

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 *array, int count);`
 - b) `void bubble (void *array, int count, int elementsize);`
- Given b) we can calculate the address of an arbitrary element of `array[k]` as follows:
`(void *) ((char *) array + 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

```
// Step 1: Define the prototype of the compare function
typedef int (*cmpFnT) (const void *p1, const void *p2);

// Step 2: Implement each compare function
int IntCmpFn (const void *p1, const void *p2)
{
    int v1, v2;

    v1 = *((int *) p1);
    v2 = *((int *) p2);

    if (v1 == v2)
    {
        return 0;
    }
    return ((v1 < v2) ? -1 : 1);
}
```

Passing Functions as Arguments

```
// Step 1: Define the prototype of the compare function
typedef int (*cmpFnT) (const void *p1, const void *p2);

// Step 2: Implement each compare function
int IntCmpFn (const void *p1, const void *p2)
{
    int v1, v2;

    v1 = *((int *) p1);
    v2 = *((int *) p2);

    if (v1 == v2)
    {
        return 0;
    }
    return ((v1 < v2) ? -1 : 1);
}
```

Passing Functions as Arguments

```
int FloatCmpFn (const void *p1, const void *p2)
{
    float v1, v2;

    v1 = *((float *) p1);
    v2 = *((float *) p2);

    if (v1 == v2)
    {
        return 0;
    }
    return ((v1 < v2) ? -1 : 1);
}
```

Passing Functions as Arguments

```
// Step 3: Implement the generic function (e.g. sorting function)
void bubble3 (void *pArray, int count, int elementSize, cmpFnT cmpFn)
{
    int xchg = false;
    int index = 0;
    void *pTemp = (void *) malloc (elementSize);

    while (!xchg)
    {
        while ((index < count - 1))
        {
            if (cmpFn ((pArray + (index * elementSize)),
                (pArray + (index + 1) * elementSize)) > 0)
            {
                // swap elements .... you write the code

                xchg = true;
            }
            index++;
        }
        if (!xchg)
        {
            return;
        }
        index = 0;
        xchg = false;
    }
    free (pTemp);
}
```

Passing Functions as Arguments

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

```
// Step 4: Write driver to test

int main ()
{
    int a[] = {5,4,3,2,1};
    float b[] = {5,4,3,2,1};
    int i;

    bubble (a, 5, sizeof (int), IntCmpFn);
    bubble (b, 5, sizeof (float), FloatCmpFn);

    return 0;
}
```