```c
/*
 * client.c
 *
 * This file is the client program,
 * which prepares the arguments, calls "rpcCall", and checks the returns.
 */

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

#include "rpc.h"

#define CHAR_ARRAY_LENGTH 100

int main() {

  /* prepare the arguments for f0 */
  int a0 = 5;
  int b0 = 10;
  int count0 = 3;
  int return0;
  int argTypes0[count0 + 1];
  void **args0;

  argTypes0[0] = (1 << ARG_OUTPUT) | (ARG_INT << 16);
  argTypes0[1] = (1 << ARG_INPUT) | (ARG_INT << 16);
  argTypes0[2] = (1 << ARG_INPUT) | (ARG_INT << 16);
  argTypes0[3] = 0;

  args0 = (void **)malloc(count0 * sizeof(void *));
  args0[0] = (void *)&return0;
  args0[1] = (void *)&a0;
  args0[2] = (void *)&b0;

  /* prepare the arguments for f1 */
  char a1 = 'a';
  short b1 = 100;
  int c1 = 1000;
  long d1 = 10000;
  int count1 = 5;
  long return1;
  int argTypes1[count1 + 1];
  void **args1;

  argTypes1[0] = (1 << ARG_OUTPUT) | (ARG_LONG << 16);
  argTypes1[1] = (1 << ARG_INPUT) | (ARG_CHAR << 16);
  argTypes1[2] = (1 << ARG_INPUT) | (ARG_SHORT << 16);
  argTypes1[3] = (1 << ARG_INPUT) | (ARG_INT << 16);
  argTypes1[4] = (1 << ARG_INPUT) | (ARG_LONG << 16);
  argTypes1[5] = 0;

  args1 = (void **)malloc(count1 * sizeof(void *));
  args1[0] = (void *)&return1;
  args1[1] = (void *)&a1;
  args1[2] = (void *)&b1;
  args1[3] = (void *)&c1;
  args1[4] = (void *)&d1;

  /* prepare the arguments for f2 */
  float a2 = 3.14159;
  double b2 = 1234.1001;
  int count2 = 3;
  char *return2 = (char *)malloc(CHAR_ARRAY_LENGTH * sizeof(char));
```

```
65  int argTypes2[count2 + 1];
66  void **args2;
67
68  argTypes2[0] = (1 << ARG_OUTPUT) | (ARG_CHAR << 16) | CHAR_ARRAY_LENGTH;
69  argTypes2[1] = (1 << ARG_INPUT) | (ARG_FLOAT << 16);
70  argTypes2[2] = (1 << ARG_INPUT) | (ARG_DOUBLE << 16);
71  argTypes2[3] = 0;
72
73  args2 = (void **)malloc(count2 * sizeof(void *));
74  args2[0] = (void *)return2;
75  args2[1] = (void *)&a2;
76  args2[2] = (void *)&b2;
77
78  /* prepare the arguments for f3 */
79  long a3[11] = {11, 109, 107, 105, 103, 101, 102, 104, 106, 108, 110};
80  int count3 = 1;
81  int argTypes3[count3 + 1];
82  void **args3;
83
84  argTypes3[0] = (1 << ARG_OUTPUT) | (1 << ARG_INPUT) | (ARG_LONG << 16) | 11;
85  argTypes3[1] = 0;
86
87  args3 = (void **)malloc(count3 * sizeof(void *));
88  args3[0] = (void *)a3;
89
90  /* prepare the arguemtns for f4 */
91  char *a4 = "non_exist_file_to_be_printed";
92  int count4 = 1;
93  int argTypes4[count4 + 1];
94  void **args4;
95
96  argTypes4[0] = (1 << ARG_INPUT) | (ARG_CHAR << 16) | strlen(a4);
97  argTypes4[1] = 0;
98
99  args4 = (void **)malloc(count4 * sizeof(void *));
100 args4[0] = (void *)a4;
101
102 /* rpcCalls */
103 int s0 = rpcCall("f0", argTypes0, args0);
104 /* test the return f0 */
105 printf("\nEXPECTED return of f0 is: %d\n", a0 + b0);
106 if (s0 >= 0) {
107   printf("ACTUAL return of f0 is: %d\n", *((int *)(args0[0])));
108 }
109 else {
110   printf("Error: %d\n", s0);
111 }
112
113
114 int s1 = rpcCall("f1", argTypes1, args1);
115 /* test the return of f1 */
116 printf("\nEXPECTED return of f1 is: %ld\n", a1 + b1 * c1 - d1);
117 if (s1 >= 0) {
118   printf("ACTUAL return of f1 is: %ld\n", *((long *)(args1[0])));
119 }
120 else {
121   printf("Error: %d\n", s1);
122 }
123
124
125 int s2 = rpcCall("f2", argTypes2, args2);
126 /* test the return of f2 */
127 printf("\nEXPECTED return of f2 is: 31234\n");
128 if (s2 >= 0) {
129   printf("ACTUAL return of f2 is: %s\n", (char *)args2[0]);
```

```
130   }
131   else {
132     printf("Error: %d\n", s2);
133   }
134
135
136   int s3 = rpcCall("f3", argTypes3, args3);
137   /* test the return of f3 */
138   printf(
139     "\nEXPECTED return of f3 is: 110 109 108 107 106 105 104 103 102 101 11\n"
140   );
141
142   if (s3 >= 0) {
143     printf("ACTUAL return of f3 is: ");
144     int i;
145     for (i = 0; i < 11; i++) {
146       printf(" %ld", *(((long *)args3[0]) + i));
147     }
148     printf("\n");
149   }
150   else {
151     printf("Error: %d\n", s3);
152   }
153
154   int s4 = rpcCall("f4", argTypes4, args4);
155   /* test the return of f4 */
156   printf("\ncalling f4 to print an non existed file on the server");
157   printf("\nEXPECTED return of f4: some integer other than 0");
158   printf("\nACTUAL return of f4: %d\n", s4);
159
160
161   /* rpcCalls , function name not exist */
162   int s9 = rpcCall("f9", argTypes0, args0);
163   printf("\nEXPECTED return of f9 is: function not exist.\n");
164   if (s9 >= 0) {
165     printf("ACTUAL return of f9 is: %d %d\n", s9,*((int *)(args0[0])));
166   }
167   else {
168     printf("Error: %d\n", s9);
169   }
170
171   /* rpcTerminate */
172   printf("\ndo you want to terminate? y/n: ");
173   if (getchar() == 'y')
174     rpcTerminate();
175
176   /* end of client.c */
177   return 0;
178 }
```