```
// 2.1.a

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

main()
{
 int shmid; char *flag;

 shmid=shmget(IPC_PRIVATE, sizeof(char), 00600);
 flag=(char*)shmat(shmid, NULL, 0);
 *flag=0;

 if (fork() == 0) {
   printf("CHILD PID: %d\n", getpid());
   *flag=1; // unblock parent
   exit(0);
 }

 while (*flag != 1); // block parent
 printf("PARENT PID: %\n", getpid());
 shmctl(shmid, IPC_RMID, NULL);
}
```

```
// 2.1.b

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

main()
{
 int shmid; char *flag;

 shmid=shmget(IPC_PRIVATE, sizeof(char), 00600);
 flag=(char*)shmat(shmid, NULL, 0);
 *flag=0;

 if (fork() == 0) {
   while (*flag != 1); // block child
   printf("CHILD PID: %d\n", getpid());
   shmctl(shmid, IPC_RMID, NULL);
   exit(0);
 }

 printf("PARENT PID: %d\n", getpid());
 *flag=1; // unblock child
}
```

```
// 2.2

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

void count(int min, int max)
{
 for (int i=min; i<=max; i++) {
     printf("%d : %d\n", getpid(), i);
     sleep(1);
 }
}

int main()
{
 int shmid; char *flag;

 shmid=shmget(IPC_PRIVATE, sizeof(char), 00600);
 flag=(char*)shmat(shmid, NULL, 0);
 *flag=0; // 0 => parent turn; 1 => child turn

 if (fork()==0) { // Child

     while(*flag != 1); // wait for my turn
     count(4,6);
     *flag = 0; // pass turn to parent

     while(*flag != 1); // wait for my turn
     count(10,12);
     *flag = 0; // pass turn to parent

     exit(0);
 }

 count(1,3);
 *flag = 1; // pass turn to child

 while(*flag != 0); // wait for my turn
 count(7,9);
 *flag = 1; // pass turn to child

 while(*flag != 0); // wait for my turn
 count(13,15);

 shmctl(shmid, IPC_RMID, NULL);

 return(0);
}
```

```
// 2.3: solution based on shared memory for both data
// exchange and synchronization; uses an array of 2
// integer cells in shared memory; the 1st cell is used
// to exchange a number; the 2nd cell is used to ensure
// alternate access to the 1st cell

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

main() {
 int num, shmid, *shmaddr;

 shmid=shmget(IPC_PRIVATE, 2*sizeof(int), 00600);
 shmaddr=(int*)shmat(shmid, NULL, 0);
 shmaddr[1]=0; // 0: child turn; 1: parent turn

 if (fork()==0) {
     do {
         scanf("%d", &num);
         shmaddr[0]=num;
         shmaddr[1]=1; // unblock parent
         while (shmaddr[1]!=0); // block child
         num=shmaddr[0];
         printf("%d\n", num);
     } while (num!=1);
     shmctl(shmid, IPC_RMID, 0);
     exit(0);
 }

 do {
     while(shmaddr[1]!=1); // block parent
     num=shmaddr[0];
     num=num+1;
     shmaddr[0]=num;
     shmaddr[1]=0; // unblock child
 } while (num!=1);

}
```

```
// 2.4: solution derived from 2.3; the 2nd cell in shared
// memory will have 3 possible values (0,1,2) that define
// which process is authorized to access the 1st cell

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

main() {
 int num, shmid, *shmaddr;

 shmid=shmget(IPC_PRIVATE, 2*sizeof(int), 00600);
 shmaddr=(int*)shmat(shmid, NULL, 0);
 shmaddr[1]=0; // 0: 1st child turn;
               // 1: 2nd child turn;
               // 2: parent turn

 if (fork()==0) { // 1st child
     do {
         scanf("%d", &num);
         shmaddr[0]=num;
         shmaddr[1]=1; // unblock 2nd child
         while (shmaddr[1]!=0); // block 1st child
         num=shmaddr[0];
         printf("%d\n", num);
     } while (num!=2);
     shmctl(shmid, IPC_RMID, 0);
     exit(0);
 }

 if (fork()==0) { // 2nd child
     do {
         while (shmaddr[1]!=1); // block 2nd child
         num=shmaddr[0];
         num=num+1;
         shmaddr[0]=num;
         shmaddr[1]=2; // unblock parent
     } while (num!=1);
     exit(0);
 }

 do { // parent
     while(shmaddr[1]!=2); // block parent
     num=shmaddr[0];
     num=num+1;
     shmaddr[0]=num;
     shmaddr[1]=0; // unblock 1st child
 } while (num!=2);

}
```

```
// 2.5.a: solution based on shared memory for both data
// exchange and synchronization; uses an array of 2
// integer cells in shared memory; the 1st cell is for
// the CHILD to pass its partial sum to the PARENT; the
// 2nd cell is 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;

 shmid=shmget(IPC_PRIVATE, 2*sizeof(int), 00600);
 shmaddr=(int*)shmat(shmid, NULL, 0);
 shmaddr[1]=0;

 if ( fork()==0 ) {
    shmaddr[0]=sum(51,100);
    shmaddr[1]=1; // unblocks parent
    exit(0);
 }

 sum_parent=sum(1,50);
 while (shmaddr[1]!=1); // blocks parent
 printf("%d\n", sum_parent + shmaddr[0]);
 shmctl(shmid, IPC_RMID, 0);
}
```

```
// 2.5.b: solution based on wait for synchronization

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

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

 if ( fork() == 0 ) { // Child
    shmaddr[0] = sum(51,100);
    exit(0);
 }

 sum_parent=sum(1,50);
 wait(NULL); // blocks parent
 printf("%d\n", sum_parent + shmaddr[0]);
 shmctl(shmid, IPC_RMID, 0);
}
```

```
// 2.6.a: solution based on shared memory for both data
// exchange and synchronization; uses an array of 10
// integer cells for the CHILDREN to pass the partial
// sums to the PARENT; uses a separate integer as a
// counter of the CHILDREN that have finished; this
// approach is INCORRECT (see // RACE CONDITION)

#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 shmid_counter, *ptr_counter;
 int i, sum_final=0;

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

 shmid_counter=shmget(IPC_PRIVATE, sizeof(int), 00600);
 ptr_counter=(int*)shmat(shmid_counter, NULL, 0);
 *ptr_counter=0;

 for (i=0; i<10; i++) {
     if (fork() == 0) { // i'th Child
          ptr_sums[i]=sum(i*10+1, i*10+10);
          (*ptr_counter)++;  // RACE CONDITION
          exit(0);
    }
 }

 // PARENT

 while (*ptr_counter != 10); // busy waiting

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

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

