```
#########################################################################
#  This program tests a general sorting routine (bsort, bubblesort)   #
#  with various arrays: int, float, double and string                 #
#                                  Peter Stallinga, UAlg, May 2025    #
#########################################################################
.eqv false 0
.eqv true 1
.macro push (%a)
  addiu $sp, $sp, -4
  sw %a, 0($sp)
.end_macro
.macro pop (%a)
  lw %a, 0($sp)
  addiu $sp, $sp, 4
.end_macro

.data

myintsarray: .word 8 7 32 8 6 11 9
myfloatsarray: .float 7.0 8.0 4.0 -13.1 67.3 9.7 4.1 -1.4
.align 3
mydoublesarray: .space 64
space: .asciiz " "
newline: .asciiz "\n"
s0: .asciiz "Ajax"
s1: .asciiz "Feyenoord"
s2: .asciiz "Porto"
s3: .asciiz "Benfica"
mytext: .asciiz "-------------\n"
x1: .asciiz "["
x2: .asciiz "]"
x3: .asciiz "comparing strings:\n"
x4: .asciiz "="
.align 2
mystringsarray: .space 16  # 4 x sizeof(char *)

.text

main:
 # build strings array:
   la $a0, mystringsarray
   la $t0, s0
   sw $t0, ($a0)
   addi $a0, $a0, 4   # size of char *
   la $t0, s1
   sw $t0, ($a0)
   addi $a0, $a0, 4   # size of char *
   la $t0, s2
   sw $t0, ($a0)
   addi $a0, $a0, 4   # size of char *
   la $t0, s3
   sw $t0, ($a0)
 # FloatArray2DoubleArray(myfloatsarray, mydoublesarray, 8) :
   la $a0, myfloatsarray
   la $a1, mydoublesarray
   la $a2, 8
   jal FloatArray2DoubleArray
 # printIntArray(myintsarray[], 7)  :
   la $a0, myintsarray
   li $a1, 7
   jal printIntArray
 # bsort(myintsarray[], 7, sizeof(int), myCompareInts) :
   la $a0, myintsarray
   li $a1, 7
   li $a2, 4
   la $a3, myCompareInts
   jal bsort
 # printIntArray(myintsarray[], 7) :
   la $a0, myintsarray
   li $a1, 7
   jal printIntArray
 # printFloatArray(myfloatsarray[], 8) :
   la $a0, myfloatsarray
   li $a1, 8
   jal printFloatArray
 # bsort(myfloatsarray[], 8, sizeof(float), myCompareFloats) :
   la $a0, myfloatsarray
   li $a1, 8
```

```
   li $a2, 4
   la $a3, myCompareFloats
   jal bsort
 # printFloatArray(myfloatsarray[], 8) :
   la $a0, myfloatsarray
   li $a1, 8
   jal printFloatArray
 # printDoubleArray(myfloatsarray[], 8) :
   la $a0, mydoublesarray
   li $a1, 8
   jal printDoubleArray
 # bsort(mydoublesarray[], 8, sizeof(double), myCompareDoubles) :
   la $a0, mydoublesarray
   li $a1, 8
   li $a2, 8
   la $a3, myCompareDoubles
   jal bsort
 # printDoubleArray(myfloatsarray[], 8) :
   la $a0, mydoublesarray
   li $a1, 8
   jal printDoubleArray
 # printStringArray(mystringsarray[], 4)  :
   la $a0, mystringsarray
   li $a1, 4
   jal printStringArray
 # bsort(mystringsarray[], 3, sizeof(char *), myCompareStrings) :
   la $a0, mystringsarray
   li $a1, 4
   li $a2, 4
   la $a3, myCompareStrings
   jal bsort
 # printStringArray(mystringsarray[], 4)  :
   la $a0, mystringsarray
   li $a1, 4
   jal printStringArray
 # return :
return:
   li $v0, 10
   syscall

############### FUNCTIONS ######################

#  int myCompareInts(const void *a, const void *b)
myCompareInts:
####################################
#  args:                           #
#    a0, a1: addresses of ints     #
#  return:                         #
#    v0 = (a1)-(a0)                #
####################################
  lw $t0, ($a0)
