Generic Programming in C

*Void **
This is where the real fun starts
There is too much coding
everywhere else! [1]

- Using `void *` and function pointers to write generic code
- Using libraries to reuse code without copying and recompiling
- Using plugins to get run-time overriding and more!

[1] *Zero*
Is where the Real Fun starts.
There's too much counting
Everywhere else!
-Hafiz

Function Pointers

- In C, the name of a function is a pointer!

```
int f1(int x); /* prototype */

int (*func)(int); /* pointer to a fn with an int arg
                     and int return */
func = f1;
n = (*func)(5); /* same as f1(5) */
n = func(5);    /* same as (*func)(5) */
```

- Dereferencing a function pointer just returns a pointer to the function, so we don't need to dereference it before calling it (but we can if we want to).
- We can also have an array of function pointers.

```
    int (*bagOfTricks[10])(int, char *);
```

- The prototype for quicksort function `qsort` in the standard C library uses a function pointer to compare function to enable a generic sort function (man 3 qsort).

```
void qsort(void *base, size_t nmemb, size_t size,
           int(*compar)(const void *, const void *));
```

Function Pointer Examples

- Function Pointer Example 1
  - C-examples/function-pointers/funkyfun.c
  - C-examples/function-pointers/funky.c
  - C-examples/function-pointers/funky.h
- Function Pointer Example 2
  - C-examples/function-pointers/test-qsort.c

In-class Exercises

- **Exercise 1**. Write the declaration of a variable named `myFuncPtr` that is a pointer to a function that takes two arguments of type `char *` and `int`, respectively, and returns a `char *` type.
- **Exercise 2**. Modify the test-qsort.c to sort the address structs by name of the student. You will have to write a `compareName` function for names and modify the call to `qsort` appropriately.

Function Pointer Examples

- Function Pointer Example 3
  - C-examples/function-pointers/fun-with-fns.c

In-class Exercises

- **Exercise 3**. Write the declaration of a variable named `myFuncCollection` that is an array of four pointers to a function that takes two arguments of type `char *` and `int`, respectively, and returns a `char *` type.
- **Exercise 4.** (Challenging!) Write the declaration of a variable named `myFuncCollection` that is a pointer to pointers to a function that takes two arguments of type `char *` and `int`, respectively, and returns a `char *` type. Declare space for this variable using `malloc` and allocate space for four pointers.

Function Pointer Examples

- Function Pointer Example 4
  - C-examples/objects/Address.h
  - C-examples/objects/Address.c
  - C-examples/objects/testAddress.c

In-class Exercises

- **Exercise 5**. Draw the memory layout of the struct address just after it is malloced in the createAddress from the file Address.c.
- **Exercise 6**. Draw the memory layout of the struct address just before the return in the createAddress from the file Address.c.
- **Exercise 7**. Write a function to free the memory allocated in createAddress function. Use the following prototype:

```
int freeAddress(struct address *this);
```

- **Exercise 8**. Write a generic function to check if two address structs are equal. The function should return 1 if they are equal and 0 otherwise. Where would you declare this function in the header file? Use the following prototype:

```
int equals(const void *this, const void *other);
```

Function Pointers: Function Pointer Example 5 (1)

We can add pointers to functions that operate on the linked list as members of the `list` structure itself.

```
struct list {
    int size;
    struct node *head;
    struct node *tail;
    int (*equals)(const void *, const void *);
    char *(*toString)(const void *);
    void (*freeList)(void *);
};
```

- `int (*equals)(const void *, const void *)`
  A pointer to a function that takes two generic pointers and returns an int (0 for `false`, 1 for `true`).
- `char *(*toString)(const void *)`
  A pointer to a function that takes a generic pointer and returns a string.
- `void (*freeList)(void *)`
  A pointer to a function that takes a generic pointer and returns nothing.

Function Pointers: Function Pointer Example 5 (2)

```
struct list *list = (struct list *) malloc(sizeof(struct list));

list->toString = printItemShort; /* assign custom function */
list->toString(list);            /* call toString function */

list->toString = printItemLong;  /* assign different function */
list->toString(list);            /* call toString function */
```

Generic Programming

- The type `void *` is used as a generic pointer in C (similar in concept to `Object` type in Java). Pointers to any type can be assigned to a variable of type `void *`, which allows `polymorphism`.
- The following example shows a polymorphic min function that works for an array of any type as long as we have a compare function for two elements of the array.

```c
/* Find the index for the min in v[0..size-1], assumes size
   is > 0 */
/* Each V[i] is pointer to a generic object, so has the type
    void  * */
int min(void *v[], int size,
        int (*compare)(const void *, const void *))
{
    int i;
    int min = 0;
    for (i = 1; i < size; i++) {
        if (compare(v[i], v[min]) < 0) {
            min = i;
        }
    }
    return min;
}
```

