Suppose that you know how to do something in one programming language, but not the other. Then the following "Rosetta table" is for you! You can access an electronic version of this table (which facilitates searching) by pointing your web browser to

"http://me.unm.edu/~rmbrann/gobag.html"

Table 1: Rosetta table for FORTRAN and C++

FORTRAN	C++
`      PROGRAM MYPRGM` `C  This program prints "hello world"` `C  and then stops` `      print*,'hello world'` `      stop(0)` `      end`	`#include <iostream.h>` `/*  This program prints "hello world"` `    and then stops */` `int main()` `{` `    cout << "hello world\n";` `    return (0);` `}`
	`signed int k;` `signed float x;`
`      DOUBLE PRECISION X`	`double float x;`
`      EXTERNAL FNT`	`extern fnt;`
`      PARAMETER (PI=3.14159)`	`const float PI = 3.14159;`
`      X=Y ! inline comment (F90)`	`x=y; // inline comment`
`.LT.` `.LE.` `.GT.` `.GE.` `.EQ.` `.NE.` `.AND.` `.OR.`	`<` `<=` `>` `>=` `==` `!=` `&&` `||`
`      IF(I.LT.10.AND.JAR(I).NE.0)` `     $ JAR(I)=2` *!The "$" in column 5 is just a continuation character to fit everything in this table.*	`if(i<10&&(jar[i]!=0)` `  jar[i]=2;` *// Note: the expressions are evaluated left to right. Hence if i equals or exceeds 10, the second expression (with its invalid access to to jar[i]) will not be evaluated.*
`INTEGER A,B` `EQUIVALENCE(A,B)`	`int B;` `int &A=B;`
`     INTEGER A,B,K` `CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC` `      K=MOD(A,B)`	`int a,b,k;` `/*****************/` `k=a%b;`
`      DIMENSION U(2,3)`	*// No analog. C++ arrays start at 0, not 1.*

Table 1: Rosetta table for FORTRAN and C++

FORTRAN	C++
`DIMENSION U(0:1,0:2)` *! This is how you can dimension a fortran array to behave like a c/c++ array. This syntax is standard (but rarely used) fortran 77.*	`float u[2][3];`
`DIMENSION Z(3),P(8),U(2,3)` `DATA Z/1,-2,3/,P/1,2,3,5(0)/` `DATA U/11,21,12,22,13,23/` *!In FORTRAN, the first index increments before the next. C++ orders data in the opposite way.* `!...` `print*,'last element= ',u(2,3)`	`float z[]={1,-2,3};`*//C++ dimensions z to number of values given, in this case, 3.* `float p[8]={1,2,3};`*//C++ dimensions p by explicitly giving 8; all unspecified values are initialized to zero.* `float u[][3]={{11,12,13},` `{21,22,23}};` *//Note that c++ automatically determines that the first dimension must equal 2.* `//...` `cout << "last element= " << u[1][2];` *//C++ starts numbering at 0 rather than 1; thus the ij component of an array is at [i-1][j-1].*
`integer m(8,9),idum(9)` *!...a copy is needed to send a matrix ROW* `do 100 i=1,9` `100 idum(i)=m(5,i)` `call myfnt(dum)` *!...no copy needed to send a matrix COLUMN* `call fnt2(m(1,3)) ! 3rd col of m`	`int m[8][9]; int dum[9];` *//...no copy needed to send a matrix ROW* `myfnt(m[4]);` *!m[4] is the 5th row of m.* *//...copy needed to send a matrix COLUMN* `for(i=0;i<9;++i)dum[i]=m[i][3];` `fnt2(dum);`
`include 'mystuff'`	`#include "mystuff"` */* Includes permit relative or absolute paths. */*
There are no intrinsic libraries provided with with every FORTRAN compiler. If special functions do exist, they are usually accessed via compile command line options. On the other hand, there are numerous intrisic functions that are defined without having to name a library.	`#include <nameOFstandardHeaderFile>` */* Note: standard headers are normally located in /usr/include.*/*
`char(0) ! NULL` `char(8) ! backspace` `char(12) ! formfeed` `char(10) ! newline` `char(13) ! return` `char(9) ! tab` `char(39) ! single tic (')` `char(34) ! double quote (")` `char(82) ! capital R`	`'\0'` *//note: single tics surround single chars* `'\b'` *//Double quotes are reserved for strings.* `'\f'` `'\n'` `'\r'` `'\t'` `'\''` `'"'` `'\122' (octal) or '\52' (hexadecimal)`
`INT('A') !gives ascii code`	`int('A') // gives ascii code`

