```
/* ----------------------------------------------------------- */
/* NAME : Bryan Wandrych                     User ID: bdwandry */
/* DUE DATE : 12/13/2021                                       */
/* PROGRAM ASSIGNMENT 6                                        */
/* FILE NAME : thread.h                                        */
/* PROGRAM PURPOSE : The purpose of this program is to         */
/* solve/generate n number of primes using the Sieves method.  */
/* This program will only use one global variable and          */
/* thread-based channels to complete the overarching program.  */
/* ----------------------------------------------------------- */
#include "ThreadClass.h"

//Global Definitions defined by header
extern int *Primes;
const int END_OF_DATA = -1;

/* ----------------------------------------------------------- */
/* FUNCTION: PrimeCheck (Class Definition)                     */
/*    This is the header file that describes the threadmentor’s*/
/*        class. This class will be shared with all files when     */
/*        threads are called for Prime Checking based threads.     */
/* PARAMETER USAGE :                                           */
/*    int index - The number to be passed along.               */
/*        int ThreadID - The identification number for Each Thread */
/*    SynOneToOneChannel *beforeChan - The previous Thread     */
/* FUNCTION CALLED :                                           */
/*    thread.cpp                                               */
/*    Threadmentor                                             */
/* ----------------------------------------------------------- */
class PrimeCheck : public Thread {
  public:
    PrimeCheck(int index, int ThreadID, SynOneToOneChannel *beforeChan);
        int PrintPrime(char * buffer);
        int CurrentThreadID();
  private:
    int Index;int neighbor;
        SynOneToOneChannel *beforeChan;
        SynOneToOneChannel *afterChan;
    PrimeCheck *primecheck;
    void ThreadFunc();
};

/* ----------------------------------------------------------- */
/* FUNCTION: MasterThread (Class Definition)                   */
/*    This is the header file that describes the threadmentor’s*/
/*        class. This class will be shared with all files when     */
/*        threads are called for Master based threads.             */
/* PARAMETER USAGE :                                           */
/*    int n - The upper bound to thread checking limit         */
/*        int ThreadID - The identification number for Each Thread */
/* FUNCTION CALLED :                                           */
/*    thread.cpp                                               */
/*    Threadmentor                                             */
/* ----------------------------------------------------------- */
class MasterThread : public Thread {
  public:
    MasterThread(int ThreadID, int n);
        int PrintMaster(char * buffer);
  private:
    int n; int TotalNumberOfPrimes;
        SynOneToOneChannel *beforeChan;
        SynOneToOneChannel *afterChan;
    PrimeCheck *primecheck;
    void ThreadFunc();
};
```

```
/* ----------------------------------------------------------- */
/* NAME : Bryan Wandrych                     User ID: bdwandry */
/* DUE DATE : 12/13/2021                                       */
/* PROGRAM ASSIGNMENT 6                                        */
/* FILE NAME : thread.cpp                                      */
/* PROGRAM PURPOSE : The purpose of this program is to         */
/* solve/generate n number of primes using the Sieves method.  */
/* This program will only use one global variable and          */
/* thread-based channels to complete the overarching program.  */
/* ----------------------------------------------------------- */
#include <iostream>
#include "thread.h"
#include <stdio.h>
#include <unistd.h>
#include <string.h>
#include <stdlib.h>
using namespace std;

/* ----------------------------------------------------------- */
/* FUNCTION: PrimeCheck                                        */
/*    This is the constructor to the PrimeCheck class          */
/*    described in the thread.h. It will supply definitions    */
/*    for pass through arguments passed from main.             */
/* PARAMETER USAGE :                                           */
/*    int index - The number to be passed along.               */
/*        int ThreadID - The identification number for Each Thread */