See full example here: min.c

Integer Quicksort

```
void swap(int v[], int i, int j)
{
    int *tmp = v[i]; v[i] = v[j]; v[j] = tmp;
}

/* qsort: sort v[left]..v[right] into increasing order */
void qsort(int v[], int left, int right)
{
    int i, last;

    if (left >= right) /* do nothing if array contains */
        return;        /* less than two elements */
    swap(v, left, (left + right)/2);
    last = left;
    for (i = left+1; i<= right; i++) {
        if (v[i] < v[left]) {
            swap(v, ++last, i);
        }
    }
    swap(v, left, last);
    qsort(v, left, last-1);
    qsort(v, last+1, right);
}
```

Polymorphic Quicksort

```c
void swap(void *v[], int i, int j)
{
    void *tmp = v[i]; v[i] = v[j]; v[j] = tmp;
}

/* qsort: sort v[left]..v[right] into increasing order */
void qsort(void *v[], int left, int right,
            int (*compare)(void *, void *))
{
    int i, last;

    if (left >= right) /* do nothing if array contains */
        return;        /* less than two elements */
    swap(v, left, (left + right)/2);
    last = left;
    for (i = left+1; i<= right; i++) {
        if (compare(v[i], v[left]) < 0) {
            swap(v, ++last, i);
        }
    }
    swap(v, left, last);
    qsort(v, left, last-1, compare);
    qsort(v, last+1, right, compare);
}
```

In-class Exercises

- **Exercise 9**. Write a *polymorphic* max function that works for an array of any type.

```
/* Find the index for the max in v[0..size-1],
    assumes size is > 0 */
int max(void *v[], int size,
    int (*compare)(const void *, const void *))
{

}
```

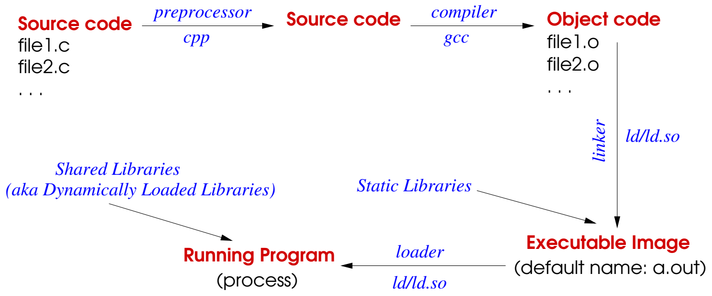
- **Exercise 10.** (More Challenging!) Evolve the following function into a polymorphic version.

```
/* Search for key in v[0..size-1] and return index if
     found, -1 otherwise. Assumes size is > 0 */
int linearSearch(int v[], int size, int key)
{
    int index = 0;
    for (index = 0; index < size; index++)
        if (v[index] == key)
            return index;
    return -1;
}
```

Libraries

- A library is a collection of code that implements commonly used methods or patterns with a public API. This is combined with generic code to facilitate code re-use.
- Libraries can be shared (also known as Dynamically Linked Libraries or DLLs) or be static.
- Is a shared library part of the process or is it a resource? It should be viewed as a resource since the operating system has to find it on the fly.
- A static library, on the other hand, becomes part of the program text.
- The concept of re-entrant code,i.e., programs that cannot modify themselves while running. Re-entrant code is necessary to write libraries.

Recall: Compiling, Linking and Running Programs



Note: The above illustrates the GCC compiler tool chain but the concepts are the same for any C/C++ compiler.

Standard C Libraries

- The standard C library is automatically linked into programs when you compile with GCC.
- The shared version of the library can be found at `/usr/lib/libc.so`
- If the static libraries are installed on your system, the static version of the standard library can be found at `/usr/lib64/libc.a` (or `/usr/lib/libc.a`)
- Check out other libraries in the `lib` directory.
- Any other libraries need to be explicitly included and the linker needs to know where find them.

Library Conventions

- A library filename always starts with **`lib`**.
- File suffixes

.a	Static libraries
.so	Shared libraries

- The library must provide a header file that can be included in the source file so it knows of function prototypes, variables, etc.

Creating a Shared Library (Linux) (1)

- Suppose we have three C files: `f1.c,` `f2.c`, and `f3.c` that we want to compile and add into a shared library that we will name `mylib`. First, we can compile the C files with the flags `-fPIC -shared` to the `gcc` compiler.

```
gcc  -I. -Wall -fPIC -shared   -c -o f1.o f1.c
gcc  -I. -Wall -fPIC -shared   -c -o f2.o f2.c
gcc  -I. -Wall -fPIC -shared   -c -o f3.o f3.c
```

- Then we can combine, the three object files into one shared library using the `ld` linker/loader.

```
ld -fPIC -shared -o libmylib.so f1.o f2.o f3.o
```

- Now we can compile a program that invokes functions from the library by linking it with the shared library.

```
gcc -I. -L. test1.c -o test1 -lmylib
```

  The compiler will search for the shared library named libmylib.so in the current folder because of the `-L.` option.
- See the CS253-resources/libraries example.