```
// 2.6.b: solution based on shared memory for both data
// exchange and synchronization; uses an array of 10
// integer cells for the CHILDREN to pass the partial
// sums to the PARENT; uses an array of 10 chars
// for the CHILDREN to synchronize with the PARENT

#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 shmid_flags;  char *ptr_flags;
 int i, sum_final=0;

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

 shmid_flags=shmget(IPC_PRIVATE, 10*sizeof(char), 00600);
 ptr_flags=(char*)shmat(shmid_flags, NULL, 0);
 for (i=0; i<10; i++) ptr_flags[i]=0;

 for (i=0; i<10; i++) {
     if (fork() == 0) { // i'th Child
          ptr_sums[i]=sum(i*10+1, i*10+10);
          ptr_flags[i]=1; // unblock parent
          exit(0);
    }
 }

 // PARENT

 for (i=0; i<10; i++) {
     while (ptr_flags[i] != 1); // block parent
     sum_final = sum_final + ptr_sums[i];
 }

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

```
// 2.6.c: solution based on shared memory for both data exchange
// and synchronization, but independent of the order in which
// CHILDREN finish; uses an array of 10 integer cells for the
// CHILDREN to pass the partial sums to the PARENT; uses an
// array of 10 chars for CHILDREN to synchronize with the
// PARENT, by recurring to an extra state (2 = collected)

#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 shmid_flags; char *ptr_flags;
 int i, sum_final=0; int j=0;

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

 shmid_flags=shmget(IPC_PRIVATE, 10*sizeof(char), 00600);
 ptr_flags=(char*)shmat(shmid_flags, NULL, 0);
 for (i=0; i<10; i++) ptr_flags[i]=0;

 for (i=0; i<10; i++) {
      if (fork() == 0) { // i'th Child
            ptr_sums[i]=sum(i*10+1, i*10+10);
            ptr_flags[i]=1;
            exit(0);
    }
 }

 // PARENT

 while (j != 10) {
      for (i=0; i<10; i++) {
            if (ptr_flags[i] == 1) { // result can be collected
                  sum_final = sum_final + ptr_sums[i];
                  ptr_flags[i] = 2;  // mark result as collected
                  j++;               // increment collected results
            }

      }
 }

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