  lw $t1, ($a1)
  sub $v0, $t1, $t0
  jr $ra

#  int myCompareFloats(const void *a, const void *b)
myCompareFloats:
####################################
#  args:                           #
#    a0, a1: addresses of floats   #
#  return:                         #
#    v0 = -1, 0, +1                #
####################################
  lwc1 $f0, ($a0)
  lwc1 $f1, ($a1)
  c.lt.s $f1, $f0
  bc1t cf_setneg
  c.lt.s $f0, $f1
  bc1t cf_setpos
  move $v0, $zero
  j cf_exit
cf_setneg:
  li $v0, -1
  j cf_exit
cf_setpos:
  li $v0, 1
cf_exit:
  jr $ra
```

```
#  int myCompareDoubles(const void *a, const void *b)
myCompareDoubles:
###################################
#  args:                          #
#    a0, a1: addresses of doubles #
#  return:                        #
#    v0 = -1, 0, +1               #
###################################
  ldc1 $f0, ($a0)
  ldc1 $f2, ($a1)
  c.lt.d $f2, $f0
  bc1t cd_setneg
  c.lt.d $f0, $f2
  bc1t cd_setpos
  move $v0, $zero
  j cd_exit
cd_setneg:
  li $v0, -1
  j cd_exit
cd_setpos:
  li $v0, 1
cd_exit:
  jr $ra

#  int myCompareStrings(const void *a, const void *b)
myCompareStrings:
###################################
#  args:                          #
#    a0, a1: addresses of address #
#                    of strings   #
#  return:                        #
#    v0 = strcmp(a0, a1)          #
###################################
  lw $t0, ($a0)   # char *t0 = * (char **) a0
  lw $t1, ($a1)   # char *t1 = * (char **) a1
cs_start_dowhile:
  lb $t2, ($t0)
  lb $t3, ($t1)
  sub $v0, $t3, $t2
  beqz $t2, cs_exit
  beqz $t3, cs_exit
  addi $t0, $t0, 1
  addi $t1, $t1, 1
  beq $t2, $t3, cs_start_dowhile
cs_exit:
  jr $ra

#  void FloatArray2DoubleArray(float *fp, double *dp, int n);
FloatArray2DoubleArray:
###################################
#  args:                          #
#     a0: address of float array  #
#     a1: address of double array #
#     a2: number of elements      #
#  return:                        #
#     void                        #
###################################
  move $t0, $zero
cfd_startloop:
  beq $t0, $a2, cfd_exitloop
  l.s $f12, ($a0)
  cvt.d.s $f12, $f12
  s.d $f12, ($a1)
  addi $t0, $t0, 1
  addi $a0, $a0, 4  # sizeof(float)
  addi $a1, $a1, 8  # sizeof(double)
  j cfd_startloop
cfd_exitloop:
  jr $ra

#  void printIntArray(int *ip, int n);
printIntArray:
###################################
#  args:                          #
#     a0: address of int array    #
#     a1: number of elements      #
#  return:                        #
```

```
#     void                          #
######################################
  move $a2, $a0
  move $t0, $zero
pi_startloop:
  beq $t0, $a1, pi_exitloop
  lw $a0, ($a2)
  li $v0, 1  # print integer
  syscall
  la $a0, space
  li $v0, 4  # print string
  syscall
  addi $t0, $t0, 1
  addi $a2, $a2, 4  # sizeof(int)
  j pi_startloop
pi_exitloop:
  la $a0, newline
  li $v0, 4  # print string
  syscall
  jr $ra

#  void printFloatArray(float *fp, int n);
printFloatArray:
######################################
#  args:                             #
#     a0: address of float array     #
#     a1: number of elements         #
#  return:                           #
#     void                          #
######################################
