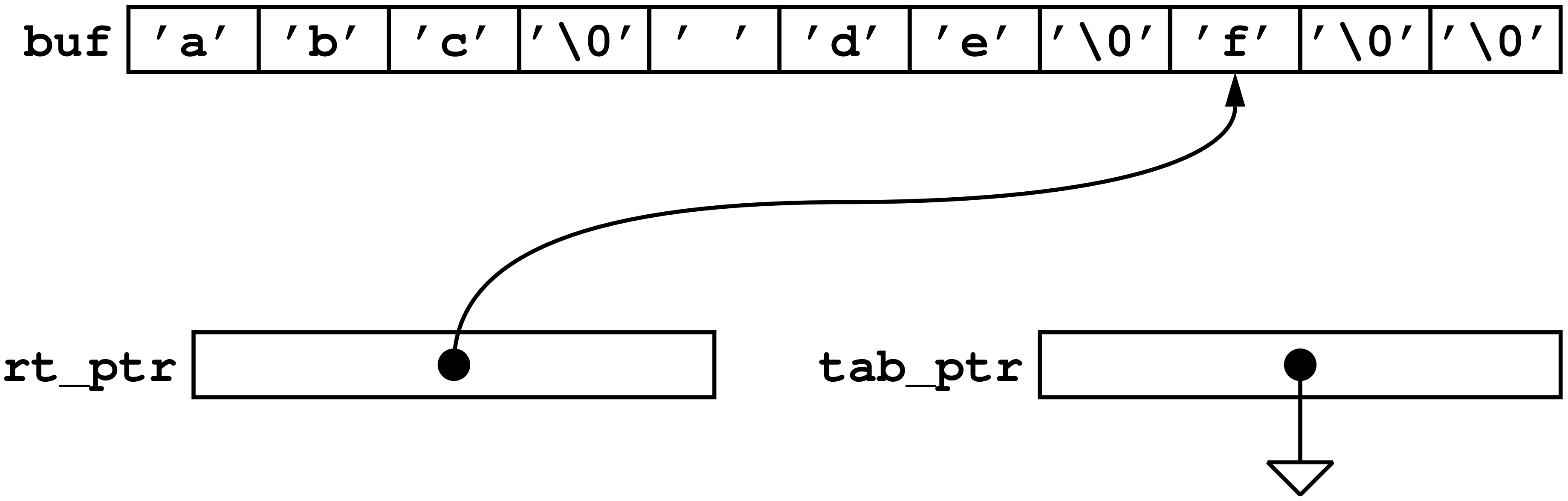
C File I/O Review: Parsing Text Input (3rd Iteration)

```
start_ptr = tab_ptr;
tab_ptr = strchr(start_ptr, '\t');
if (tab_ptr != NULL) {
  *tab_ptr++ = '\0';
}
/* start_ptr now contains a
   "null-terminated string" */
```



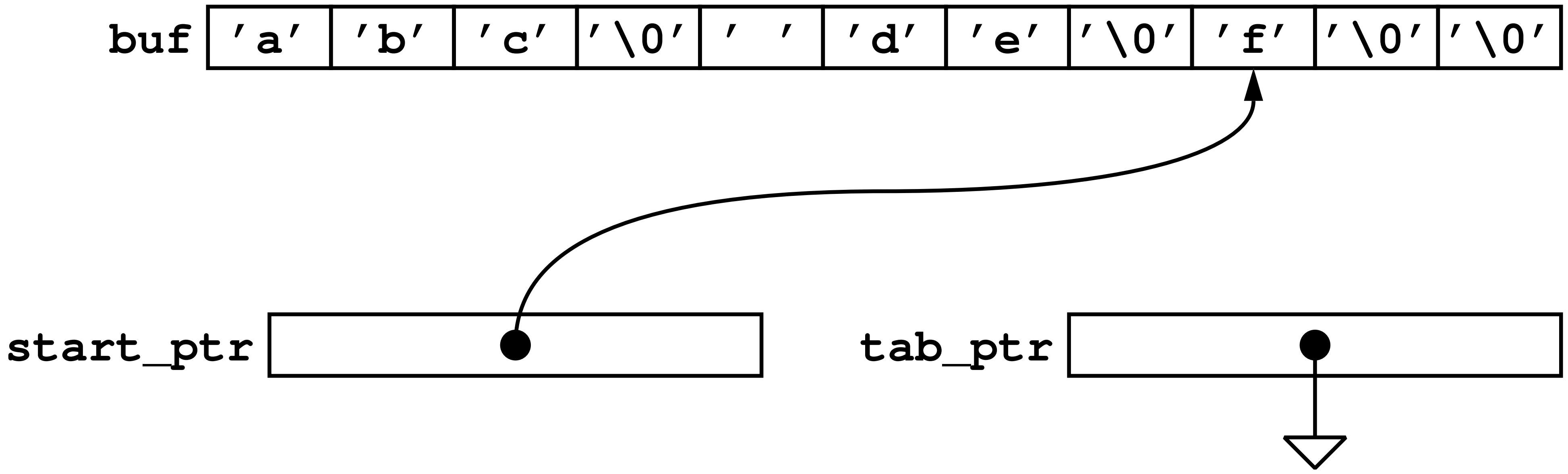
C File I/O Review: Parsing Text Input (3rd Iteration)

```
start_ptr = tab_ptr;
tab_ptr = strchr(start_ptr, '\t');
if (tab_ptr != NULL) {
  *tab_ptr++ = '\0';
}
/* start_ptr now contains a
   "null-terminated string" */
```