```
// 2.6.d: solution based on wait for synchronization

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

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

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

 // PARENT

 while (wait(NULL)!=-1); // blocking waiting

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

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

```
// 2.6.e: solution based on wait for synchronization, but
// independent of the order in which CHILDREN finish; each
// child exits with an exit status equal to the index of the
// shared memory cell in which put its partial sum; for each
// cell index got from an exit status, the PARENT gets the
// partial sum stored in the shared memory cell with that index

#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <sys/ipc.h>
#include <sys/shm.h>
#include <sys/wait.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, status, j;

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

 for (i=0; i<10; i++) {
     if (fork() == 0) { // i'th CHILD
          ptr_sums[i]=sum(i*10+1, i*10+10);
          exit(i); // note the exit status ...
     }
 }

 // PARENT

 for (i=0; i<10; i++){
     wait(&status); // blocking waiting
     j = WEXITSTATUS(status);
     sum_final = sum_final + ptr_sums[j];
 }

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

```
// 2.7.a : we have basically a ring of 11 processes, with no
// no more than a process, at any time, changing the sum in
// shared memory; the flags that implement the ring also
// ensure that there are no race conditions accessing the sum

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

int main()
{
 int shmid_sum, *ptr_sum;
 int shmid_flags; char *ptr_flags;
 int i;

 shmid_sum=shmget(IPC_PRIVATE, sizeof(int), 00600);
 ptr_sum=(int*)shmat(shmid_sum, NULL, 0);
 *ptr_sum = 0;

 shmid_flags=shmget(IPC_PRIVATE, 11*sizeof(char), 00600);
 ptr_flags=(char*)shmat(shmid_flags, NULL, 0);
 for (i=0; i<11; i++) ptr_flags[i]=0;

 for (i=0; i<10; i++) {
     if (fork() == 0) { // i'th Child
          while (ptr_flags[i] != 1); // block this child
          *ptr_sum=*ptr_sum+(i*i);
          printf("%d %d\n", i, *ptr_sum);
          ptr_flags[i+1]=1; // unblock brother or parent
          exit(0);
    }
 }

 // PARENT

 ptr_flags[0] = 1; // unblock child 0
 while (ptr_flags[10] != 1); // block parent

 printf("%d\n", *ptr_sum);
 shmctl(shmid_sum, IPC_RMID, 0);
 shmctl(shmid_flags, IPC_RMID, 0);

 return(0);
}
```

```
// 2.7.b : solution with a single flag to control all processes

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

int main()
{
 int shmid_sum, *ptr_sum;
 int shmid_flag; char *ptr_flag;
 int i;

 shmid_sum=shmget(IPC_PRIVATE, sizeof(int), 00600);
 ptr_sum=(int*)shmat(shmid_sum, NULL, 0);
 *ptr_sum = 0;

 shmid_flag=shmget(IPC_PRIVATE, 1*sizeof(char), 00600);
 ptr_flag=(char*)shmat(shmid_flag, NULL, 0);
 *ptr_flag = -1;

 for (i=0; i<10; i++) {
     if (fork() == 0) { // i'th Child
           while (*ptr_flag != i); // block this child
           *ptr_sum=*ptr_sum+(i*i);
           printf("%d %d\n", i, *ptr_sum);
           (*ptr_flag)++; // unblock brother or parent
           exit(0);
    }
 }

 // PARENT

 *ptr_flag = 0; // unblock child 0
 while (*ptr_flag != 10); // block parent

 printf("%d\n", *ptr_sum);
 shmctl(shmid_sum, IPC_RMID, 0);
 shmctl(shmid_flag, IPC_RMID, 0);

 return(0);
}
```

```
// 2.8.a1 incorrect solution; has several race conditions

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

int main()
{
 int N, *A, i, ret, shmid_A, shmid_counter, *counter;

 scanf("%d", &N);

 shmid_A=shmget(IPC_PRIVATE, N*sizeof(int), 00600);
 A=(int*)shmat(shmid_A, NULL, 0);

 shmid_counter=shmget(IPC_PRIVATE, sizeof(int), 00600);
 counter=(int*)shmat(shmid_counter, NULL, 0);
 *counter=0;

