```
// 4.11.a

#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <sys/types.h>

int main() {
 int fd[2]; char c;

 pipe(fd);

 if (fork() == 0) {
   close(fd[0]);
   printf("CHILD PID: %d\n", getpid());
   write(fd[1], &c, 1); // unblock PARENT if it is blocked
   exit(0);
 }

 close(fd[1]);
 read(fd[0], &c, 1); // if PARENT is fast it will block here
 printf("PARENT PID: %d\n", getpid());

 return(0);
}
```

```
// 4.11.b

#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <sys/types.h>

int main() {
 int fd[2]; char c;

 pipe(fd);

 if (fork() == 0) {
   close(fd[1]);
   read(fd[0], &c, 1); // if CHILD is fast it will block here
   printf("CHILD PID: %d\n", getpid());
   exit(0);
 }

 close(fd[0]);
 printf("PARENT PID: %d\n", getpid());
 write(fd[1], &c, 1); // unblock CHILD if it is blocked

 return(0);
}
```

```
// 4.12

#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <sys/types.h>

void count(int min, int max) {
 int i;
 for(i=min; i<=max; i++)
      printf("pid = %d\t i = %d\n", getpid(), i);
}

int main() {
 int fd1[2], fd2[2]; char c;

 pipe(fd1); pipe(fd2);

 if (fork()==0) {
      close(fd1[1]); close(fd2[0]);

      read(fd1[0], &c, 1); // self-block
      count(4,6);
      write(fd2[1], &c, 1); // unlock parent

      read(fd1[0], &c, 1); // self-block
      count(10,12);
      write(fd2[1], &c, 1); // unlock parent

      exit(0);
 }

 close(fd1[0]); close(fd2[1]);

 count(1,3);
 write(fd1[1], &c, 1); // unlock child

 read(fd2[0], &c, 1); // self-block
 count(7,9);
 write(fd1[1], &c, 1); // unlock child

 read(fd2[0], &c, 1); // self-block
 count(13,15);

 return(0);
}
```

```
// 4.13.a

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

int main()
{
 int fd[2], num;

 pipe(fd);

 if (fork()==0) {
      do {
            scanf("%d", &num);
            write(fd[1], &num, sizeof(int));
            read(fd[0], &num, sizeof(int));
            printf("%d\n", num);
      } while (num != 1);
      exit(0);
 }

 do {
      read(fd[0], &num, sizeof(int));
      num ++;
      write(fd[1], &num, sizeof(int));
 } while (num != 1);

 return(0);
}
```

```
// 4.13.b; #include directives are the same as in 4.13.a

int main()
{
 int fd1[2], fd2[2], num;

 pipe(fd1); pipe(fd2);

 if (fork()==0) {
      close(fd1[0]); close(fd2[1]);
      do {
            scanf("%d", &num);
            write(fd1[1], &num, sizeof(int));
            read(fd2[0], &num, sizeof(int));
            printf("%d\n", num);
      } while (num != 1);
      exit(0);
 }

 close(fd1[1]); close(fd2[0]);
 do {
      read(fd1[0], &num, sizeof(int));
      num ++;
      write(fd2[1], &num, sizeof(int));
 } while (num != 1);

 return(0);
}
```

```
// 4.14


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

int main() {
 int fd1[2], fd2[2], fd3[2], num;

 // pipe fd1 : to block the 1st Child
 // pipe fd2 : to block the 2nd Child
 // pipe fd3 : to block the Parent
 pipe(fd1); pipe(fd2); pipe(fd3);

 if (fork()==0) { // 1st child
     close(fd1[1]); close(fd2[0]); close(fd3[0]); close(fd3[1]);
     do {
         scanf("%d", &num);
         write(fd2[1], &num, sizeof(int)); // unblock 2nd child
         read(fd1[0], &num, sizeof(int));  // block 1st child
         printf("%d\n", num);
     } while (num!=2);
     exit(0);
 }

 if (fork()==0) { // 2nd child
     close(fd2[1]); close(fd3[0]); close(fd1[0]); close(fd1[1]);
     do {
         read(fd2[0], &num, sizeof(int));  // block 2nd child
         num++;
         write(fd3[1], &num, sizeof(int)); // unblock parent
     } while (num!=1);
     exit(0);
 }

 close(fd3[1]); close(fd1[0]); close(fd2[0]); close(fd2[1]);
 do { // parent
     read(fd3[0], &num, sizeof(int));  // block parent
     num++;
     write(fd1[1], &num, sizeof(int)); // unblock 1st child
 } while (num!=2);

 return(0);
}
```

```
// 4.15.a: solution fully based on an unnamed pipe

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

int sum(int min, int max)
{
 int result=0, i;

 for (i=min; i<=max; i++)
     result += i;

 return(result);
}

main()
{
 int fd[2], sum_child, sum_parent;

 pipe(fd);

 if ( fork()==0 ) { // Child
    sum_child=sum(51,100);
    close(fd[0]);
    write(fd[1],&sum_child,sizeof(int));
    exit(0);
 }

 sum_parent=sum(1,50);
 close(fd[1]);
 read(fd[0],&sum_child,sizeof(int)); // blocking reading
 printf("%d\n", sum_parent + sum_child);
}
```

```
// 4.15.b: solution based on shared memory for data
// exchange and an unnamed pipe for synchronization

#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <sys/ipc.h>
#include <sys/shm.h>
#include <sys/types.h>

int sum(int min, int max)
{
 int result=0, i;

 for (i=min; i<=max; i++)
     result += i;

 return(result);
}

main()
{
 int shmid, *shmaddr, sum_parent;
 int fd[2]; char c;

 shmid=shmget(IPC_PRIVATE, sizeof(int), 00600);
 shmaddr=(int*)shmat(shmid, NULL, 0);

 pipe(fd);

 if ( fork()==0 ) { // Child
    shmaddr[0]=sum(51,100);
    close(fd[0]);
    write(fd[1],&c,1);
    exit(0);
 }

 sum_parent=sum(1,50);
 close(fd[1]);
 read(fd[0],&c,1); // blocking waiting
 printf("%d\n", sum_parent + shmaddr[0]);
 shmctl(shmid, IPC_RMID, 0);
}
```