/*    SynOneToOneChannel *beforeChan - The previous Thread     */
/* FUNCTION CALLED:                                            */
/*    ThreadMentor                                             */
/* ----------------------------------------------------------- */
PrimeCheck::PrimeCheck(int index, int ThreadID, SynOneToOneChannel *beforeChan) :
beforeChan(beforeChan) {
        Index = index;
        neighbor = false;
        UserDefinedThreadID = ThreadID;
        afterChan = new SynOneToOneChannel("Prime", ThreadID, ThreadID + 1);
}

/* ----------------------------------------------------------- */
/* FUNCTION: PrintPrime                                        */
/*    This class is to simplify the printing experience for    */
/*    the PrimeCheck threads. It will be used a myriad of      */
/*    different times throughout a Prime Checking Threads.     */
/* PARAMETER USAGE :                                           */
/*    char * buf - This is a pass through buffer pointer       */
/*    that will be used in printing.                           */
/* FUNCTION CALLED:                                            */
/*    PrimeCheck(...)                                          */
/* ----------------------------------------------------------- */
int PrimeCheck::PrintPrime(char * buf) {
        char writeBuffer[10000];
        sprintf(writeBuffer, buf);
        write(1, writeBuffer, strlen(writeBuffer));
        memset(writeBuffer, 0, 10000);
        return 1;
}

/* ----------------------------------------------------------- */
/* FUNCTION: CurrentThreadID                                   */
/*    This function will be unique to every instantiated       */
/*    Thread. It will will return the ThreadID the thread      */
/*        that calls this function.                                */
/* PARAMETER USAGE :                                           */
/*    N/A                                                      */
/* FUNCTION CALLED:                                            */
/*    PrimeCheck(...)                                          */
/* ----------------------------------------------------------- */
int PrimeCheck::CurrentThreadID() {
        return Index;
}

/* ----------------------------------------------------------- */
/* FUNCTION: PrimeCheck::ThreadFunc                            */
/*    This will be the portion that will do the checking if    */
/*    if a number is going to be prime or not.                 */
/* PARAMETER USAGE :                                           */
/*    MasterThread::ThreadFunc() – A number passed from master */
/*    will be checked if it is prime or not                    */
/* FUNCTION CALLED :                                           */
/*        MasterThread::ThreadFunc() (Indirectly – Using Channels) */
/*    ThreadMentor                                             */
/*    Main                                                     */
/* ----------------------------------------------------------- */
void PrimeCheck::ThreadFunc() {
        Thread::ThreadFunc();
        char writeBuffer[10000];
        int CurrentThread = 0;
        sprintf(writeBuffer, "%*cP%d starts and memorizes %d\n", (Index + 1), ’ ’,
Index, Index);
        PrintPrime(writeBuffer);
        while (1) {
                beforeChan->Receive(&CurrentThread, sizeof(int));
                if (CurrentThread == END_OF_DATA) {
                        break;
                }
                sprintf(writeBuffer, "%*cP%d receives %d\n", (Index + 1), ’ ’, Index,
CurrentThread);
                PrintPrime(writeBuffer);

                if (CurrentThread % Index == 0) {
                        sprintf(writeBuffer, "%*cP%d ignores %d\n", (Index + 1), ’ ’,
Index, CurrentThread);
                        PrintPrime(writeBuffer);
                } else if (neighbor == 1){
                        sprintf(writeBuffer, "%*cP%d sends %d to P%d\n", (Index + 1), ’
’, Index, CurrentThread, primecheck->CurrentThreadID());
                        PrintPrime(writeBuffer);
                        afterChan->Send(&CurrentThread, sizeof(int));
                } else {
                        neighbor = 1;
                        sprintf(writeBuffer, "%*cP%d creates P%d\n", (Index + 1), ’ ’,
Index, CurrentThread);
                        PrintPrime(writeBuffer);
                        primecheck = new PrimeCheck(CurrentThread, UserDefinedThreadID +
1, afterChan);