Creating a Shared Library (Linux) (2)

- If we use the `-L` option, then the compiler searches the folder(s) provided with that option before searching in the system folders for libraries.
- If your shared library is in some other folder (than the current folder), specify that folder with the `-L` option to the compiler. For example, if your library is in the sub-folder `lib` underneath the current folder, you can use

  ```
  gcc test1.c -o test1 -Iinclude -Llib -lmylib
  ```
- When you run the executable, again the system has to be able to find the shared library. If it is not in the current folder (or installed in a system library folder), then the *loader* uses the environment variable `LD_LIBRARY_PATH` to find a set of folders to search in. For example:

  ```
  export LD_LIBRARY_PATH=$LD_LIBRARY_PATH:.:./lib
  ```
- **In-class Exercise**: Explain what folders (and in what order) will the loader search when looking for a library with the above declaration.

Creating a Static Library

Suppose we have three C files: `f1.c`, `f2.c`, and `f3.c` that we want to compile and add into a static library that we will name `mylib`.
First, we will compile the C files with the flag `-fPIC` to the `gcc` compiler.

```
gcc  -I. -Wall -fPIC   -c -o f1.o f1.c
gcc  -I. -Wall -fPIC   -c -o f2.o f2.c
gcc  -I. -Wall -fPIC   -c -o f3.o f3.c
```

Then we can combine, the three object files into one static library using the `ar` archive program.

```
ar rcv libmylib.a f1.o f2.o f3.o
```

At this point, we can write a test program that invokes functions from the library and link it with the static library.

```
gcc -I. -Wall -static -L. test1.c -lmylib -o test1.static
```

The rules for finding a static library are the same as for shared libraries. You will need to have a static version of the standard C library installed. You can do that with the following command (on Fedora Linux):

```
sudo dnf install glibc-static
```

Shared libraries: dynamic versus static linking

- Dynamic linking. This is the default. Here the library is kept separate from the executable, which allows the library code to be shared by more than one executable (or more than one copy of the same executable). Makes the executable size smaller leading to faster loading up time. The downside is that if the we move the executable to another system but the library is missing there, the executable will not run!
- Static linking. With static linking we can create a self contained executable. We need to create a static library that we can then link to using the `-static` option with the C compiler. See the output below. Note that static executable is very large in comparison to the dynamically linked executable.

```
$ ls -l
-rwxr-xr-x  1 amit home   2503 Aug 30 00:16 libmylib.so
-rwxr-xr-x  1 amit home   5220 Aug 30 00:16 test1
-rwxr-xr-x  1 amit home 404507 Aug 30 00:16 test1.stati
```

Using Shared Libraries in Linux (1)

- We can use libraries to help organize our own projects as well. For example, we can create a library that contains our list, queue, and stack classes. Then we can link to that library to use in our projects instead of having to copy the code.
- We can build a library and simply copy the library into the same folder as our executable program. The system will find the library since it is in the same folder. But there are better ways of dealing with libraries.

Using Shared Libraries in Linux (2)

- Here are three better ways of organizing your library code with respect to the code that uses it.
  - Per project. Create a `lib` folder to hold your libraries, an `include` folder for the header files inside your project. With this approach, your project is self-contained although you do have to copy the library into each project.
  - Per user. Place the `lib` and `include` folders at some fixed location in your home directory. Then specify these folders to the C compiler (via the `-I` and `-L` options) when you are building projects that use those libraries. Also export the `LD_LIBRARY_PATH` variable appropriately.
  - Per system. Install the library into one of the standard system library folders like `/lib`, `/usr/lib`, `/usr/local/lib` etc. Install the header files in a standard system header file folder like `/usr/include`, `/usr/local/include` etc.
    - On Linux, use the command `ldconfig` to register the new libraries with the compiler. See man page for ldconfig.

Using Shared Libraries in Linux (3)

- Assuming that we have installed our libraries in `lib` and the header files in `include`, which are subfolders in the project folder.
- When a program starts, the system will search for libraries using the environment variable name `LD_LIBRARY_PATH`. We can add the following line to the end of the file `~/.bashrc`.
  ```
  export LD_LIBRARY_PATH=$LD_LIBRARY_PATH:./lib:.
  ```
- You will also have to add the option `-L./lib` when you compile your programs so that the compiler will search for libraries in the subfolder named `lib` in the current folder.

Using Shared Libraries in Linux (4)

- Similarly, we can place all the relevant header files (like `Job.h`, `Node.h`, and `List.h`) in the the directory `include` in your project folder. Then include the option `-I./include` anytime you are compiling a program that uses the List class. You can now include the header files as follows:

  ```
  #include <Job.h>
  #include <Node.h>
  #include <List.h>
  ```

  The compiler will automatically search for the files in the `include` directory in your current folder.
- After the above two steps, you can simply link in your library without having to copy the code to each new directory you work in.
  ```
  gcc -Wall -I./include -L./lib -o prog1 prog1.c -lmylib
  ```

How to check for library dependency?

- **Linux**: Use the tool ldd. For example:

```
$ ldd /bin/date
    linux-vdso.so.1 (0x00007ffc2734e000)
    libc.so.6 => /lib64/libc.so.6 (0x00007f753f161000)
    /lib64/ld-linux-x86-64.so.2 (0x000055f379c2f000)
```

- **MS Windows**: Use the tool depends (available from `http://www.dependencywalker.com`).
- **MacOSX**: Use the tool otool.