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

 for (i=0; i<N; i++) {
     ret=fork();
     if (ret == 0) {
          if (A[i] % 2 == 0) {
               printf("A[%d]: %d : even\n", i, A[i]);
               exit(0);
          }
          else {
               printf("A[%d]: %d : odd\n", i, A[i]);
               (*counter)++;    // RACE-CONDITION 1
               exit(0);
          }
     }
 }

 while (*counter == 0);          // RACE-CONDITION 2
 //while (wait(NULL)!=-1);       // solution to RACE-CONDITION 2
 printf("odds: %d\n", *counter); // RACE-CONDITION 3

 shmctl(shmid_A, IPC_RMID, NULL);
 shmctl(shmid_counter, IPC_RMID, NULL);

 return(0);
}
```

```
// 2.8.a2 correct solution; has no race conditions

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

int main()
{
 int N, *A, i, ret, shmid_A, shmid_flags; char *flags;

 scanf("%d", &N);

 shmid_A=shmget(IPC_PRIVATE, N*sizeof(int), 00600);
 A=(int*)shmat(shmid_A, NULL, 0);

 shmid_flags=shmget(IPC_PRIVATE, N*sizeof(char), 00600);
 flags=(char*)shmat(shmid_flags, NULL, 0);

 srandom(getpid());
 for (i=0; i<N; i++) {
     A[i] = random();
     flags[i] = (char)-1; // mark each flag as not yet changed
 }

 for (i=0; i<N; i++) {
     ret=fork();
     if (ret == 0) {
          if (A[i] % 2 == 0) {
               printf("A[%d]: %d : even\n", i, A[i]);
               flags[i]=0; // change flag to even detected
               exit(0);
          }
          else {
               printf("A[%d]: %d : odd\n", i, A[i]);
               flags[i]=1; // change flag to odd detected
               exit(0);
          }
     }
 }

 int odds=0;
 for (i=0; i<N; i++) {
      while(flags[i] == -1); // busy waiting for flag to change
      odds += flags[i];
 }
 printf("odds: %d\n", odds);
 shmctl(shmid_A, IPC_RMID, NULL);
 shmctl(shmid_flags, IPC_RMID, NULL);

 return(0);
}
```

```
// 2.8.b

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

int main()
{
 int N, *A, i, ret, shmid_A, shmid_flags; char *flags;

 scanf("%d", &N);

 shmid_A=shmget(IPC_PRIVATE, N*sizeof(int), 00600);
 A=(int*)shmat(shmid_A, NULL, 0);

 shmid_flags=shmget(IPC_PRIVATE, N*sizeof(char), 00600);
 flags=(char*)shmat(shmid_flags, NULL, 0);

 for (i=0; i<N; i++)
      flags[i] = (char)-2; // mark each flag as standby

 for (i=0; i<N; i++) {
      ret=fork();
      if (ret == 0) {
            while(flags[i] != -1); // wait for unblock
            if (A[i] % 2 == 0) {
                  printf("A[%d]: %d : even\n", i, A[i]);
                  flags[i]=0; // change flag to even detected
                  exit(0);
            }
            else {
                  printf("A[%d]: %d : odd\n", i, A[i]);
                  flags[i]=1; // change flag to odd detected
                  exit(0);
            }
      }
 }

 srandom(getpid());
 for (i=0; i<N; i++) {
      A[i] = random();
      //flags[i] = (char)-1;  // mark each flag as unblock
 }

 for (i=0; i<N; i++)
      flags[i] = (char)-1; // mark each flag as unblock

 int odds=0;
 for (i=0; i<N; i++) {
       while(flags[i] == -1); // busy waiting for flag to change
       odds += flags[i];
 }
 printf("odds: %d\n", odds);
 shmctl(shmid_A, IPC_RMID, NULL);
 shmctl(shmid_flags, IPC_RMID, NULL);

 return(0);
}
```

// 2.9

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

int main()
{
 int pids[10], i, shmid_flags; char *flags;

 shmid_flags=shmget(IPC_PRIVATE, 10*sizeof(char), 00600);
 flags=(char*)shmat(shmid_flags, NULL, 0);

 for (i=0; i<10; i++)
      flags[i] = (char)-1; // mark each flag as blocked

 for (i=0; i<10; i++) {
      pids[i]=fork();
      if (pids[i] == 0) {
           while(flags[i] != 0); // wait for unblock
           printf("%d\n", getpid());
           flags[i] = 1; // signal printf done
           exit(0);
      }
 }

 for (i=0; i<10; i++) { // unblock children with even PID
      if (pids[i] % 2 == 0)
           flags[i] = (char)0;  // mark flag as unblock
 }

 for (i=0; i<10; i++) { // check children with even PID
      if (pids[i] % 2 == 0)
           while (flags[i] != 1);  // wait flag to be signaled
 }

 for (i=0; i<10; i++) { // unblock children with odd PID
      if (pids[i] % 2 != 0)
           flags[i] = (char)0;  // mark flag as unblock
 }

 for (i=0; i<10; i++) { // check children with odd PID
      if (pids[i] % 2 != 0)
           while (flags[i] != 1);  // wait flag to be signaled
 }

 printf("parent: done\n");
 shmctl(shmid_flags, IPC_RMID, NULL);

 return(0);
}
```