Table 1: Rosetta table for FORTRAN and C++

FORTRAN	C++
`write(*,*)'The value is ',val` `print*,'Hello world'` *! print*, is the same as write(*,*)*	`#include <iostream.h>` `cout << "The value is " << val;` `cout << "Hello world" << endl;` *// using endl (instead of \n) will both add a new line and flush the print buffer.*
`read(*,*)val`	`#include <iostream.h>` `cin >> val;`
c Read 2 values separated by COMMAS or SPACES `read(*,*)val1,val2`	*// Read 2 values separated by SPACE(S)* `cin >> val1 >> val2;`
`character word*8,phrase*60` `!...` `word='Hello'` `c------------------------` `ldum=lnblnk(word)` *! result is 5* *! The function lnblnk is intrinsic in F90, but not F77.* `c------------------------` `phrase=word(1:ldum)//" world"` `c------------------------` `j=LEN(phrase)` *! result is 60* `c------------------------` `if(phrase.eq.'Hello world')then` `j=0` `else` `j=1` `end if` `if(word.eq.'done')return` `c------------------------` `print*,'Please enter a string'` `read(*,*)phrase` `c------------------------` `print*,'Enter a word'` `read(*,*)word` `c------------------------` `print*,phrase(2:2)` *! 2nd character*	`#include <string.h>` `char word[9]; char phrase[61];` *//set the dimensions one larger than FORTRAN counterpart because the last character in a C++ string must be the ASCII null character ('\0').* `//...` `strcpy(word,"Hello");` `/*----------------------*/` `ldum=strlen(word);` `/*----------------------*/` `strcpy(phrase,word);` `strcat(phrase," world");` `/*----------------------*/` `j=sizeof(phrase);` `/*----------------------*/` `if(!strcmp(phrase,"Hello world"){` `j=0;` `}else{` `j=1;` `}` `if(strcmp(word,'done')==0)return(0);` `/*----------------------*/` `cout << "Please enter a string\n";` `cin.getline(phrase,sizeof(phrase));` *//use the getline form when the string being read might contain whitespace* `/*----------------------*/` `cout << "Enter a word"` `cin >> word;` `/*----------------------*/` `cout << phrase[1]` *// 2nd character*
`CHARACTER*6 DUM` `DATA DUM/'bargle'/`	`char dum[]="bargle"`
`x=x+1` `x=x-1`	`++x;` `--x;`

Table 1: Rosetta table for FORTRAN and C++

FORTRAN	C++
x=x+2 **x=x-2** **x=x*2** **x=x/2** **x=MOD(x,2)**	**x+=2;** **x-=2;** **x*=2;** **x/=2;** **x%=2;**
INTEGER SIZE,RESULT,VALUEX DATA SIZE/5/ **SIZE=SIZE+1** **RESULT=SIZE** **VALUEX=SIZE** **SIZE=SIZE+1**	int size=5,result,valuex; **result=++size**; *//obfuscatory programming.* **valuex=size++**; *//obfuscatory programming.*
NUMBER=5 RESULT=NUMBER NUMBER=NUMBER+1 RESULT=NUMBER NUMBER=NUMBER+1	number=5; result=number++ ; *//obfuscatory. Hard to tell whether result gets value 5 or 6.* */* Here is the better way to do the same thing */* result=number; ++number; *// When written on a single line, there is no functional difference between ++var and var++. The form ++var is slightly more efficient.*
RESULT=VAL*5+3 VAL=VAL+1	result=(val++*5)+3; *//obfuscatory*
X=02137 *! Leading zeros are ignored in FORTRAN. This number is equivalent to 2137.*	x=2137; *//leading zeros must not be used for base-10 numbers (C++ will interpret them as octal).*
!There is no FORTRAN means of getting the dimensioned number of elements in an array. You track it with parameter statements.	float farr[5];int iarr[7];char carr[22]; lenOFfloatArray=sizeof(farr)/ sizeof(float); *//gives 5.* lenOFintArray=sizeof(iarr)/ sizeof(int); *//gives 7* lenOFstring=sizeof(carr) *//division by sizeof(char) is not necessary because sizeof(char) is 1.*
	#include <iostream.h> //How to get # elmts in an array main() { int arr[5]; cout << "No. elements in arr = " << sizeof(arr)/sizeof(int) << endl; return (0); }

Table 1: Rosetta table for FORTRAN and C++

FORTRAN	C++
`      LOGICAL DEJAVU` `      DATA DEJAVU/.FALSE./` `      SAVE DEJAVU` `!...` `      IF(.NOT.DEJAVU)THEN` `        PRINT*,'welcome'` `        DEJAVU=.TRUE.` `      END IF` `      IF(X.GT.Y)THEN` `       PRINT*,'X exceeds Y'` `      ELSE IF(X.EQ.Y.or.X.lt.-Y)THEN` `       PRINT*,'X=Y or is less than -Y'` `      ELSE IF(.NOT.DEJAVU)` `       PRINT*,'Replacing X val again'` `       X=Y` `      ELSE` `       PRINT*,'Replacing X by Y'` `       X=Y` `      END IF`	`#include <iostream>` `bool dejavu=0;` *// Type "bool" is not supported by all c++ compilers; it is part of the NEW standard.* `static dejavu;` `//...` `if(!dejavu){` `  cout << "welcome" << endl;` `  dejavu=0;` `}` `if(x>y)` `  cout << "X exceeds Y" << endl;` `else if((x==y)||(x<-y))` `  cout << "X=Y or is less than -Y";` `else if(!dejavu){` `  print*,'Replacing X val again';` `  x=y;}` `else {` `  cout << "Replacing X by Y";` `  x=y;}`
`      IF(I.LT.J)GO TO 73` `      ...` `   73 CONTINUE` *! statement label must be integer and lie in columns 1 through 5.* `      ...`	`if(i<j)goto myjumpspot` `   ...` `myjumpspot:` *// gotos are almost never the most natural way to branch control elsewhere.* `   ...`
`      ITOTAL=0` `C---------------------------------` `C...Infinite loop` `   10 CONTINUE` `        PRINT*,'Enter number to add',` `     $       ' or 0 to stop'` `         READ(*,*)ITEM` `         IF(ITEM.EQ.0)GO TO 20` `         IF(ITEM.LT.0)GO TO 10` `         ITOTAL=ITOTAL+ITEM` `         PRINT*,'subtotal: ',ITOTAL` `      GO TO 10` `C---------------------------------` `C` *Label 20 is the break location* `   20 PRINT*,"Final total=',ITOTAL`	`int total=0` `//---------------------------------` `//...Infinite loop` `while(1){` `     cout<<"Enter number to add"` `         <<" or 0 to stop"<<endl;` `     cin>>item;` `     if(item==0)break;` `     if(item<0)continue;` `     total+=item;` `     cout<<"subtotal: "<<total<<endl;` `        }` `C---------------------------------` `//` *the "break" automatically takes us here* `cout<<"Final total="<<total`