                        primecheck->Begin();
                }
        }

        sprintf(writeBuffer, "%*cP%d receives END\n", (Index + 1), ’ ’, Index);
        PrintPrime(writeBuffer);
        if (neighbor == 1) {
                CurrentThread = END_OF_DATA;
                afterChan->Send(&CurrentThread, sizeof(int));
                primecheck->Join();
        }
        Primes[UserDefinedThreadID – 1] = Index;
        Exit();
}

//---------------------------------------------------------------------------------------
-----------------------------------------//
/* ----------------------------------------------------------- */
/* FUNCTION: MasterThread                                      */
/*    This is the constructor to the MasterThread class        */
/*    described in the thread.h. It will supply definitions    */
/*    for pass through arguments passed from main.             */
/* PARAMETER USAGE :                                           */
/*    int n - The upper bound to thread checking limit         */
/*        int ThreadID - The identification number for Each Thread */
/* FUNCTION CALLED:                                            */
/*    ThreadMentor                                             */
/* ----------------------------------------------------------- */
MasterThread::MasterThread(int ThreadID, int n) {
        TotalNumberOfPrimes = n;
        UserDefinedThreadID = ThreadID;
        afterChan = new SynOneToOneChannel("Master", UserDefinedThreadID,
UserDefinedThreadID + 1);
}

/* ----------------------------------------------------------- */
/* FUNCTION: PrintMaster                                       */
/*    This class is to simplify the printing experience for    */
/*    the PrimeCheck threads. It will be used a myriad of      */
/*    different times throughout a Master Threads.                 */
/* PARAMETER USAGE :                                           */
/*    char * buf - This is a pass through buffer pointer       */
/*    that will be used in printing.                           */
/* FUNCTION CALLED:                                            */
/*    MasterThread(...)                                        */
/* ----------------------------------------------------------- */
int MasterThread::PrintMaster(char * buf) {
        char writeBuffer[10000];
        sprintf(writeBuffer, buf);
        write(1, writeBuffer, strlen(writeBuffer));
        memset(writeBuffer, 0, 10000);
        return 1;
}

/* ----------------------------------------------------------- */
/* FUNCTION: MasterThread::ThreadFunc                          */
/*    This is a thread that will only be used to pass numbers  */
/*    3 to n to their respected PrimeCheck Threads.            */
/* PARAMETER USAGE :                                           */
/*    N/A                                                      */
/* FUNCTION CALLED :                                           */
/*    ThreadMentor                                             */
/*    Main                                                     */
/* ----------------------------------------------------------- */
void MasterThread::ThreadFunc() {
        Thread::ThreadFunc();
        char writeBuffer[10000];
        PrintMaster("Master starts\n");
        primecheck = new PrimeCheck(2, UserDefinedThreadID + 1, afterChan);
        primecheck->Begin();
        int input = 3;
        do {
                if (input - 1 == TotalNumberOfPrimes) {
                        PrintMaster("Master sends END\n");
                        input = END_OF_DATA;
                        afterChan->Send(&input, sizeof(int));
                        break;
                } else {
                        sprintf(writeBuffer, "Master sends %d to P2\n", input);
                        PrintMaster(writeBuffer);
                        afterChan->Send(&input, sizeof(int));
                }
                input++;
        } while (input != END_OF_DATA);

        primecheck->Join();
        PrintMaster("Master prints the complete result:\n");

        for (int i = 0; i < TotalNumberOfPrimes; i++) {
                if (i == 0) {
                        PrintMaster("  ");
                }
                if (Primes[i] == -1) {
                        PrintMaster("\n");
                        break;
                } else {
                        sprintf(writeBuffer, "%d ", Primes[i]);
                        PrintMaster(writeBuffer);