// 4.16.a: solution fully based on unnamed pipes

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

int sum(int min, int max)
{
 int result=0, i;

 for (i=min; i<=max; i++)
     result += i;

 return(result);
}

main()
{
 int fd[2], i, sum_partial, sum_final=0;

 pipe(fd);

 for (i=0; i<10; i++) {
     if (fork() == 0) { // i'th CHILD
          sum_partial=sum(i*10+1, i*10+10);
          close(fd[0]); write(fd[1],&sum_partial,sizeof(int));
          exit(0);
    }
 }

 // PARENT

 close(fd[1]);
 for (i=0; i<10; i++) {
     read (fd[0],&sum_partial,sizeof(int)); // blocking reading
     sum_final = sum_final + sum_partial;
 }

 printf("%i\n", sum_final);
}
```

```
// 4.16.b: solution based on shared memory for data
// exchange and an unnamed pipe for synchronization

#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <sys/ipc.h>
#include <sys/shm.h>
#include <sys/types.h>

int sum(int min, int max)
{
 int result=0, i;

 for (i=min; i<=max; i++)
     result += i;

 return(result);
}


main()
{
 int shmid_sums, *ptr_sums;
 int i, sum_final=0;
 int fd[2]; char c;

 shmid_sums=shmget(IPC_PRIVATE, 10*sizeof(int), 00600);
 ptr_sums=(int*)shmat(shmid_sums, NULL, 0);

 pipe(fd);

 for (i=0; i<10; i++) {
     if (fork() == 0) { // i'th CHILD
          ptr_sums[i]=sum(i*10+1, i*10+10);
          close(fd[0]); write(fd[1], &c, 1);
          exit(0);
    }
 }