Table 1: Rosetta table for FORTRAN and C++

FORTRAN	C++
	The following code fragment illustrates how to use bool on systems that don't support it and it also illustrates the use of the identifier "static". `#ifdef COMPILER_SUPPORTS_BOOL` `typedef bool Bool` `#else` `typedef int Bool` `const int false=0;` `const int true=1;` `#endif` `static int my_limited_var;` `int my_global_var;` `static int myfunct1(){...}` `float myfunct2(arg1,arg2){` `  static bool dejavu=false` `  ...` `  }` `main(){...}` */* Since they are defined outside of any function definitions, both my_limited_var and my_global_var are "visible" to all functions in the file. However, my_global_var is also visible to functions in other files while the "static" identifier limits the scope of my_limited_var to only the file in which it is declared. Likewise, the funtion myfunct2 is callable from any file whereas myfunct1 is callable only from this file in which it is declared. The "static" identifer for the variable "dejavu" declared within the function myfunct2 makes dejavu exist even after the function is done, but its value is visible only from within myfunct2.*
`      INTEGER ITEMS` `C...` `      PRINT*,'enter number of items'` `      READ(*,*,ERR=98)ITEMS` `C...` `   98 PRINT*,'BAD INPUT!'` `      RETURN(1)`	`int items` `//...` `cout<<"enter number of items"` `cin>>items;` `if(!cin.good()){` `  cout<<"BAD INPUT!"<<endl;` `  return(1)}`

Table 1: Rosetta table for FORTRAN and C++

FORTRAN	C++
`IF(X.EQ.'A')THEN` `  ! do A stuff` `ELSE IF(X.EQ.'B'` `$  .OR.X.EQ.'C')THEN` `  IF(X.EQ.B)THEN` `  ! do B-only stuff` `  END IF` `  ! do B and C stuff` `ELSE IF(X.EQ.'D')THEN` `  ! do D stuff` `ELSE` `  ! all else` `ENDIF` *! The c++ "switch" is like old-fashioned FORTRAN computed gotos with better formatting.*	`switch (x)` *// Beware: x must be char or int.* `{` `  case 'a':` `  /* do 'a' stuff */` `  break;` `  case 'b':` `  /* do 'b'-only stuff */` `  case 'c':` `  /* do 'b' and 'c' stuff */` `  break;` `  case 'd':` `  /* do 'd' stuff */` `  break;` `  default:` `  /* all else */` `}` *// Beware: if you misspell the word "default", the compiler will think the misspelled word is a goto label, and your code will run wrong even though it compiles just fine.*
`C print a line of '@' signs` `  CHARACTER*10 JNKSTR` `  INTEGER X` `  READ(*,*)X` `C Create a format statement` `C For example, if x is 23, create` `C a string ... (23('@'))` `C Then use that string as the format` `  WRITE(JNKSTR,*)'(',X,'(''@''))'` `  WRITE(*,JNKSTR)` `C alternative solution _` `C  |` `C  v`	`/* print a line of @ signs */` `int x;` `cin >> x;` `while (x--){` `  cout<< '@';` `}` */* Note:* *if x>0, prints x @'s* *if x==0, does nothing except decrements x* *if x<0, infinite loop!* *Placement of "while" controls when the conditional test is performed (before or after body of loop) compare with the do-while form below.*
`C ANOTHER way to` `C print a line of '@' signs` `  CHARACTER*100 TMPCHR` `  INTEGER X` `  READ(*,*)X` `  TMPCHR=' '` `  DO 10 I=1,X` `10 TMPCHR(I:I)='@'` `  WRITE(*,*)TMPCHR`	`/* another way to print a line of @ signs */` `int x;` `cin >> x;` `do {` `  cout << '@';` `}while (x--);` */* Note:* *if x>=0, prints x+1 @'s* *if x<0, infinite loop!*