                }
        }

        PrintMaster("Master terminates\n");
        Exit();
}
```

```
/* ----------------------------------------------------------- */
/* NAME : Bryan Wandrych                      User ID: bdwandry */
/* DUE DATE : 12/13/2021                                       */
/* PROGRAM ASSIGNMENT 6                                        */
/* FILE NAME : thread-main.cpp                                 */
/* PROGRAM PURPOSE : The purpose of this program is to         */
/* solve/generate n number of primes using the Sieves method.  */
/* This program will only use one global variable and          */
/* thread-based channels to complete the overarching program.  */
/* ----------------------------------------------------------- */
#include <iostream>
#include "thread.h"
#include <stdio.h>
#include <unistd.h>
#include <string.h>
#include <stdlib.h>
using namespace std;

//Global Primes array initialization
int * Primes;

/* ----------------------------------------------------------- */
/* FUNCTION: SpawnThreads                                      */
/*    This is the function that will spawn the Master thread.  */
/*        In the Masterthread logic, it will spawn the PrimeCheck  */
/*        threads accordingly.
*/
/* PARAMETER USAGE :                                           */
/*    int n = Passed from Main that indicates the upper bound  */
/*        for prime numbers to be generated.                                      */
/* FUNCTION CALLED:                                            */
/*    Main                                                     */
/* ----------------------------------------------------------- */
int SpawnThreads(int n) {
        //Spawn Master Thread
        MasterThread *master = new MasterThread(0, n);
        master->Begin();
        master->Join();
        return 1;
}

/* ----------------------------------------------------------- */
/* FUNCTION: Main                                              */
/*    This is the starting point of the program. It will start */
/*    out by checking a couple of initial conditions and will  */
/*        instantiate n from the argument                          */
/* PARAMETER USAGE :                                           */
/*    Argv[1] = Passed from arguments from the command line    */
/*        that indicates the upper bound for prime numbers to be   */
/*        generated.                                                              */
/* FUNCTION CALLED:                                            */
/*    N/A                                                      */
/* ----------------------------------------------------------- */
int main(int argc, char *argv[]) {
        //This is the Initial Condition Section
        int n;
        if (argc == 1) {
                n = 30;
        } else if(argc > 2) {
                printf("ERROR: There is only aloud to have at max one argument.\n");
                return -1;
        } else if (atoi(argv[1]) < 3) {
                printf("ERROR: N must be greater than or equal to 3\n");
                return -1;
        } else {
                n = atoi(argv[1]);
        }

        Primes = (int *) malloc(sizeof(int) * n);

        for (int i = 0; i < n; i++) {
                Primes[i] = -1;
        }

        SpawnThreads(n);

        return 1;
}
```

```
CC        = c++
FLAGS     =
CFLAGS    = -g -O2 -Wno-write-strings -Wno-cpp -w
DFLAGS    = -DPACKAGE=\"threadsystem\" -DVERSION=\"1.0\" -DPTHREAD=1 -DUNIX_MSG_Q=1
-DSTDC_HEADERS=1
IFLAGS    = -I/local/eit-linux/apps/ThreadMentor/include
TMLIB     = /local/eit-linux/apps/ThreadMentor/Visual/libthreadclass.a
TMLIB_NV  = /local/eit-linux/apps/ThreadMentor/NoVisual/libthreadclass.a
OBJ_FILE = thread.o thread-main.o
EXE_FILE = prog6
${EXE_FILE}: ${OBJ_FILE}
                ${CC} ${FLAGS}  -o ${EXE_FILE}  ${OBJ_FILE} ${TMLIB} -lpthread

thread.o: thread.cpp
                ${CC} ${DFLAGS} ${IFLAGS} ${CFLAGS} -c thread.cpp

thread-main.o: thread-main.cpp
                ${CC} ${DFLAGS} ${IFLAGS} ${CFLAGS} -c thread-main.cpp

noVisual: ${OBJ_FILE}
                ${CC} ${FLAGS}  -o ${EXE_FILE}  ${OBJ_FILE} ${TMLIB_NV} -lpthread

clean:
                rm -f ${OBJ_FILE} ${EXE_FILE}
```

```
========================== COMPILATION ==========================
rm -f thread.o thread-main.o prog6
c++ -DPACKAGE=\"threadsystem\" -DVERSION=\"1.0\" -DPTHREAD=1 -DUNIX_MSG_Q=1
