GENERIC PROGRAMMING

C++ has Templates

- C has void* and function pointers
- How do we write a Linked List that accepts **any** data type?
- How do we apply the same function to every element in a list? Print?

Sorting Integers

```
void bubble1 (int *pArray, int count)
{
  bool bExchange = true;
  int indx;
  int temp;
  while (bExchange)
  {
    bExchange = false;

    for (indx = 0; indx < count - 1; indx++)
    {
      if (*(pArray + indx) > *(pArray + indx + 1))
      {
        temp = *(pArray + indx);
        *(pArray + indx) = *(pArray + indx + 1);
        *(pArray + indx + 1) = temp;
        bExchange = true;
      }
    }
  }
}
```

Generic Sorting

- How do we make the previous sorting function generic?
  a) void bubble (void *arry, int count);
  b) void bubble (void *arry, int count, int elementsize);

- Given b) we can calculate the address of an arbitrary element of arry[k] as follows:

```
(void *) ((char *) arry + k * elementsize)
```

- Address arithmetic cannot be performed on type void *

Compare

- How do we compare two void* items?
  - no data type information

```
#include <string.h>
// if return value < 0 then str1 < str2
// if return value > 0 then str1 > str2
// if return value = 0 then str1 == str2
int memcmp(void* str1, void*  str2, size_t size);


int memcpy(void* dest, void* src, size_t size);
```

size_t

- Look in a C Eclipse Project | Includes | /usr/lib64/gcc/x86_64-suse-linux/4.5/include | stddef.h

- line 208

```
#define __SIZE_TYPE__ long unsigned int

typedef __SIZE_TYPE__ size_t;
```

Generic Sort

```
void bubble2 (void *pArray, int count, int elementSize)
 {
   bool bExchange = true;
   int indx;
   void *pTemp = malloc (elementSize);
   while (bExchange)
   {
     bExchange = false;
     for (indx = 0; indx < count - 1; indx++)
     {
       if (memcmp((void *)((char *) pArray + indx * elementSize),
                  ((void *)((char *) pArray + (indx + 1) * elementSize)),
                  elementSize) > 0)
       {
         // Exchange code here
         bExchange = true;
       }
     }
   }
   free (pTemp);
 }
```

Generic Sort Driver

```
int main ()
{
  int numbersInt[] = {5, 4, 3, 2, 1};
  char numbersChar[] = {'E', 'D', 'C', 'B', 'A'};
  float numbersFloat[] = {5.0, 4.0, 3.0, 2.0, 1.0};
  int i;

  bubble1 (numbersInt, 5);
  bubble2 (numbersChar, 5, sizeof (char));
  bubble2 (numbersFloat, 5, sizeof (float));

  for (i = 0; i < 5; i++)
  {
   printf("%d\n", numbersInt[i]);
  }
  for (i = 0; i < 5; i++)
  {
   printf("%c\n", numbersChar[i]);
  }
  for (i = 0; i < 5; i++)
  {
   printf("%f\n", numbersFloat[i]);
  }

  return 0;
}
```

Generic Sort Results

Results

```
1
2
3
4
5
A
B
C
D
E
2.000000
3.000000
1.000000
4.000000
5.000000
Press any key to continue
```

Why do you think the floating point numbers are not sorted correctly?

Function Pointers

- Since a pointer is just an address, we can have pointers to functions!
- A function can be called using this address
- Function pointers can be passed as arguments to other functions or return from functions

- Define the function pointer
  - **`returnType (*name)(paramType ...)`**

Above main

```
int (*foo) (int);

int negate (int x)
{
     return -x;
}

int square (int x)
{
     return x * x;
}
```

Call a Function Using the Pointer

```
foo = &negate;
printf ("\nNegative of 5 is %d\n", (*foo)
(5));

foo = &square;
printf ("\nSquare of 5 is %d\n", (*foo) (5));
```

Passing Functions as Arguments

```
// Function prototype of callback function
typedef int (*callbackFunct) (const void *pParam1, const void *pParam2);

// Integer compare callback function
int intCompare (const void *pParam1, const void *pParam2)
{
  int int1, int2;

  int1 = *((int *) pParam1);
  int2 = *((int *) pParam2);

  if (int1 == int2)
  {
    return 0;
  }

  return ((int1 < int2) ? -1 : 1);
}
```

Passing Functions as Arguments

```
// Floating-point compare callback function
int floatCompare (const void *pParam1, const void *pParam2)
{
  float float1, float2;

  float1 = *((float *) pParam1);
  float2 = *((float *) pParam2);

  if (float1 == float2)
  {
    return 0;
  }

  return ((float1 < float2) ? -1 : 1);
}
```

Passing Functions as Arguments

```
// Generic sort with compare callback function
void bubble (void *pArray, int count, int elementSize, callbackFunct cmp)
{
  bool bIsSorted = false;
  int index;
  void *pTemp = malloc (elementSize);

  while (!bIsSorted)
  {
    bIsSorted = true;
    for (index = 0; index < count - 1; ++index)
    {
      if (cmp ((pArray + (index * elementSize)),
               (pArray + (index + 1) * elementSize)) > 0)
      {


        switch elements


      }
    }
  }
  free (pTemp);
}
```

Passing Functions as Arguments

- A working version of the previous slides can be found in the class repository with project name **PassingFunctions**

```
int main ()
{
  int i;
  int aInts[] = {5, 4, 3, 2, 1};
  float aFloats[] = {5.0, 4.0, 3.0, 2.0, 1.0};

  bubble (aInts, 5, sizeof (int), intCompare);
  for (i = 0; i < 5; ++i)
  {
    printf ("%5d", aInts[i]);
  }
  puts ("");
```