Table 1: Rosetta table for FORTRAN and C++

FORTRAN	C++
`      CHARACTER  REPLY*4` `      DO 216 I=3,50,8` `        PRINT*,'I=',I` `        PRINT*,'Wanna see I+8?'` `        READ(*,*)REPLY` `C       strings are read like any other variable.` `        IF(REPLY.NE.'yes')GO TO 77` `        PRINT*,I+8` `        J=I/4` `        IF(J.GT.1.AND.J.LT.3)GOTO 216` `       PRINT*,'J =',J` `  216  CONTINUE` `   77 CONTINUE`	`#include <strings.h>` `char reply[5];` `for (i=3;i<51;i+=8){` `   cout<<"i="<<i<<endl;` `   cout<<"Wanna see i+8?"<<endl;` `   //strings are read with cin.getline` `   cin.getline(reply,sizeof(reply);` `   if(strcmp(reply,"yes"))break;` `   cout<<i+8;` `   j=i/4;` `   if(j>1 && j<3)continue;` `   cout<<"j =",j;` `}`
`   65 CONTINUE` `        ! body of infinite loop` `        READ(*,*)NUM` `        IF(NUM.EQ.-1)GO TO 3` `        IF(2*NUM.LE.6)GOTO 65` `        PRINT*,"NUM EXCEEDS 6"` `      GO TO 65` `    3 CONTINUE` `C  Alternatively, in FORTRAN90,` `      do` `       !body of infinite loop` `      enddo`	`for (;;){` `   // body of infinite loop` `   cin>>num` `   if(num==-1)break;` `   if(2*num<=6)continue;` `   cout<<"num exceeds 6"<<endl;` `}` `/* alternatively */` `while (1){` `   // body of infinite loop` `}`
`      J=1` `      DO 23 I=0,99` `      J=J+3` `      PRINT*,I,J` `   23 CONTINUE` `   99 CONTINUE`	*// comma operator (invented to allow us to do multiple operations in the "for" expression)* `for (i=0,j=1; i<100; i++,j+=3){` `  cout<<i,j<<endl;` `}`
`      INTEGER KOUNT`	`auto int kount;` *// the "auto" in the above declaration is added for clarity. Temporary local variables will always override global variables of the same name. If you know that's what you are doing, you might emphasize the point by using the "auto" qualifier.*

Table 1: Rosetta table for FORTRAN and C++

FORTRAN	C++
`PROGRAM TESTF` `EXTERNAL MYFNT`	`#include <iostream.h>` `/* function prototypes */` `int myfunction(float);` *//extern by default.* `void myroutine(const float x,float y, float z);` *// names of arguments are ignored in prototypes. You can include them for clarity. Protype can look EXACTLY like the function defintion line.*
`COMMON MYCOM/U,V,R,S/` *! For FORTRAN, these global variables may be accessed in any routine that includes MYCOM.*	`float u,v,r,s` *// declaration outside "main" makes these global (accessible everywhere).*
`READ(*,*)U,V,R,S` `PRINT*,MYFNT(U)` `T=3.0` `V=R+S` `Q=R**S` `CALL MYROUT(T,V,Q)` `STOP(0)` `END`	`int main` `{` `  cin >> u >> v >> r >> s;` `  cout<<myfunction(u);` `  float t=3.0;` `  float v=r+s;` `  q=r^s;` `  myrout (t,v,q);` `  return(0);` `}`
`INTEGER FUNCTION MYFNT(X)` `U=2.0` `MYFNT=X**2+U` `RETURN` `END`	`int myfunction(float x){` `  int u=2.0;` *//local declaration overrides global* `  return(x^2+u);` `}`
Fortran always calls by reference. To call by value, a fortran calling routine would need to explicitly make a copy of a variable and then pass the copy.	*/* Below, "void" means no return value (like a FORTRAN subroutine). "const" says value of x will not change, the ampersands tell the compiler to pass-by-reference. Without them, x and y would be sent "by value" (meaning copies of u and v would be sent and upon return to main, u and v would still have their original values.). Arrays are always passed by reference even withou the ampersand.*/*
`SUBROUTINE MYROUT(X,Y,Z)` `COMMON MYCOM/UU,DUM1,RR,DUM3/` *! Note: contents of "MYCOM" may have different names in different routines. Variables are positional.* `DATA NUMC/0/` `SAVE NUMC` `NUMC=NUMC+1` `IF(X.GT.0.0)THEN` `  Y=Y+Z` `ELSE` `  Y=3.0` `  Z=4.0` `END IF` `IF(Z.GT.UU)PRINT*,Z,UU,RR,NUMC` `RETURN` `END`	`void myrout(const float x, float &y, float &z)` `{` `  float &uu=u;` *// uu is another name for u* `  static int numcalls=0;` `  ++numcalls;` `  if(x>0.0)y=y+z;` `  else {y=3.0;z=4.0;}` `  if(z>uu) cout<<z<<uu<<r<<numcalls;` `  return;` `}`