-DSTDC_HEADERS=1 -I/local/eit-linux/apps/ThreadMentor/include -g -O2 -Wno-write-strings
-Wno-cpp -w -c thread.cpp
c++ -DPACKAGE=\"threadsystem\" -DVERSION=\"1.0\" -DPTHREAD=1 -DUNIX_MSG_Q=1
-DSTDC_HEADERS=1 -I/local/eit-linux/apps/ThreadMentor/include -g -O2 -Wno-write-strings
-Wno-cpp -w -c thread-main.cpp
c++   -o prog6  thread.o thread-main.o
/local/eit-linux/apps/ThreadMentor/NoVisual/libthreadclass.a -lpthread
make: ’prog6’ is up to date.
Compilation done.
============================ TEST 1 ============================
Master starts
Master sends 3 to P2
   P2 starts and memorizes 2
   P2 receives 3
Master sends 4 to P2
   P2 creates P3
   P2 receives 4
Master sends 5 to P2
   P2 ignores 4
   P2 receives 5
Master sends 6 to P2
   P2 sends 5 to P3
    P3 starts and memorizes 3
    P3 receives 5
    P3 creates P5
   P2 receives 6
Master sends 7 to P2
   P2 ignores 6
   P2 receives 7
Master sends 8 to P2
   P2 sends 7 to P3
    P3 receives 7
    P3 sends 7 to P5
   P2 receives 8
Master sends 9 to P2
   P2 ignores 8
   P2 receives 9
Master sends 10 to P2
   P2 sends 9 to P3
      P5 starts and memorizes 5
      P5 receives 7
      P5 creates P7
   P2 receives 10
    P3 receives 9
Master sends 11 to P2
   P2 ignores 10
    P3 ignores 9
   P2 receives 11
Master sends 12 to P2
   P2 sends 11 to P3
    P3 receives 11
    P3 sends 11 to P5
   P2 receives 12
Master sends 13 to P2
   P2 ignores 12
   P2 receives 13
Master sends 14 to P2
   P2 sends 13 to P3
      P5 receives 11
      P5 sends 11 to P7
    P3 receives 13
    P3 sends 13 to P5
   P2 receives 14
Master sends 15 to P2
   P2 ignores 14
   P2 receives 15
Master sends 16 to P2
   P2 sends 15 to P3
        P7 starts and memorizes 7
        P7 receives 11
        P7 creates P11
      P5 receives 13
    P3 receives 15
      P5 sends 13 to P7
    P3 ignores 15
   P2 receives 16
Master sends 17 to P2
   P2 ignores 16
   P2 receives 17
Master sends 18 to P2
   P2 sends 17 to P3
    P3 receives 17
    P3 sends 17 to P5
   P2 receives 18
Master sends 19 to P2
   P2 ignores 18
   P2 receives 19
Master sends 20 to P2
   P2 sends 19 to P3
        P7 receives 13
        P7 sends 13 to P11
      P5 receives 17
      P5 sends 17 to P7
    P3 receives 19
    P3 sends 19 to P5
   P2 receives 20
Master sends 21 to P2
   P2 ignores 20
   P2 receives 21
Master sends 22 to P2
   P2 sends 21 to P3
            P11 starts and memorizes 11
            P11 receives 13
            P11 creates P13
        P7 receives 17
        P7 sends 17 to P11
      P5 receives 19
      P5 sends 19 to P7
    P3 receives 21
    P3 ignores 21
   P2 receives 22
Master sends 23 to P2
   P2 ignores 22
   P2 receives 23
Master sends 24 to P2
   P2 sends 23 to P3
    P3 receives 23
    P3 sends 23 to P5
   P2 receives 24
Master sends 25 to P2
   P2 ignores 24
   P2 receives 25
Master sends END
```

```
   P2 sends 25 to P3
            P11 receives 17
            P11 sends 17 to P13
        P7 receives 19
        P7 sends 19 to P11
      P5 receives 23
      P5 sends 23 to P7
    P3 receives 25
    P3 sends 25 to P5
   P2 receives END
              P13 starts and memorizes 13
              P13 receives 17
              P13 creates P17
            P11 receives 19
            P11 sends 19 to P13
        P7 receives 23
        P7 sends 23 to P11
      P5 receives 25
      P5 ignores 25
    P3 receives END
      P5 receives END
              P13 receives 19
              P13 sends 19 to P17
            P11 receives 23
            P11 sends 23 to P13
        P7 receives END
                  P17 starts and memorizes 17
                  P17 receives 19
                  P17 creates P19
              P13 receives 23
              P13 sends 23 to P17
            P11 receives END
                  P17 receives 23
                  P17 sends 23 to P19
                    P19 starts and memorizes 19
              P13 receives END
                    P19 receives 23
                    P19 creates P23
                  P17 receives END
                    P19 receives END
                        P23 starts and memorizes 23
                        P23 receives END
Master prints the complete result:
  2 3 5 7 11 13 17 19 23
Master terminates

============================ TEST 2 ============================
Master starts
Master sends 3 to P2
   P2 starts and memorizes 2
   P2 receives 3
   P2 creates P3
Master sends 4 to P2
   P2 receives 4
Master sends 5 to P2
   P2 ignores 4