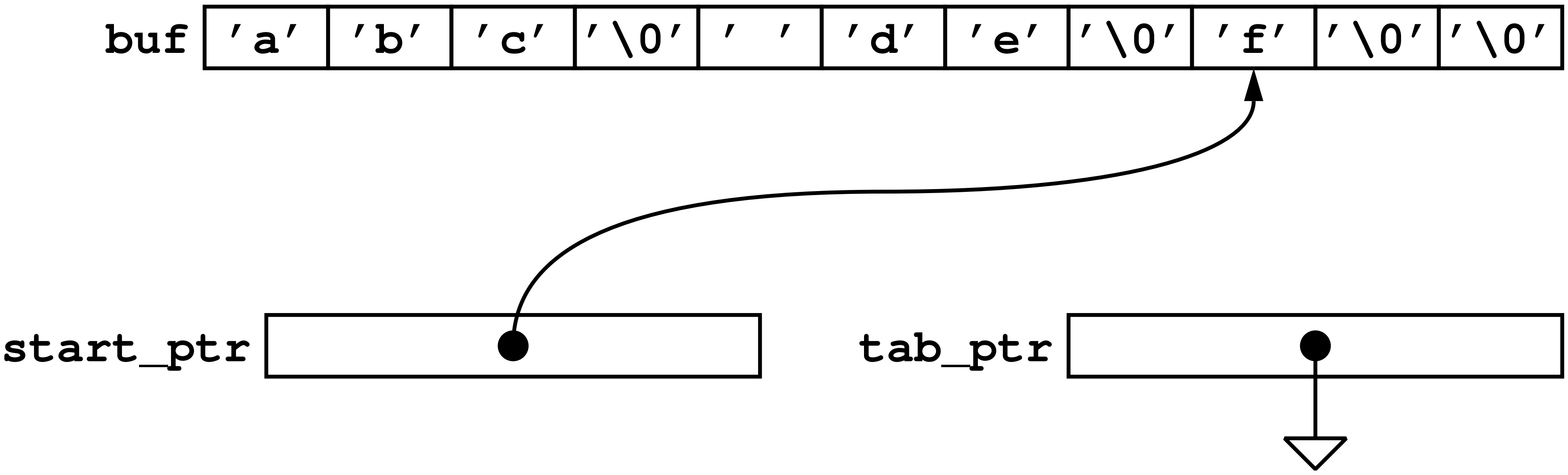
C File I/O Review: Parsing Text Input (3rd Iteration)

```
start_ptr = tab_ptr;
tab_ptr = strchr(start_ptr, '\t');
if (tab_ptr != NULL) {
  *tab_ptr++ = '\0';
}
/* start_ptr now contains a
   "null-terminated string" */
```



C File I/O Review: Parsing Text Input (3rd Iteration)

```
start_ptr = tab_ptr;
tab_ptr = strchr(start_ptr, '\t');
if (tab_ptr != NULL) {
  *tab_ptr++ = '\0';
}
/* start_ptr now contains a
   "null-terminated string" */
```



Warmup #1

- I'm giving you a lot of details on how to do things in C
  - this is the first and last assignment that I will do this!
  - you must learn C on your own
- Read man pages
- Ask questions in class Google Group
  - or send e-mail to me
- Come to office hours, especially if you are stuck

Warmup #1 - Miscellaneous Requirements

- Run your code against the *grading guidelines*
  - must not change the test program
- You must not use any *external code fragments*
- You must not use *array* to implement any list functions
  - must use pointers
- If input file is large, you must not read the whole file into into a large memory buffer
- It's important that every byte of your data is read and written correctly.
  - `diff` commands in the grading guidelines must *not* produce *any* output or you will not get credit
- Please see Warmup #1 spec for additional details
  - please read the *entire* spec *yourself*

Demos

- Explain how `Makefile` work

```
listtest: listtest.o my402list.o
        gcc -o listtest -g listtest.o my402list.o

listtest.o: listtest.c my402list.h
        gcc -g -c -Wall listtest.c

my402list.o: my402list.c my402list.h
        gcc -g -c -Wall my402list.c

clean:
        rm -f *.o listtest
```

- Explain why the above `Makefile` satisfied *separate compilation* requirement
- Explain how to convert the above `Makefile` to a `Makefile` students can use for part (2)

Demos

- Instead of using `FileZilla`, just use `putty`
  - actually, if you have Ubuntu 14.04 installed or if you are using a Mac, you should use `scp`
  - if not, use `putty` as the `ssh` client and use `sftp` to transfer files between your laptop and `nunki`
- Demonstrate how to use `putty` to `ssh` into `nunki`
- Create "hello.c" on your laptop

```
#include <stdio.h>

int main(int argc, char *argv[])
{
    printf("Hello World!\n");
    return 0;
}
```

- Demonstrate how create a `warmup1` subdirectory on `nunki` and use `sftp` to transfer "`hello.c`" into that directory

Demos

- Demonstrate how to run gcc

```
gcc -g -Wall hello.c
```

- Demonstrate how to debug `a.out`

```
gdb a.out
(gdb) break main
(gdb) run abc xyz
(gdb) print argc
(gdb) print argv[0]
(gdb) print argv[1]
(gdb) print argv[2]
(gdb) print &argc
(gdb) next
```

- Demonstrate how to copy and paste from the grading guidelines
- Demonstrate how to copy and paste the `bsubmit` command from the spec

Unix Commands

- Walk through and demonstrate the commands on the Unix Command Line Reference web page
  - click on the "summary of some commonly used Unix commands" link at the bottom of the class home page

```
ls
cat
more
pwd
mkdir (directory name)
cd
cp (src file path) (dest file path)
mv (src file path) (dest file path)
man (cmd name)
rm (file path)
rmdir (empty directory name)
ps
kill (proc id)
pico/nano (file path)
exit
```