Table 1: Rosetta table for FORTRAN and C++

FORTRAN	C++
COMMON DUM/VAR1,KLS/ *! put common in any routine that uses VAR1 and KLS.* *! initialize VAR1 and KLS in the first routine where dum will be accessed (many codes write special initialization routines that are guaranteed to be called first at run time).* **SUBROUTINE INIT** **COMMON DUM/VAR1,KLS/** **VAR1=0.0** **KLS=3** **RETURN** **END**	*/* First choose one single file where you will actually create the storage for the global variable. Define and initialize near the top of that file (not within function bodies)... */* **float var1=0.0;** **float kls=3;** */* If the variable is to be used within routines whose source code resides in other files, declare the existence of the variable by using "extern". The actual definition (above) will be found at link time.*/* extern float var;
INTEGER FUNCTION GETVAL	int getval(void){...} *alternatively,* int getval() {...}
SUBROUTINE MYPROC	void my_procedure() {...}
	/ This function returns a POINTER to the largest element in array. Thus, the calling routine may use it on either the right or left side of the equals sign.*/* int **&**biggest(const int **&**array[], const int n_elements){ int indx; *// current indx* int big=0; *// indx to bigest element* for (indx=1;indx<n_elements;++indx) {if(array[biggest]<array[indx]) big=indx;} return (**array[big]**); } *Note: the ampersand for the array argument is not strictly necessary, but it is added for clarity to emphasize that array[big] is not a local variable — it must exist after the call in order for us to pass a reference to it!* ... *Sample usage ...* *This replaces the largest element by 999.* biggest(item_array,5)=999
! FORTRAN77 does not support function overloading. *!The integer and real functions must* *!have different names.* INTEGER FUNCTION ISQ(j) ISQ=J*J RETURN END REAL FUNCTION RSQ(X) RSQ=X*X RETURN END	*/* C++ supports function overloading. Two different functions with the same name are distinguished by their argument type. */* int sq(int j) { return(j*j); } float sq(float x) { return(x*x); }

Table 1: Rosetta table for FORTRAN and C++

FORTRAN	C++
DOUBLE PRECISION TEMP DOUBLE PRECISION TRAJCT, BALVEL DIMENSION TRAJCT(3),BALVEL(3)	typedef double Real8; typedef Real8 Vector[3]; typedef Vector Velocity; Real8 temperature; Vector trajectory; Velocity Ball_vel;
	enum day_of_week{SUNDAY, MONDAY, ..., SATURDAY}; enum day_of_week today=MONDAY; char day_names[7][9]={"Sunday","Monday",..,"Saturday"}; cout<<"Today is "<<day_names[int(today)]<<endl;
	Use enumerated types to define integer parameters within classes (compiler will not accept const int). class Queue { public: enum {MAX_QUEUE_LENGTH = 200}; private: int qLength; int qBegin; int data[MAX_QUEUE_LENGTH]; public: // Create the Queue inline Queue::Queue(void); // Put an item in the Queue void put(const int item); // Get an item from the Queue int get(void); // Print the Queue void list(void); // Destroy the Queue ~Queue(); };

Table 1: Rosetta table for FORTRAN and C++

FORTRAN	C++
	The basic components of a class... The following example is for a "stack" which is a collection of data that is stored one-at-a-time using "push" and retrieved on a last-in-first-out basis using "pop." The name of the calss is "stack" the function "stack" automatically gets called whenever a stack object is created. `const int STACK_SIZE=100;` `class stack{` `/*Member variables*/` `  private:` `   int count; //num items in stack` `   int data[STACK_SIZE]; //items` `/*Member functions*/` `  public:` `   //Initialize the stack` `   stack (void);` `   //Push an item onto the stack` `   void push(const int item);` `   //Pop an item from the stack` `   int pop(void);` */* A friend of a class is either an object that has access to private data and/or private functions. (Friends are not actual members of the class). Below the function stack_equal compares both the public and private data to see if two stacks are equal.*/* `   friend stack_equal(const stack &s1, const stack &s2);` `};` `inline void stack::push(const int item){` `  data[count]=item;` `  ++count;` `}` `inline int stack::pop(void){` `  //Stack goes down by one` `  --count;` `}`

Table 1: Rosetta table for FORTRAN and C++

FORTRAN	C++
	The constructor and destructor functions for a class... Note: constructors must never return a value, so the compiler knows that this constructor is of type "void". Compiler will complain if you nevertheless put it in. `inline stack::stack(void){` `  count=0; //Zero the stack` `}` `stack::~stack(void){` `  if(count != 0)` `  cerr<<"Error: stack not empty\n";` `{`
	Using a class... Note: The constructor stack.stack() is called automatically when the stack is created. `class stack mystack;` `mystack.push(12);` `mystack.push(47);` `cout<<mystack.pop();  //prints 47` `cout<<mystack.pop();  //prints 12`
	Constructors may be overloaded. Below we define a class for a person's name and phone number. `class person{` `  private:` `  char name[80];` `  char phone[80];` `  public:` `  person(const char pnam[],pnum[]);` `  // ... rest of class` `};` `  person::person(const char pnam[],pnum[]){` `  strcpy(name,pnam);` `  strcpy(phone,pnum);` `}` `  person::person(const char pnam[]){` `  strcpy(name,pnam);` `  strcpy(phone,"no phone");` `}` `//------- use the class` `main()` `{ person rebecca("Rebecca Brannon","555-1234");`