   P2 receives 5
Master sends 6 to P2
   P2 sends 5 to P3
    P3 starts and memorizes 3
    P3 receives 5
    P3 creates P5
   P2 receives 6
Master sends 7 to P2
   P2 ignores 6
   P2 receives 7
   P2 sends 7 to P3
Master sends 8 to P2
    P3 receives 7
    P3 sends 7 to P5
   P2 receives 8
Master sends 9 to P2
   P2 ignores 8
   P2 receives 9
   P2 sends 9 to P3
Master sends 10 to P2
      P5 starts and memorizes 5
      P5 receives 7
      P5 creates P7
    P3 receives 9
    P3 ignores 9
   P2 receives 10
   P2 ignores 10
Master sends 11 to P2
   P2 receives 11
Master sends 12 to P2
   P2 sends 11 to P3
    P3 receives 11
   P2 receives 12
    P3 sends 11 to P5
Master sends 13 to P2
   P2 ignores 12
   P2 receives 13
   P2 sends 13 to P3
Master sends 14 to P2
      P5 receives 11
      P5 sends 11 to P7
    P3 receives 13
    P3 sends 13 to P5
   P2 receives 14
   P2 ignores 14
Master sends 15 to P2
   P2 receives 15
Master sends 16 to P2
   P2 sends 15 to P3
        P7 starts and memorizes 7
        P7 receives 11
        P7 creates P11
      P5 receives 13
      P5 sends 13 to P7
    P3 receives 15
    P3 ignores 15
   P2 receives 16
   P2 ignores 16
Master sends 17 to P2
        P7 receives 13
        P7 sends 13 to P11
   P2 receives 17
            P11 starts and memorizes 11
   P2 sends 17 to P3
Master sends 18 to P2
            P11 receives 13
            P11 creates P13
    P3 receives 17
   P2 receives 18
    P3 sends 17 to P5
Master sends 19 to P2
   P2 ignores 18
   P2 receives 19
Master sends 20 to P2
   P2 sends 19 to P3
      P5 receives 17
      P5 sends 17 to P7
    P3 receives 19
    P3 sends 19 to P5
   P2 receives 20
Master sends 21 to P2
   P2 ignores 20
        P7 receives 17
      P5 receives 19
   P2 receives 21
      P5 sends 19 to P7
        P7 sends 17 to P11
Master sends 22 to P2
   P2 sends 21 to P3
    P3 receives 21
   P2 receives 22
    P3 ignores 21
   P2 ignores 22
Master sends 23 to P2
   P2 receives 23
Master sends 24 to P2
   P2 sends 23 to P3
    P3 receives 23
    P3 sends 23 to P5
   P2 receives 24
Master sends 25 to P2
   P2 ignores 24
   P2 receives 25
   P2 sends 25 to P3
Master sends 26 to P2
            P11 receives 17
            P11 sends 17 to P13
        P7 receives 19
        P7 sends 19 to P11
      P5 receives 23
      P5 sends 23 to P7
    P3 receives 25
    P3 sends 25 to P5
   P2 receives 26
Master sends 27 to P2
   P2 ignores 26
   P2 receives 27
   P2 sends 27 to P3
Master sends 28 to P2
              P13 starts and memorizes 13
              P13 receives 17
              P13 creates P17
            P11 receives 19
            P11 sends 19 to P13
        P7 receives 23
        P7 sends 23 to P11
      P5 receives 25
      P5 ignores 25
    P3 receives 27
    P3 ignores 27
   P2 receives 28
Master sends 29 to P2
   P2 ignores 28
   P2 receives 29
   P2 sends 29 to P3
Master sends 30 to P2
```

```
    P3 receives 29
    P3 sends 29 to P5
   P2 receives 30
Master sends END
   P2 ignores 30
   P2 receives END
      P5 receives 29
      P5 sends 29 to P7
              P13 receives 19
    P3 receives END
              P13 sends 19 to P17
            P11 receives 23
            P11 sends 23 to P13
                  P17 starts and memorizes 17
        P7 receives 29
        P7 sends 29 to P11
      P5 receives END
                  P17 receives 19
                  P17 creates P19
              P13 receives 23
              P13 sends 23 to P17
            P11 receives 29
            P11 sends 29 to P13
        P7 receives END
                  P17 receives 23
                  P17 sends 23 to P19
              P13 receives 29
            P11 receives END
              P13 sends 29 to P17
                    P19 starts and memorizes 19
                    P19 receives 23
                    P19 creates P23
                  P17 receives 29
                  P17 sends 29 to P19
              P13 receives END
                    P19 receives 29
                    P19 sends 29 to P23
                  P17 receives END
                        P23 starts and memorizes 23
                        P23 receives 29
                        P23 creates P29
                    P19 receives END
                        P23 receives END
                              P29 starts and memorizes 29
                              P29 receives END
Master prints the complete result:
  2 3 5 7 11 13 17 19 23 29
Master terminates
```