 // PARENT

 close(fd[1]);
 for (i=0; i<10; i++) read (fd[0], &c, 1); // blocking waiting
 // alternative: { char cc[10]; read (fd[0], cc, 10); }

 for (i=0; i<10; i++)
     sum_final = sum_final + ptr_sums[i];

 printf("%i\n", sum_final);
 shmctl(shmid_sums, IPC_RMID, 0);
}
```

```
// 4.16.c: solution based on shared memory for data exchange and
// an unnamed pipe for synchronization, but independent of the
// order in which CHILDREN finish; each child writes in the pipe
// the index of the shared memory cell in which puts its partial
// sum; for each cell index read from the pipe, the PARENT
// collects the partial sum stored in the shared memory cell

#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <sys/ipc.h>
#include <sys/shm.h>
#include <sys/types.h>

int sum(int min, int max)
{
 int result=0, i;

 for (i=min; i<=max; i++)
     result += i;

 return(result);
}

main()
{
 int shmid_sums, *ptr_sums;
 int i, sum_final=0;
 int fd[2]; int j;

 shmid_sums=shmget(IPC_PRIVATE, 10*sizeof(int), 00600);
 ptr_sums=(int*)shmat(shmid_sums, NULL, 0);

 pipe(fd);

 for (i=0; i<10; i++) {
     if (fork() == 0) { // i'th CHILD
          ptr_sums[i]=sum(i*10+1, i*10+10);
          close(fd[0]); write(fd[1], &i, sizeof(int));
          exit(0);
    }
 }

 // PARENT

 close(fd[1]);
 for (i=0; i<10; i++) {
     read (fd[0], &j, sizeof(int)); // blocking waiting
     sum_final = sum_final + ptr_sums[j];
 }

 printf("%i\n", sum_final);
 shmctl(shmid_sums, IPC_RMID, 0);
}
```

```
// 4.17

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

int fd[11][2];

void close_fds_not_needed(int i) {
 int j;

 for (j=0; j<11; j++) {
     if (j == i) close(fd[j][1]);
     else if (j == (i+1)%11) close(fd[j][0]);
     else { close(fd[j][0]); close(fd[j][1]); }
 }
}

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

 for (i=0; i<11; i++)
     pipe(fd[i]);

 for (i=1; i<11; i++)
     if (fork()==0) { // Child
         close_fds_not_needed(i);
         read(fd[i][0], &sum, sizeof(int));
         sum = sum + (i*i);
         //printf("Child %d: sum = %d\n", i, sum);
         write(fd[(i+1)%11][1], &sum, sizeof(int));
         exit(0);
     }

 close_fds_not_needed(0);
 write(fd[1][1], &sum, sizeof(int));
 read(fd[0][0], &sum, sizeof(int));
 printf("%d\n", sum);

 return(0);
}
```

```
// 4.18.a

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

int main() {
 int N, *A, i, odd, odds, fd[2];

 scanf("%d", &N);
 A=(int*)malloc(N*sizeof(int));

 pipe(fd);

 srandom(getpid());
 for (i=0; i<N; i++)
     A[i] = random();

 for (i=0; i<N; i++) {
     if (fork() == 0) {
          odd = (A[i] % 2 != 0) ? 1 : 0;
          printf("A[%d]=%d\todd=%d\n", i, A[i], odd);
          close(fd[0]); write(fd[1], &odd, sizeof(int));
          free(A); exit(0);
     }
 }

 close(fd[1]); odds=0;
 for (i=0; i<N; i++) {
     read(fd[0], &odd, sizeof(int));
     odds += odd;
 }
 printf("odds: %d\n", odds);

 free(A); return(0);
}
```

```
// 4.18.b

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

int main() {
 int N, *A, i, num, odd, odds, fd1[2], fd2[2];

 scanf("%d", &N);

 pipe(fd1); pipe(fd2);

 for (i=0; i<N; i++) {
     if (fork() == 0) {
          close(fd1[1]); read(fd1[0], &num, sizeof(int));
          odd = (num % 2 != 0) ? 1 : 0;
          printf("num=%d\todd=%d\n", num, odd);
          close(fd2[0]); write(fd2[1], &odd, sizeof(int));
          exit(0);
     }
 }

