**任务描述**

本关任务：编写使用MPI计算斐波那契数列的C语言小程序，并按要求输出对应的斐波那契数列。

**相关知识**

为了完成本关任务，你需要掌握：

* 什么是MPI
* MPI常用函数

**什么是MPI**

说到并行计算，我们有一个不可绕开的话题——MPI编程。MPI是一个跨语言的通讯协议，用于编写并行计算机。支持点对点和广播。MPI是一个信息传递应用程序接口，包括协议和和语义说明，他们指明其如何在各种实现中发挥其特性。MPI的目标是高性能，大规模性，和可移植性。MPI在今天仍为高性能计算的主要模型。与OpenMP并行程序不同，MPI是一种基于信息传递的并行编程技术。消息传递接口是一种编程接口标准，而不是一种具体的编程语言。简而言之，MPI标准定义了一组具有可移植性的编程接口。

在C语言中我们可以通过包含mpi.h来获得MPI相关的函数声明与数据结构的定义。当用mpicc编译链接好程序后，我们可以使用mpirun来运行编译链接好的MPI程序。mpirun的使用方法很简单，只要设置想要并行进程的数量和想运行哪个MPI程序即可。例如：mpirun -np 4 ./helloMPI的意思就是会跑4个helloMPI程序。

**MPI常用函数**

MPI调用借口的总数虽然庞大，但常用的几个函数分别为MPI\_Init、MPI\_Comm\_size、MPI\_Comm\_rank、MPI\_Send、MPI\_Recv和MPI\_Finalize。

**MPI\_Init**

函数声明如下：

1. int MPI\_Init (int\* argc ,char\*\* argv[])

该函数通常应该是第一个被调用的MPI函数用于并行环境初始化，其后面的代码到MPI\_Finalize()函数之前的代码在mpirun跑的每个进程中都会被执行一次。除MPI\_Initialized()外， 其余所有的MPI函数应该在其后被调用。MPI系统将通过argc,argv得到命令行参数（也就是说main函数必须带参数，否则会出错）。

**MPI\_Finalize**

函数声明如下：

1. int MPI\_Finalize (void)

该函数用于退出MPI系统，所有进程正常退出都必须调用。表明并行代码的结束,结束除主进程外其它进程。串行代码仍可在主进程(即rank为0的进程)上运行，但不能再有MPI函数(包括MPI\_Init())。

**MPI\_Comm\_size**

函数声明如下：

1. int MPI\_Comm\_size (MPI\_Comm comm ,int\* size)

该函数用于获得并行的进程个数。并将个数返回给size。其中：

* comm: 指定一个通信子,也指定了一组共享该空间的进程, 这些进程组成该通信子的group。在调用该函数时，一般将MPI\_COMM\_WORLD作为参数传给comm。
* size: 获得通信子comm中规定的group包含的进程的数量。

**MPI\_Comm\_rank**

函数声明如下：

1. int MPI\_Comm\_rank (MPI\_Comm comm ,int\* rank)

该函数用于得到本进程在通信空间中的rank值，并将该值返回给rank。rank值即在group中的逻辑编号(该rank值为0到p-1间的整数，相当于进程的ID。其中：

* comm: 指定一个通信子,也指定了一组共享该空间的进程, 这些进程组成该通信子的group。在调用该函数时，一般将MPI\_COMM\_WORLD作为参数传给comm。
* rank: 获得本进程在通信子comm中规定的group中的逻辑编号。

**MPI\_Send**

函数声明如下：

1. int MPI\_Send( void \*buff, int count, MPI\_Datatype datatype, int dest, int tag, MPI\_Comm comm)

该函数用于对指定的进程以阻塞的方式发送数据。其中：

* buff：你要发送的数据。
* count：你发送的消息的个数(注意：不是长度，例如你要发送一个int整数，这里就填写1，如要是发送“hello”字符串，这里就填写6(C语言中字符串未有一个结束符，需要多一位))。
* datatype：你要发送的数据类型，这里需要用MPI定义的数据类型，如MPI\_INT。
* int dest：目的地进程号，你要发送给哪个进程，就填写目的进程的进程号，也就是rank值。
* tag：消息标签，接收方需要有相同的消息标签才能接收该消息。
* comm：通讯域。表示你要向哪个组发送消息。

**MPI\_Recv**

函数声明如下：

1. int MPI\_Recv( void \*buff, int count, MPI\_Datatype datatype, int source, int tag, MPI\_Comm comm, MPI\_Status \*status)

该函数用于对指定的进程以阻塞的方式接收数据。其中：

* buff：你接收到的消息要保存到哪个变量里。
* count：你接收的消息的个数(注意：不是长度，例如你要接收一个int整数，这里就填写1，如要是接收“hello”字符串，这里就填写6(C语言中字符串未有一个结束符，需要多一位))。它是接收数据长度的上界. 具体接收到的数据长度可通过调用MPI\_Get\_count函数得到。
* datatype：你要接收的数据类型，这里需要用MPI定义的数据类型，如MPI\_INT。
* source：接收端进程号，你要需要哪个进程接收消息就填写接收进程的进程号，也就是rank值。
* tag：消息标签，需要与发送方的tag值相同的消息标签才能接收该消息。
* comm：通讯域。表示你要接收哪个组发送过来的消息。
* status: 消息状态。接收函数返回时，将在这个参数指示的变量中存放实际接收消息的状态信息，包括消息的源进程标识，消息标签，包含的数据项个数等。

假如现在跑了4个进程，想要让0号进程接收1，2，3号进程平行发送过来的消息，代码如下：

1. #include <stdio.h>
2. #include <string.h>
3. #include "mpi.h"
4. void main(int argc, char\* argv[])
5. {
6. int numprocs, myid, source;
7. MPI\_Status status;
8. char message[100];
9. MPI\_Init(&argc, &argv);
10. MPI\_Comm\_rank(MPI\_COMM\_WORLD, &myid);
11. MPI\_Comm\_size(MPI\_COMM\_WORLD, &numprocs);
12. if (myid != 0) { //非0号进程发送消息
13. strcpy(message, "Hello World!");
14. MPI\_Send(message, strlen(message) + 1, MPI\_CHAR, 0, 99,
15. MPI\_COMM\_WORLD);
16. }
17. else { // myid == 0，即0号进程接收消息
18. for (source = 1; source < numprocs; source++) {
19. MPI\_Recv(message, 100, MPI\_CHAR, source, 99,
20. MPI\_COMM\_WORLD, &status);
21. printf("接收到第%d号进程发送的消息：%s\n", source, message);
22. }
23. }
24. MPI\_Finalize();
25. }

输出结果如下：

1. 接收到1号进程发送过来的消息：Hello World！
2. 接收到2号进程发送过来的消息：Hello World！
3. 接收到3号进程发送过来的消息：Hello World！

**编程要求**

根据提示，在右侧编辑器补充代码，使用MPI实现打印斐波那契额数列的功能。并且需要完成的输入与输出的格式如下：

* 输入: 需要输出的斐波那契额数列的长度，类型为int，如4
* 输出: 打印出斐波那契数列，如输入为4，输出为1 1 2 3

**PS：斐波那契数列中第n个值可以使用如下公式计算：**a\_n=\frac{1}{\sqrt 5}[(\frac{1+\sqrt 5}{2})^n - (\frac{1-\sqrt 5}{2})^n]*a*​*n*​​=​√​5​​​​​1​​[(​2​​1+√​5​​​​​)​*n*​​−(​2​​1−√​5​​​​​)​*n*​​]