Name: Bryan Wandrych
Username: bdwandry
M-Number: M17571110

1. The logic of your program

This program starts out by reading in a parameter from an argument that gets passed through via the command line. This passed through argument will represent the n.

This is the maximum/upper-limit of total number to be checked if they're prime or not. The initial conditions state that n must be greater than or equal to 3. If lower

The program will terminate with a print statement enclosed. After all of the initial conditions are passed, the program will then start out by spawning a Master Thread,

then passing along 2 to n numbers to a instantiated PrimeChecking Thread(s) by piping from master. This program works by using a Sieves paradigm, so it will only checked

if the previous threads that were declared as prime, are a multiple of the current prime being checked (i.e. if (current-prime % past-prime(s) != 0)). Once all of the numbers

are passed from 2 <= k <= n, then the program will pass along an ending sequence of -1 and the program will print the founded Primes array and gracefully terminate (all threads

included).

2. Why does your program work?

This program was constructed using "Linear Array Sorting Algorithm/Paradigm" defined on ThreadMentorsWiki. A lot of the concepts are used from this documented example.

Since this program is roughly architected based off of this design philosophy, the program is working in a similar approach to how each threads are threads are essientially

creating a linked list between each of the prior threads. This Primes checking algorithm takes it one step further by creating a doubly linked list between each of the threads,

in which all prior threads (that are deamed as primes) will be used if the current k thread (current thread being checked) is prime. If it is, another Prime Thread will be created

at the end, essientially adding on to the linked list (of threads).

3. The meaning, initial value and the use of each variable. Explain why their initial values and uses are correct. Justify your claim.

Global Variable(s):

extern int *Primes; - This is the storage for the global Primes Array that is instiated in the beggining of the program. It will n number of elements and will store every checked prime accordingly.

const int END_OF_DATA = -1; - This is global, but not modified at all throughout the program, all this variable does is passed along when the last number is checked to represent the end of checking.

(This variable is used in the Linear Array Sorting Algorithm).

PrimeCheck : public Thread

int index -The current element that is being checked.

int ThreadID -Used to keep track of the current Thread Identification Number, this will get incremented accordingly per number being passed by each thread.

int PrintPrime(char * buffer) -Simply used for printing.

int CurrentThreadID() -The purpose of this function is to return the current ThreadID. It will be used when printing sends, i.e. P2 sends 5 to P3.

int Index; -The passed through number from master.

int neighbor; -To see if there is a neighboring thread.

