```
/* fork1.c - Demonstrate the fork() */

#include <stdio.h>
#include <unistd.h>

main()
{
      int i;
      printf("Ready to fork...\n");
      i=fork();
      printf("Fork returned %d\n",i);
}
```

```
/* fork2.c - Child process can run different code than parent
process */

#include <stdio.h>
#include <unistd.h>

main()
{
      int i,j;
      j=0;

      printf("Ready to fork...\n");

      i=fork();

      if (i == 0)
        {
        printf("The child executes this code.\n");
        for (i=0; i<5; i++)
          j=j+i;
        printf("Child j=%d\n",j);
        }
      else
        {
        printf("The parent executes this code.\n");
        for (i=0; i<3; i++)
          j=j+i;
        printf("Parent j=%d\n",j);
        }
}
```

```
/* fork3.c -  fork can be called iteratively */

#include <stdio.h>
#include <unistd.h>

main()
{
      int i;

      i=getpid();
      printf("Parent=%d\n",i);
      fork();
      fork();
      i=getpid();
      printf("Who am I? %d\n",i);
}
```

```
/*
 * sh1.c: sample version 1 of a UNIX command shell/interpreter.
 *
 */

#include <stdio.h>
#include <stdlib.h>
#include <string.h>

int main()
{
    char line[256];
    char prompt[] = "sh1 % ";

    /* spit out the prompt */
    printf("%s", prompt );

    /* Try getting input. If error or EOF, exit */
    while( fgets(line, sizeof line, stdin) != NULL )
    {
      /* fgets leaves '\n' in input buffer. ditch it */
      line[strlen(line)-1] = '\0';

      /* This is where I take the easy route.
         fork(), execvp() & co. go here in your version*/

      system( line );

      printf("%s", prompt );

    }
    return 0;
}
```