Basics for C++

Picked from Lippman 5th edition

Arrays

- Arrays have a fixed size
- Array declarator has the form `a[d]` for type `a` and dimension `d`. The dimension must be known at compile time, hence a `constexpr`.
- Values inside the array are default initialized:
    1. If outside a function, set to '0'
    2. Inside a function, uninitialized
- Cannot use auto to deduce type of the array
- No arrays of references
- Can do list initialization, which also allows skipping the dimension `d`.
- Character arrays can be initialized thorough string literals. Need to take into account the space for a null.
- Cannot copy initialize or assign arrays.
- To define a pointer or reference to an array, you need to use brackets: `int (*PArray)[10] = &arr`.
- Read decarations inside to out.
- Can use a ranged for loop on arrays.
- Compiler does not do bound checking on `[]` operand value.
- Arrays usually devolve to pointers.
- The name of the array `arr` becomes synonymous to `&arr[0]`.
- This means when you use `auto p = arr;` where `int[10] arr`, the deduced type is `int *`.
- `decltype(arr)` returns `int[10]` faithfully.
- Pointers can act as iterators on arrays.
- You can use `begin` and `end` on arrays to get 'iterators' just like vectors.
- Pointers hence support dereference, increment, comparisons etc-.
- `int *ap = ap2+4` does not add 4 to the value, but adds 4*`sizeof(*ap)`, essentially it is like `&ap[4]`.
- Difference of pointers follows the same logic.
- Adding 0 to null pointers or subtracting two nul pointers is okay.
- `*ia + 4` does the dereference first.
- The builtin subscript operator on arrays allows for `ia[-2]` kind of expressions.
- C style strings are a convention of using character arrays that are null terminated.
- Doing comparisons on C-style strings ends up comparing the pointers.
- Try using `strcmp(s1, s2) < 0` or something.
- While using `strcpy`, make sure there is enough space first.
- We can use `.c_str` on string to get a C-style string, but it has type `const`

`char *`.

- Arrays of arrays are multi-dimensional arrays in C++.
- `int ia[3][4]` is array of size 3, each element is array of size 4, each element is int.
- We can define references to array, so we can write a statement like `int (&row)[4] = ia[1]` to get an alias for the second row.
- To use a ranged for loop, use code like:

```
for (auto &row : ia) {
    for (auto col : row) {
        col = cnt;
        ++cnt;
    }
}
```

- We used a reference since otherwise the `auto` would make a pointer out of it.
- Try using `begin` and `end` instead.
- An array typedef looks like `typdef int intarray[4]`.

Ranged for loop

- Anything can go into after the colon `:`, that defines `begin` and `end`.
- This allows for braced initializer list, arrays, and vectors etc-.

Type conversions

- Implicit conversoins happen without programmer knowledge, and occur at:
  1. Different integral types are promoted to a common large integral type in an expression.
  2. In conditions, non-bool are convereted to bool
  3. In initializations, the initializer is converted to the type of the lhs.
  4. For operands of mixed types on arithmetic operators, they are converted to a common type.
  5. During function calls.
- Arithmetic conversions usually happen in two steps, first to integral level and then higher. Unsigned vs signed is often machine dependent.
- Array to pointer conversions are frequent wherever demanded. Not done only when `decltype` is used.
- 0 and nullptr can convert to any pointer type, a pointer to nonconst can convert to `void*` and any pointer can convert to `const void *`. Inheritance also allows for a few.
- Pointer to nonconst can convert into pointer to const.
- Classes themselves can selectively define some conversions.
- Type casts give a way to do explicit type conversions, could be `static_cast`, `const_cast` or `reinterpret_cast`

Functions

- A function is a block of code with a name, usually called using the call operator `()`.
- During a function call, function parameters are initialized and then control is transferred.
- Execution ends when a return type statement is found
- Arguments provide values for the initialization of the parameters.
- An empty parameter list still needs to be specified as `()`.
- Might instead say `void f(void)`.
- Need to specify type of each parameter separately.
- Unused parameters can be left nameless.
- The return type of a function cannot be an array or a function,but it could be a pointer to those.
- Scope: part of program text where name is visible, lifetime: part of execution that object exists.
- Parameters and other local variables are automatic objects, meaning they lose identity after block scope is over.
- To have an object whose lifetime continues across calls to the function, define it as `static`. Initialized before the first time it is defined. Eg:

```
size_t count_calls() {
    static size_t ctr = 0;
    return ++ctr;
}
```

- A function declaration is definition minus the body. Also called function prototype.
- Declare the functions in header and define them in a seperate source code file that imports this header.
- To enable seperate compilation, provide `-c` flag which gives `.o` as output. Link the object to files together to get `./a.out`.
- Arguments can be passed by reference by declaring the parameter as a reference type.
- Pointers give indirect access to shared data.
- It is ineffeicient and sometimes impossible to copy objects, hence it is safer to use references.
- Unchanged reference parameters should be declared const.
- References can also be useed to return additional information to the calling environment.
- Just as in normal initialization, top-level consts are ignored between parameter and argument.
- This means that the following functions cannot be distinguished so the second declaration is in error:

```
void fcn(const int i);
void fcn(int i);
```

- The basic general rules for initialization are:
  1. ignore top level consts
  2. can use nonconst argument for low level const parameter, but not vice versa
  3. plain reference must be initialized by object of exact same type (cannot pass literals, expressions, or even const references)
- So always try to make references const if possible.
- Arrays interefere in two ways: you cannot copy them and they usually devolve into pointers.
- So the following three are the same:

```
void print(const int *);
void print(const int[]);
void print(const int[10]);
```

- This means size of the array is unknown to function, specified through one of three ways:
  1. Special marker after last object of array.
  2. Sentinel pointer also passed.
  3. Explicit size parameter passed.
- The same discussion of references applies to pointers, since pointers to const can accept nonconst, but not vice-versa.
- To define an array reference parameter, we can use `void print(int (&arr)[10])`, where the bracket is important.
- To pass a multidimensional array, we would have to essemtially pass a pointer to the first element of the outermost array, which means you need to know the dimensions of all the rest of the arrays.
- Command line parameters to functions are passed to main through the "explicit size parameter" approach `int main(int argc, char **argv)`.
- To pass varying number of arguments of the same type `a`, you can use `initializer_list<a>`. It provides functionalities to assign the list, get the begin and end iterators (for range based for loops) and getting the size.
- When you want to pass multiple values to a `initializer_list` type parameter, enclose them in curvy braces.
- An example:

```
void err_msg(initializer_list<string> il) {
    for (auto &s: il) {
        cout << s << " ";
    }
    cout << endl;
}
string s = "bye";
err_msg({"hello", s});
```

- We can instead use ellipses to interface with C code, but isn't recommended, since varargs objects are usually not copied correctly under C++.