  move $a2, $a0
  move $t0, $zero
pf_startloop:
  beq $t0, $a1, pf_exitloop
  lwc1 $f12, ($a2)
  li $v0, 2  # print float
  syscall
  la $a0, space
  li $v0, 4  # print string
  syscall
  addi $t0, $t0, 1
  addi $a2, $a2, 4  # sizeof(float)
  j pf_startloop
pf_exitloop:
  la $a0, newline
  li $v0, 4  # print string
  syscall
  jr $ra

#  void printDoubleArray(double *dp, int n);
printDoubleArray:
######################################
#  args:                             #
#     a0: address of double array    #
#     a1: number of elements         #
#  return:                           #
#     void                          #
######################################
  move $a2, $a0
  move $t0, $zero
pd_startloop:
  beq $t0, $a1, pd_exitloop
  l.d $f12, ($a2)
  li $v0, 3  # print double
  syscall
  la $a0, space
  li $v0, 4  # print string
  syscall
  addi $t0, $t0, 1
  addi $a2, $a2, 8  # sizeof(double)
  j pd_startloop
pd_exitloop:
  la $a0, newline
  li $v0, 4  # print string
  syscall
  jr $ra

#  void printStringArray(char **cp, int n);
```

```
printStringArray:
####################################
#  args:                           #
#     a0: address of char * array   #
#     a1: number of elements        #
#  return:                         #
#     void                         #
####################################
  move $a2, $a0
  move $t0, $zero
  la $a0, mytext
  li $v0, 4
  syscall
ps_startloop:
  beq $t0, $a1, ps_exitloop
  lw $a0, ($a2)
  li $v0, 4  # print string
  syscall
  la $a0, newline
  syscall
  addi $t0, $t0, 1
  addi $a2, $a2, 4  # sizeof(char *)
  j ps_startloop
ps_exitloop:
  la $a0, mytext
  syscall
  jr $ra

# void bsort(void base[], int n, int size, cmpfun)
bsort:
####################################
#  args:                           #
#     a0: address of array base[]   #
#     a1: number n of elements      #
#     a2: size of an element        #
#     a3: address of cmpfun         #
#  return:                         #
#     void                         #
####################################
  push $ra
  push $s0
  push $s1
  push $s2
  push $s3
  move $s0, $a0      # local copies a0..a2
  addi $s1, $a1, -1  # -1 because we need for-loop to n-1
  move $s2, $a2
  move $s3, $a3
b_startwhile:
  li $t7, false      # flag change = false;
  move $t0, $zero    # i = 0
b_startloop:
  beq $t0, $s1, b_exitloop
  mul $t1, $s2, $t0
  add $a0, $s0, $t1  # a0: address of element i
  add $a1, $a0, $s2  # a1: address of element i+1
  push $t0           # save 'i' and 'change' on the stack
  push $t7
  jalr $s3           # call the compare function
  pop $t7            # pop 'i' and 'change' from the stack
  pop $t0
  bgez $v0 b_continue
    # exchange two values; exchange s2 number of bytes
  move $t3, $zero
  b_x_startloop:
    beq $t3, $s2, b_x_exitloop
    lb $t1, ($a0)
    lb $t2, ($a1)
    sb $t2, ($a0)
    sb $t1, ($a1)
    addi $a0, $a0, 1
    addi $a1, $a1, 1
    addi $t3, $t3, 1
    j b_x_startloop
  b_x_exitloop:
  li $t7, true       # flag change = true
b_continue:
  addi $t0, $t0, 1   # i++
```

```
  j b_startloop
b_exitloop:
  bnez $t7, b_startwhile # while (change==true)
  pop $s3
  pop $s2
  pop $s1
  pop $s0
  pop $ra
  jr $ra
```