Table 1: Rosetta table for FORTRAN and C++

FORTRAN	C++
	Copy constructor. (used to make an exact copy of a class). The copy constructor has the same name as the class itself and it takes a class object as its sole argument) `stack::stack(const stack &old_stack){` `  for (int i=0; i<old_stack.count;` `  ++i){data[i]=old_stack.data[i];` `}` `count=old_stack.count;` `}` *If you do nor write a copy constructor, then the compiler will attempt to make one for you, but it might make errors. To avoid problems, if you do not want to explicitly define a copy constructor, you should declare the copy constructor to be "private" so there will be a compiler error if someone tries to access it. Example:* `class nocopy{` `  // Body of the class` `  private:` `  nocopy(const nocopy &old_class);` `};`
	Friends are not restricted to just functions. One class can be a friend to another. For example: `class item {` `  private:` `  int data;` `  friend class set_of_items;` `};` `class set_of_items {...}` *In this case, since the class* `set_of_items` *is a friend of* `item`*, it has access to all the members of item.*

Table 1: Rosetta table for FORTRAN and C++

FORTRAN	C++
	class employee { `private:` `  char last_name[40];` `  char first_name[40];` `  int ssn;` `  int dob;` `  float pay_rate;` `public:` `  void chng_pay(float new_rate);` `  void print_employee_name() const;` `  // ...` `}` *You can define a class object that is always constant.* `class employee fred` *The following places a copy of "fred" into "employee_of_the_week" (using the copy constructor)* `const employee employee_o_week(fred);` *Member data may be modified for fred, but not for employee_of_week. In general, you will not be able to call any of the "employee" member functions unless their prototypes in the class definition end in "const". Thus, you cannot change the pay rate of the employee_of_week, but you can print the name.*
	For example, when the user asks for last-name-first formatting, then they generally want <u>all</u> stacks will be formatted like that from that point forward. Use the "static" modifier to identify class members (both data and functions) that are to be identical (global) among <u>all</u> existing objects of that class. Here's how: `class employee {` `  private:` `  static int last_name_1st_form=0;` `  /*...other member data that varies from employee to employee */` `  public:` `  static void set_prefs();` `  //...` `}`
`INCLUDE "../myfile"` *! includes permissibly contain any valid FORTRAN statements, both declarations and/or source code.*	`#include "../myfile"` *// includes contain declarations only — no source code.*

Table 1: Rosetta table for FORTRAN and C++

FORTRAN	C++
! These are C preproccessor directives, but you can use them in a fortran file as well. HELPFUL HINTS: `CC -E file.cc` *sends output of the preprocessor to the standard output.* `CC -DDEBUG file.cc` *Defines the variable "DEBUG" even if it is not defined directly in the coding — very convenient since you don't have to change each individual file (just change the compile options in your makefile).* `CC -DMAX=10 file.cc` *is the general form for defining preprocessor variables.*	`#define A 32*B` *// C preprocessor directive replaces all occurances of A with 32*B. This is texual substituion, and there are no compiler sanity checks. It will replace the expression int A =4; by 32*B=4, which will result in a mysterious compiler error.* `#undef A` `#define mysign(u) (u)/abs(u)` *The above will, for example, change mysign(f(x)) to f(x)/ABS(f(x)). This is an inelegant way to get macros. With c++, you should use inline functions for safer and better performance. Alternaitively, if the definition is only for a parameter, (e.g., #define A 22), use the c++ const directive (const A=22;).* *The nice thing about using the preprocessor is that the parameterized macro (mysign) will work for both integers and reals, whereas we would otherwise have to write two inline functions. Disadvantage: you should enclose the argument (u) in parens everywhere it appears. Leaving paren off first "u" will give wrong answer for mysign(2+2) cuz it will compute 2+2/abs(2+2), which gives 2+2/4, or just 2.* `#if !defined(INCLUDED_ABC)` `#define INCLUDED_ABC` `...` `#endif`
	`#define DIVISIBLE_BY_TEN(x) !((x)%10)` `#define SWAP(X,Y) {register int hold=x;x=y;y=hold}`
	`#define BOOL int` `#define TRUE (1)` `#define FALSE (0)`
	// This is good for feature control, *// poor for configuration management.* `#if HPUX || SUNOS` `...` `#elif UNICOS` `...` `#else` `...` `#endif`

Table 1: Rosetta table for FORTRAN and C++

FORTRAN	C++
	`#ifdef DEBUG` `  cout << "Debugging on"` `#elseif PARALLEL` `  cout << "Parallel version on"` `#else` `  cout << "standard version"` `#endif /*DEBUG*/` `#ifndef`
	!To avoid nesting problems, set up your header files like this... `#ifndef _CONST_H_INCLUDED` `/* Define constants */` `#define _CONST_H_INCLUDED` `#endif`
`C The only way to comment out a large` `C section of code is to put "C" in` `C 1st column of all lines.` `C  EXTERNAL SREPT` `CC Handle end of section stuff` `C  CALL DMPTBL`	*/* In C, the slash-star sytax may be used in large comments, but it is a poor way to comment out a piece of coding that itself contains slash-star comments. The better way is to use the preprocessor like this:*/* `#ifdef UNDEFINED` `  section_report();` `  /* Handle end of section stuff */` `  dump_table();` `#endif /* UNDEFINED */`
`  CHARACTER STRNG*6` `  X=3.14159` `  Z=8.76E14` `  K=34` `  STRNG="Here is a long string"` `c Note that STRNG is truncated` `c to ...  "Here i"` `  PRINT5,X,K,Z,STRNG` `  5 FORMAT('x=',f9.4,' i=',I2,` `  $ 4X, 'Z:',G12.2,12(/` `),A3,2x,'bye ya''ll')` *the above print statement results in* `x=  3.141 i=34  Z:0.87e15` *(12 cariage returns; thus 11 blank lines)* `Her  bye ya'll`	`#include <string.h>` `char strng[6]; char buf[80];` `float x=3.14159;` `float z=8.76e14;` `int k=34;` `strcpy(strng,'Here i');` `sprintf(buf,"x=%9f i=%2d  z:%12.2g` *<12 carriage returns -- I don't know how to do this yet>* `%3s  bye ya'll",x,k,z,strng);` `cout<<buf<<endl;`