 A=(int*)malloc(N*sizeof(int));

 srandom(getpid()); close(fd1[0]);
 for (i=0; i<N; i++) {
     A[i] = random(); // num = random();
     write(fd1[1], &A[i], sizeof(int));
     // write(fd1[1], &num, sizeof(int));
 }

 close(fd2[1]); odds=0;
 for (i=0; i<N; i++) {
     read(fd2[0], &odd, sizeof(int));
     odds += odd;
 }
 printf("odds: %d\n", odds);

 free(A); return(0);
}
```

```
// 4.19.a

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

int main()
{
 int pids[10], i, fd1[2], fd2[2], fd3[3]; char c;

 // pipe fd1 : to block odd Children
 // pipe fd2 : to block even Children
 // pipe fd3 : to block the Parent
 pipe(fd1); pipe(fd2); pipe(fd3);

 for (i=0; i<10; i++) {
     pids[i]=fork();
     if (pids[i] == 0) {
         close(fd3[0]);
         if (getpid() % 2 == 0) {
             close(fd2[1]); close(fd1[0]); close(fd1[1]);
             read(fd2[0], &c, sizeof(char));
         }
         else {
             close(fd1[1]); close(fd2[0]); close(fd2[1]);
             read(fd1[0], &c, sizeof(char));
         }
         printf("%d\n", getpid());
         write(fd3[1], &c, sizeof(char));
         exit(0);
     }
 }

 close(fd1[0]); close(fd2[0]); close(fd3[1]);

 for (i=0; i<10; i++) { // unblock children with even PID
     if (pids[i] % 2 == 0) {
         write(fd2[1], &c, sizeof(char));
     }
 }

 for (i=0; i<10; i++) { // await for children with even PID
     if (pids[i] % 2 == 0) {
         read(fd3[0], &c, sizeof(char));
     }
 }

 for (i=0; i<10; i++) { // unblock children with odd PID
     if (pids[i] % 2 != 0) {
         write(fd1[1], &c, sizeof(char));
     }
 }

 for (i=0; i<10; i++) { // await for children with odd PID
     if (pids[i] % 2 != 0) {
         read(fd3[0], &c, sizeof(char));
     }
 }

 printf("parent: done\n");
 return(0);
}
```

```
// 4.19.b

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

int main()
{
 int num_evens=0, num_odds=0, pid, i, fd1[2], fd2[2], fd3[3]; char c;

 // pipe fd1 : to block odd Children
 // pipe fd2 : to block even Children
 // pipe fd3 : to block the Parent
 pipe(fd1); pipe(fd2); pipe(fd3);

 for (i=0; i<10; i++) {
     pid=fork();
     if (pid == 0) {
         close(fd3[0]);
         if (getpid() % 2 == 0) {
             close(fd2[1]); close(fd1[0]); close(fd1[1]);
             read(fd2[0], &c, sizeof(char));
         }
         else {
             close(fd1[1]); close(fd2[0]); close(fd2[1]);
             read(fd1[0], &c, sizeof(char));
         }
         printf("%d\n", getpid());
         write(fd3[1], &c, sizeof(char));
         exit(0);
     }
     if (pid % 2 == 0) num_evens++; else num_odds++;
 }

 close(fd1[0]); close(fd2[0]); close(fd3[1]);

 for (i=0; i<num_evens; i++) { // unblock children with even PID
     write(fd2[1], &c, sizeof(char));
 }

 for (i=0; i<num_evens; i++) { // await for children with even PID
     read(fd3[0], &c, sizeof(char));
 }

 for (i=0; i<num_odds++; i++) { // unblock children with odd PID
     write(fd1[1], &c, sizeof(char));
 }

 for (i=0; i<num_odds++; i++) { // await for children with odd PID
     read(fd3[0], &c, sizeof(char));
 }

 printf("parent: done\n");
 return(0);
}
```