RICHARD ROBINSON

C, UNIX, & JAVA

Intro to C

The Basics

C is a command line language. The basic syntax of any C program code is

```
#include <stdio.h>
int main(void) {
    /* Program */
}
```

If the program takes command line arguments, then the function instead becomes

```
int main(int argc, char *argv[])
```

A string is treated as an array of characters where the last element is `\0` and is declared by `char *`. For a constant variable, use `const` as a prefix in the declaration. Alternatively,

```
#define NAME value
```

may be used. Numbers in base 8 and 16 are prefixed by `0` and `0x`, respectively. An enumeration is an array of constants, with syntax

```
enum type { c1 = value, c2, c3 = value ...};
int x = c2
```

where the values are optional.

Formatting Output

The `printf()` function outputs text. In the output string, `\%x.yf` formats the respective float such that there are x and y characters to the left and right of the decimal point, respectively.

For example, `printf("%6.1f string", 42.42)` outputs `42.4 string`.

To cast a variable is to change it from one type to another, via

```
int var = (int)(var2);
```

where `var2` need not be an int. Additionally, use `x.f` to convert the int x to type float.

Conditionals & Loops

The ternary condition `x = exp1 ? a : b` is equivalent to

The code `c = (a > b)? a : b` is equivalent to `c = max(a,b)`.

```
if (exp1) x = a;
else x = b;
```

The switch statement is a condensed if-else-if-else... statement, with the general form

```
switch (exp) {
    case 1: statements; break;
    case 2: statements; break
    ...
    default: statements
}
```

Note **`break`** is optional. If not included, the next case may be executed. Otherwise, it branches out of a statement or loop (except for if-else).

The **`continue`** command is used in a nested if statement to skip a certain iteration(s) of the outer loop. It is often used to skip certain elements, as follows:

```
while (exp) {
    if (condition) continue; /* skip statements */
    statements;              /* if !condition... */
}
```

Functions

A function must be declared before it is called. The general format is:

```
int name(int, int);
/* code that calls function */
int name(int x, int y) {
    /* code goes here */
    return z;
}
```

The argument type must match the function type. A function who's type is **`void`** has no return value.

Note an argument of type **`char[]`** may be used in an **`int`** function.

The `assert(condition)` function terminates a program if the condition is false. It is exclusively used for debugging.

Software Design

Scope

Variables prefixed with `static` have their scope restricted to the file they are in. Header files imported via

```
#include "filename.h"
```

are used to store declarations and code shared over multiple source files, and effectively act as a macro. A pointer is declared using `*` where `&` is the address operator. The syntax is used as

```
int i;
int *j = &i;
```

Pointers are mainly used within functions; any changed made to the local variables within the function will not affect nor change the variables of the calling function, used as:

```
func(&a, &b);
void func(int* x, int* y) {
    /* code */
}
```

Additionally, `*var` is effectively `var[]`. In this case, `var` is `&var[0]` when used in an expression, and `s[i]` is `*(s+i)`.

An array can contain pointers with syntax `int *arr[N]`.

Arrays

The line `char str[] = "string"` is equivalent to `char str[] = {'s','t',...}`.

An array whom's size is larger than the number of elements has its remaining elements as 0. Size need not be specified. The number of elements for such array is

```
num_elements = sizeof(arr) / sizeof(arr[0]);
```

A function may return a pointer, which would have the form

```
int* func(void) {
```

```
    return &var;
}
```

Like any array, a string may be indexed at any element. An important standard function is

```
char *strstr(const char *s, const char *t)
```

which searches for `t` in `s`. When indexing an array of strings, use `char *str[] = {"string"}`. A for loop must be used to then iterate through each char in each string. Alternatively,

```
do {
    c = getchar();
    /* code */
} while (c != '\0' && c != '\n');
```

may be used.