Table 1: Rosetta table for FORTRAN and C++

FORTRAN	C++
`      SUBROUTINE MYROUT` `!  A function call is not allowed on the left side of an` `!  ordinary FORTRAN assignment, so the compiler` `!  interpretes the following as the definition of a` `!  statement function.` `!  (Must occur before first executable).` `      SQUARE(V)=V*V` `C...` `      X=SQUARE(X)` `      RETURN` `      END`	`...` `void myroutine(){` `//...` `x=square(x);` `//...` `}` `inline int square(int value) {` `  return (value*value)` `}`
`      FUNCTION MYFNT(ARRAY)` `      DIMENSION ARRAY(*)`	*// Here's how you pass an array* `my_function(int array[])` *// C++ automatically turns arrays into reference arguments. You can emphasize this point by using an ampersand even though you don't have to.*
	// Here's how you pass a pointer `function(int *var)`
`      FUNCTION FCTRL(NUM)` `      N=NUM` `      NUMFAC=1` `    7 IF(N.EQ.0)GO TO 88` `      NUMFAC=NUMFAC*N` `      N=N-1` `      GO TO 7` `   88 FCTRL=NUMFAC` `      RETURN` `      END`	*// Here's a recursive function that computes the factorial of n by directly using the formula n!=n(n-1)!* `int factorial(int num)` `{` `  if (num==0)return(1);` `  /*else*/` `  return (num*factorial(num-1));` `}`
	// Bitwise exclusive OR operator `^` *// Bitwise complement operator* `~`
`C while (K.GT.0) do loop` `      K=-1` `   23 IF(.NOT.(K.GT.0)) GO TO 33` `        K=K-3` `        GO TO 23` `   33 CONTINUE` `      PRINT*,K ! prints -1`	 `k=-1` `while(k>0){` `  k-=3` `}` `cout<<k //prints -1`
`C Repeat...UNTIL (K.LE.0) structure` `      K=-1` `   23 CONTINUE` `        K=K-3` `        IF(.NOT.(K.LE.0))GO TO 23` `      PRINT*,K ! prints -4`	*//do...until (condition) is the same as do while !(condition).* `k=-1` `do{` `  k-=3;` `}while(!(k<=0));` `cout<<k  // prints -4`

Table 1: Rosetta table for FORTRAN and C++

FORTRAN	C++
`INTEGER I` `REAL R` `DOUBLE PRECISION D` `COMPLEX X` `CHARACTER*1 C` `CHARACTER*80 S` `LOGICAL L`	`int i;` `float r;` `double d;` `???? x;` `char c;` `char s[80];` `bool L;`
! Algebraic and misc functions *! Here and in the table cells that follow, the variable IRDX may be integer, real, double, or complex, the variable RD may be real or double, etc.* `ABS(IRDX)` `EXP(RDX)` `LOG(RDX)` `LOG10(RDX)` `MAX(IRD1,IRD2,...)` `MIN(IRD1,IRD2,...)` `MOD(IRD1,IRD2)` `SIGN(IRD1,IRD2)` *!transfer sign of IRD2 to IRD1* `SQRT(irdx)`	 `<math.h> abs(i); fabs(rd)` `exp` `log` `log10` `max` `min` `ird1%ird2` `?` `sqrt(irdx)`
! Integer functions `INT(IRDX)` *!integer truncation toward zero* `NINT(IRDX)` *!nearest integer=INT(r+0.5) if r.ge.0* *! or INT(r-0.5) if r.lt.0.*	
! Trigonometric functions `ACOS(IRDX)` `ASIN(RD)` `ATAN(RD)` `ATAN2(RDy,RDx)` *!RDy is the y-component of the point, and RDx is the x component of the point.* `COS(RDX)` `SIN(RDX)` `TAN(RDX)`	
! Hyperbolic functions `COSH(RD)` `SINH(RD)` `TANH(RD)`	
! Complex functions `AIMAG(X)` *!imaginary part of X* `CMPLX(IRD)` *!gives cmpx number (REAL(IRD),0)* `CMPLX(IRD1,IRD2)` *!(REAL(IRD1),REAL(IRD2))* `CONJG(X)` *! complex conjugate*	

