CSE 333 Section 8 - Casting & Client-Side Networking

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++:

`static_cast<to_type>(expression);`

- ★ Converts between pointers of related types.
  - Compiler error if not related.
- ★ Performs not pointer conversion (e.g. float to int conversion).

`dynamic_cast<to_type>(expression);`

- ★ Converts between pointers of related types.
  - Compiler error if not related.
  - Also checks at runtime to make sure it is a 'safe' conversion (returns `nullptr` if not).

`const_cast<to_type>(expression);`

- ★ Used to add or remove const-ness.

`reinterpret_cast<to_type>(expression);`

- ★ Casts between incompatible types *without changing the data*.
  - The types you are casting to and from must be the same size.
  - 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:

```
class Base {
 public:
  int x;
};
```

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

```
int64_t x = 0x7ffffffffe870;
char* str = _____(x);
```

```
void foo(Base *b) {
  Derived *d = _____(b);
  // additional code omitted
}
```

```
Derived *d = new Derived;
Base *b = _____(d);
```

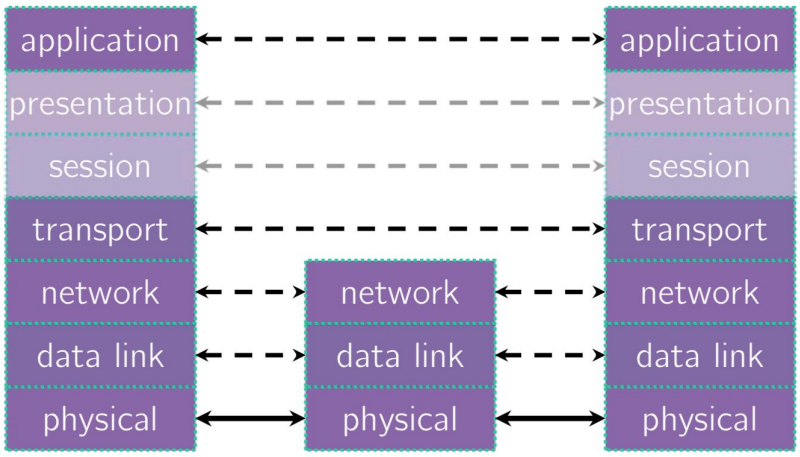
```
double x = 64.382;
int64_t y = _____(x);
```

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()`)

```
// returns 0 on success, negative number on failure
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

struct addrinfo {
     int ai_flags;              // additional flags
     int ai_family;             // AF_INET, AF_INET6, AF_UNSPEC
     int ai_socktype;           // SOCK_STREAM, SOCK_DGRAM, 0
     int ai_protocol;           // IPPROTO_TCP, IPPROTO_UDP, 0
     size_t ai_addrlen;         // length of socket addr in bytes
     struct sockaddr* ai_addr;  // pointer to socket addr
     char* ai_canonname;        // canonical name
     struct addrinfo* ai_next;  // can have linked list of records
}
```

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

```
// returns file descriptor on success, -1 on failure (errno set)
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()`)

```
// returns 0 on success, -1 on failure (errno set)
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()`)

```
// returns amount read, 0 for EOF, -1 on failure (errno set)
ssize_t read(int fd, void *buf, size_t count);

// returns amount written, -1 on failure (errno set)
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()`)

```
// returns 0 for success, -1 on failure (errno set)
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.