Welcome back to section! We’re glad that you’re here :)

**Casting in C++**

While in C++, we want to use casts that are more explicit in their behaviour. This gives us a better understanding of what happens when we read our code, because C-style casts can do many (sometimes unwanted) things. There are four types of casts we will use in C++:

\[
\text{static\_cast<to\_type>(expression);} \\
\quad \text{★ Converts between pointers of related types.} \\
\quad \text{ Compiler error if not related.} \\
\quad \text{★ Performs not pointer conversion (e.g. float to int conversion).}
\]

\[
\text{dynamic\_cast<to\_type>(expression);} \\
\quad \text{★ Converts between pointers of related types.} \\
\quad \text{ Compiler error if not related.} \\
\quad \text{ Also checks at runtime to make sure it is a 'safe' conversion (returns nullptr if not).}
\]

\[
\text{const\_cast<to\_type>(expression);} \\
\quad \text{★ Used to add or remove const-ness.}
\]

\[
\text{reinterpret\_cast<to\_type>(expression);} \\
\quad \text{★ Casts between incompatible types without changing the data.} \\
\quad \text{ The types you are casting to and from must be the same size.} \\
\quad \text{ Will not let you convert between integer and floating point types.}
\]

**Exercise 1**

For each of the following snippets of code, fill in the blank with the most appropriate C++ style cast. Assume that we have the following classes defined:

```cpp
class Base { 
public: 
  int x; 
}; 
class Derived : public Base { 
public: 
  int y; 
};
```

```cpp
int64_t x = 0x7fffffffffe870; 
char* str = ____________________________(x); 
void foo(Base *b) { 
  Derived *d = _____________________________(b); 
  // additional code omitted 
} 
Derived *d = new Derived; 
Base *b = _____________________________(d); 

double x = 64.382; 
int64_t y = _____________________________(x); 
```
Networking Quick Review

Exercise 2

a) What are the following protocols used for? (Bonus: In what layer of the networking stack is it found?)

- DNS
- IP
- TCP
- UDP
- HTTP

b) Why would you want to use TCP over UDP?

c) Why would you want to use UDP over TCP?
Step-by-step Client-Side Networking

Step 1. Figure out what IP address and port to talk to. (getaddrinfo())

```c
int getaddrinfo(const char *hostname,           // hostname to lookup
    const char *servname,         // service name
    const struct addrinfo *hints, // desired output
                (optional)
    struct addrinfo **res);       // results structure
```

Step 2. Create a socket. (socket())

```c
int socket(int domain,                          // AF_INET, AF_INET6, etc.
    int type,                                   // SOCK_STREAM, SOCK_DGRAM, etc.
    int protocol);                              // usually 0
```

Step 3. Connect to the server. (connect())

```c
int connect(int sockfd,                           // fd from step 2
    struct sockaddr *serv_addr,  // socket addr from step 1
    socklen_t addrlen);          // size of serv_addr
```

Step 4. Transfer data through the socket. (read() and write())

```c
ssize_t read(int fd, void *buf, size_t count);  

ssize_t write(int fd, void *buf, size_t count);
```

These are the same POSIX calls used for files, so remember to deal with partial reads/writes!

Step 5. Close the socket when done. (close())

```c
int close(int fd);
```
Exercise 3
Fitting the Pieces Together. The following diagram depicts the basic skeleton of a C++ program for client-side networking, with arrows representing the flow of data between them. Fill in the names of the functions being called, and the arguments being passed. Then, for each arrow in the diagram, fill in the type and/or data that it represents.

1. `____________ (hostname, servname, ___, ___)`
2. `____________ (___, type, protocol)`
3. `____________ (___, ________, addrlen)`
4. `read(____, buf, count)
write(____, buf, count)`
5. `close(____)`

Dotted boxes should be filled in with the data type.