Table 1: Rosetta table for FORTRAN and C++

FORTRAN	C++
! String and character functions `INDEX(S1,S2)` *! If character string S2 is contained within string S1, this operation gives the position of the first character of S2 in S1. Example: INDEX("mytest string","test") gives a value of 3.* `LEN(S)`	`<strings.h> strstr(s1,s2)` `sizeof(s)`
! Casting functions `REAL(IRDX)` *!single precision real part* `DBLE(IRDX)` `ICHAR(C)` *!integer equivalent of character* `CHAR(I)` *!ASCII character for integer I.*	`float(irdx)` `dble`
	/ Define bit constants. In binary, bits are numbered 76543210 by convention. Thus, for example, the constant represented by bit 5 is 00100000. An easy way to define this constant is 1<<5. Thus, you might see coding like this:*/* `//True if any error is set` `const int ERR=(1<<0)  // bit 0` `//Framing error flage` `const int FRAME_ERR=(1<<1)  //bit 1` `...` *etc.* */* Here is how you set a bit */* `char flags=0;  // Start all flags 0` `...` `flags |= FRAME_ERR // frame error` */*(keep in mind the above is a short-hand way of writing flags=flags|FRAME_ERR.* */* Here is how you test a bit */* `if((flags&ERROR)!=0)` `  cerr<<"Error flag is set\n";` */* Here is how you clear a bit */* `flags &= ~FRAME_ERR //clear frame err`
	`int i=15  // decimal (base 10)` `int j=017  // octal (base 8)` `int k= 0xF // hexidecimal (base 16)`
	`~i  // complement of integer` `i&j  // Bitwise AND` `i|j  // Bitwise OR` `i^j  // Exclusive OR` `i<<n` *// Shift left (same as multiplying by 2^n)* `i>>n` *// Shift right (same as dividing by 2^n)*

Table 1: Rosetta table for FORTRAN and C++

FORTRAN	C++
	`(~i)+1` *// The negative of a signed integer.*
	`cout << "Result of " << hex << c1 << " & " << c2 << " = " << (c1&c2) << dec << endl;` *Note: the "hex" in the cout stream makes all subsequent integers be output in hexidecimal form. The output format is restored to decimal with the "dec" parameter. Neither hex nor dec result in output per se.*
	// the following function determines if an iteger is even. `inline int even(const int value){` `return((value&1)==0);` `}`

Table 1: Rosetta table for FORTRAN and C++

FORTRAN

```
! This program reads  input having one of two forms.
! The first form is  KEYWORD=VALUE, which is for
! assigning values. The other form is just  KEYWORD
! for executing tasks such as "help" or "stop"

      PROGRAM MAIN



      PARAMETER (IBSIZE=80)
      PARAMETER (KWLEN=10)
      CHARACTER LINE*IBSIZE,KW*KWLEN


C     infinite loop over statement 77
   77 CONTINUE

      PRINT*,"Enter input:"
      LINE=' '
      READ(*,*)LINE


      IEQ=INDEX(LINE,'=')
      IF(IEQ.GT.0)THEN
        READ(LINE(1:IEQ-1),*)KW
        READ(LINE(IEQ+1:IBSIZE,*)VAL
        PRINT*,
     $  'you entered ',KW,'=',VAL
      ELSE
        KW=LINE
        IF(KW.EQ.'help')THEN
         CALL HELP
        ELSEIF(KW.EQ.'stop')THEN
         GO TO 88
        ELSE
         PRINT*,'unknown: ',KW
         GO TO 77
        ENDIF
      ENDIF
      GO TO 77  ! end of infinte loop
   88 PRINT*,'Thank you. Bye!'
      STOP
      END
```

C++

```
#include <iostream.h>
#include <strstream.h>
#include <string.h>
#include <strings.h>

main(){
 Int i;
 void demo();

 const int BUFSIZE=80;
 const int KWLEN=10;
 char line[BUFSIZE];
 char kw[KWLEN];
 float val;

 while (1)   /* infinite loop */
 {
    cout<<"Enter input: "<<flush;

    cin.getline(line,BUFSIZE);
    istrstream is(line);


    if(strstr(line,"=")>0){
        is.getline(kw,KWLEN,'=');
        is>>val;
        cout <<"you entered: "
             <<kw<<'='<<val;
    }else{
        is>>kw;
        if(!strcmp(kw,"help"))
                        {help();}
        else if(!strcmp(kw,"stop"))
                        {break;}
        else
       { cout<<"unknown: "<<kw<<endl;
          continue;
        }
    }
 }  /* end of infinite loop */
 cout<<"Thank you. Bye!"

}  /* end of main */
```

COMPILE TIME ERROR MESSAGES

Table 2: C++ error messages

C++ compiler message	Possible cause and resolution
`"hh.cc", line 17: Error: Badly formed expression.`	`missing semicolon`
`"max.cc", line 11: Error: Badly formed expression.`	*A preprocessing directive might be bad. Consider, for example, accidentally putting an "=" sign:* `#define MAX=10` `...` `for (k = MAX; k>0,k--)...` *Runing CC -E program.cc will send the result to the standard output:* `for (k = =MAX; k>0,k--)...` *The problem here is that the preprocessor syntax for defining MAX was not supposed to have an equal sign.*
`stack overflow`	`Too many local (auto) variables. Change large arrays to type global.`
`"hh.cc", line 92: Error: Unexpected ")" -- Check for matching parenthesis.`	`Arguments to a "for" loop wrongly delimited by commas instead of semicolons.`