22C:050 solutions for HW#6

8.3

```
char buffer[BLOCK_SIZE];
char* buffer_loc=0;
int pos=0;

read(f,c)
{
    if(pos= buffer_loc)
    {
        readblock(f, buffer, BLOCK_SIZE);
        buffer_loc += BLOCK_SIZE;
    }
    c=buffer[pos% BLOCK_SIZE];
    pos++;
}
```

8.5

(A common storage management discipline on such a system establishes all of memory as a stack; whenever a program wants to load another program, it loads it immediately beyond itself in memory, thus effectively pushing one program on the stack. If procedure calls are done using a stack, this may require a second stack, as is shown in the map of the memory usage on such a small system given in Figure 8.6)
Refer to Figure 8.6. Map of the memory use in a typical small system.

1. check before loading each program
2. check before each procedure calls (stack frame and local variables)
3. check before each dynamic memory allocation (this usually happens in heap, so I do not think it is proper here)

9.3

```
void read( struct filevariable * f, char c )
{
    switch (f->class) {
    case keyboard:    keyread( &(f->variant.keyfile), c ); break;
    case display:     dispread( &(f->variant.dispfile), c ); break;
    case tape:        taperead( &(f->variant.tapefile), c ); break;
    }
```

```
}

void readblock( struct filevariable * f, char buffer[], int size=BLOCK_SIZE)
{
       int i=0;
        switch (f->class) {
              case keyboard:
                                          while(i<size)
                                          {
                                               keyread( &(f->variant.keyfile), c );
                                               i++;
                                           }
                                          break;
              case display:
                                          while(i<size)
                                          {
                                               dispread( &(f->variant.keyfile), c );
                                               i++;
                                           }
                                           break;
              case tape:
                                           tapewriteblock(&(f->variant.keyfile, buffer, size);
                                           break;
              }
   }
```

9.5

Note that the operation delete will erase words from the current position, which is different from backspace operation.

```
void comreadline( struct filevariable * f, char buf[], int len );
{
          int p = 0;

          do {
                    char ch = f->read(f);
                    …
                    else if (ch == '\x7F') { /* character is backspace */
                            ch = f->read(f);
                                   if(ispace(ch)|| ispunct(ch))ch = f->read(f);
                            else
                                   while(isdigit(ch)||isalpha(ch))ch = f->read(f);
                     }
```

```
        } while ((ch != '\n') && (ch != '\r'));
        …
}
```