Large Programs and External Libraries (Continued)

1 Makefiles

- When we have a number of files to compile together we need a rule-set to perform this
- The `make` command provides this
- Requires a rule-file called the `Makefile`
- Declarative programming style set of rules for building the program
- Format of each rule:

```
target [target ...]: [component ...]
        [command 1]
        ...
        [command n]
```

- N.B. Tab character
- `target` - what you want to make
- `component` - something which needs to exist (might need another rule)

2 Makefiles: Example

- Files: `main.c`, `counter.h`, `counter.c`, `sales.h`, `sales.c`

```
all: counter.o sales.o main.c
        gcc -o program main.c counter.o sales.o

counter.o: counter.c counter.h
        gcc -c counter.c

sales.o: sales.c sales.h
        gcc -c sales.c

clean:
        rm -rf program counter.o sales.o
```

3 Makefiles: Macros

- Macros can be used to store definitions
  - `AUTHOR = Konrad Dabrowski`
- They can be generated from commands
  - `DATE = `date``
- And used in the `Makefile`

```
all:
        echo $(AUTHOR) compiled this on $(DATE)
```

- `all:`
- Running this gives:
- `echo Konrad Dabrowski compiled this on `date``
  - `Konrad Dabrowski compiled this on Thu 16 Jan 10:54:36 GMT 2020`

4 Makefiles: Pattern Rules

- We can specify a pattern rule which matches multiple files
- e.g. compile C files into object files:

```
DEPS = counter.h sales.h
%.o: %.c $(DEPS)
	gcc -c $< -o $@
```

- This would change our original Makefile example to:

```
all: counter.o sales.o main.c
	gcc -o program main.c counter.o sales.o

%.o: %.c $(DEPS)
	gcc -c $< -o $@

clean:
	rm -rf program counter.o sales.o
```

5 Makefiles: A few comments

- Comments - lines starting with `#`
- Lazy evaluation
- If a target exists and has a timestamp later than all of its components assume it is up to date and don't bother to re-process
- Nothing to do with C
- Although Makefiles are often used with C programs there is no intrinsic link - can use with any code/work
- You can run any specific rule by invoking its target:
- `make sales.o`

6 A: Singly-linked list

- Implement a program that stores numbers in a linked list and then prints them out.
- The nodes of your linked list should be an appropriate `struct` type and should be dynamically allocated using `malloc()`.
- The program should let the user input numbers at runtime and should contain a function that adds nodes to the end of the list.
- (Note that pressing `Ctrl+D` makes `scanf()` return `EOF`.)

7 B: More advanced linked list

Add functions to:

- delete the last number in the list
- add a number to the start of the list
- search for a number in the list and return either a pointer to it or `NULL` if the number is not in the list

8 C: Doubly-linked list

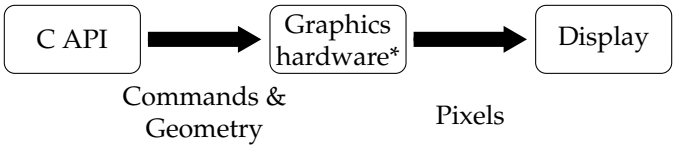
- Change your code to use a doubly-linked list.
- Add a function which takes a pointer to a node in the list and deletes the corresponding node from the list. Remember to `free()` the memory used by the node!

9 Implement `calloc()` and `realloc()`.

- Write functions `calloc2()` and `realloc2()`, that use `malloc()` and `free()` to implement the functionality of `calloc()` and `realloc()`, respectively.
- Remember that `calloc()` sets the allocated memory to zero (for this exercise, you may ignore testing for integer overflows when multiplying the arguments of `calloc()` together).
- When implementing the copying part of `realloc()`, recall that `char` is 1 byte; the C standard states that you may use `char *` pointers to access individual bytes of memory.

10 External libraries

- One of the reasons why C is so popular is the huge collection of tried and tested libraries available across many different computing platforms. E.g. OpenGL



- Commands from your program are sent by the API to the graphics hardware which generates pixels for display
- *in OpenGL the hardware behaves as a state machine

11 OpenGL programming

- On its own OpenGL is:
  1. Low level
  2. O/S independent
- Hence it is usually used with:
  - GLU a utility library with high level shape support
  - GLUT utility library for window creation and I/O

12 Commonly used C libraries

- general: `libglib` / `libgobject` / `libpthread`
- console: `libncurses`
- 2D graphics: `libX11` / `libSDL`
- 3D graphics: `libGL` / `libGLU` / `libGLUT`
- GUI toolkits: `libgtk` / `libQT`
- Images: `libjpeg` / `libpng` / `libgif`
- text rendering: `libpango` / `libfreetype`
- sound: `libasound` / `libSDL`
- compression: `libz` (`zlib`) / `libgzip` / `libbz2`
- encryption: `libcrypt` / `libssl` / `libgssapi` / `libkrb5`
- XML: `libxml2`
- web: `libcurl`

13 Usage of libraries

- If a library is *statically linked* then a copy of the library is included in the executable
- C/C++/assembly can be combined
- Often bound to other languages e.g. php, XML, curl
- Many of these libraries will be *dynamically linked*
- LGPL (Lesser Gnu Public License) often used
- Try `ldd /usr/bin/php` on Linux to list dynamic dependencies

14 Dynamic vs static linking

- Dynamic linking takes place at run-time not build-time
- Reduces filespace demands (bloat) by keeping only one copy of the library
- Can help with updates e.g. for security
- Dynamic libraries are called differently by OSs
- Linux: shared objects (`.so`)
- Windows: Dynamic Link Libraries (`.dll`)
- OSX: `.dylib`
- Can lead to "DLL Hell": many versions of the same dynamic library
- Best to include version number with library