SynOneToOneChannel *beforeChan;-This keeps track of the previous threads that have a prime number.

SynOneToOneChannel *afterChan; -This keeps track of the next threads that have a prime number.

PrimeCheck *primecheck; -This will be the location of the next prime thread.

class MasterThread : public Thread

MasterThread(int ThreadID, int n); -Used to pass through relavent information from Main to start the MasterThread Thread.

int PrintMaster(char * buffer); -Simply used for printing.

int n; -This represend the total number of primes aloud to be checked.

int TotalNumberOfPrimes; -This represend the total number of primes aloud to be checked.

SynOneToOneChannel *beforeChan; -This keeps track of the previous threads that have a prime number.

SynOneToOneChannel *afterChan; -This keeps track of the next threads that have a prime number.

PrimeCheck *primecheck; -This will be the location of the next prime thread.

4. Answer the following:

(a) Can we use asynchronous channels to solve this problem? If you say "yes", prove it with a convincing argument. If you say "no", show a counter-example.

Yes, this program can be computed asynchronously. We would then think of master as placing all of the numbers into a buffer, in this case of this program,

it would be placed into p2. P2 would then start pushing out each recieved value in in the order their recieved, one-by-one. Then building out the list of threads

in order based on their received value.

(b)If the last thread in the chain receives a number larger than the one this thread memorized, then the incoming number must be a prime number.

Why is this true? Prove this fact with a convincing argument.

Given a number that is greater than the last memorized prime number. If the larger number is not prime, that means it would be a multiple of the past primes.

The last thread would only be reached if it has managed to make through all of the checkings of the previous primes. Which would indicate that the last larger

number would be prime.

(c)Explain how you can fill the array Primes elements in a consecutive way.

Throughout this program, each thread is incremented based off the last. So starting out the ThreadID (passed through master) at zero and then passing it along to

their respected Prime Checking Threads. Each Prime Checking Thread (afterwords) created is directly tied to their respeced ThreadID, meaning that each ThreadID being used is directly

connected to the current number being checked. The array will only be modified at the respected ThreadID -1 location. Which means that every location in the array will be modified

consecutively.

(d)You do not need a mutex to protect the global array Primes when a prime thread is saving its memorized prime number? Prove this with a convincing argument.

The Primes array does not need to be behind a mutex lock because each PrimeChecking thread being created is executing in a sequential fashion based on the number being passed/recieved.

These threads are acting in sequential fashion meaning, one thread will only must compute before another one can continue to "memorize" the next prime number.

5.You must terminate your program gracefully. More precisely, The last three output lines must be printed by Master.

```
        This program does terminate gracefully, all respected PrimeCheck threads are
placed with their respected protocols to make sure they will terminate.
        As well in master, the program will wait (Join()) for all of the PrimeCheck
thread created to terminate. Thus guarenting the last 3 lines in master will always
print last.
```

CS3331 Program 6 Grade Report

You receive 0 point if any one of the following occurs
No further grading will be done

Problem	*Check All Apply*	*You Receive*
Not-compile		0
Compile-but-not-run		0
Meaningless and/or vague Program		0
Did not implement the indicated methods (e.g., used semaphores, etc)		0
Did not follow the required program structure		0
Used non-channel primitives		0
Other significant deviation from specification, e.g., maximum parallelism		0
Totally wrong and unacceptable output		0

This part applies only if you have a working program

Item		*Max Possible*	*You Receuve*
Style & Doc.	Header in each file	2	2
	Good indentation	2	2
	Good comments	2	2
	Good use of function, variable names, etc & no `GOTO`	2	2
Spec	Handles input and argument list properly	2	2
	Correct output format	4	4
Correctness	Work on sample data	25	25
	Work on our data	25	25
`README`	Missing `README` – next two items receive 0	0	0
	Well-written `README`	3	3
	Answer questions properly	3	3
Deduction	Busy Waiting	-10	
	Race Conditions 10 points each	-10	
	Deadlocks 10 points each	-10	
Total		70	70

Your Score: 70