C Reference Sheet v0.5

Made for CS-2113 by Dr. T. Wood @ GWU

The Simplest Program

```
#include <stdio.h>
int main ()
{
   printf("CRUSH all humans!\n");
   return 0;
}
```

Control Structures

```
/* if-statement: */
if (i == 1) {
    printf ("one\n");
}
else if (i == 2 || i == 3) {
    printf ("two or three\n");
}
else {
    printf ("not 1, 2, or 3\n");
}

/* for-loop example: */
int i; // cannot declare inside loop
printf ("Numbers 0 through 9: \n");
for (i=0; i < 10; i++) {
    printf (" %d\n", i);
}

/* while-loop example*/
float f = 0;
printf ("Some floats < 10: \n");
while (f < 10) {
    printf (" %f\n", f);
    f+=1.75;
}

/* switch statement example: */
switch (i) {
  case 1: {
    printf ("one\n");
    break;
  }
  case 2: {
    printf ("two\n");
    break;
  }
  default: {
    printf ("something else\n");
  }
}
```

Variables and Arrays

```
int i; // must declare at top of block
float j = 10.341; // declare and set

int profits[52];
profits[0] = 100; // first entry
profits[51] = 1000; // last entry

float temps[10]; // can use any type
temps[4] = 84.512;
```

Warning: vars start filled with junk!

Type	Storage*	Range	printf label
(unsigned) char	1 byte	one letter (or -128 to 128)	%c or %d
(unsigned) int	4 bytes	-2,147,483,648 to 2,147,483,647	%d (%u)
(unsigned) short	2 bytes	-32,768 to 32,767	%d (%u)
(unsigned) long	4 or 8 bytes	-2,147,483,648 to 2,147,483,647	%ld (%lu)
(unsigned) long long	8 bytes	-9,223,372,036,854,775,808 to 9,223,372,036,854,775,807	%lld (%llu)

Type	Storage	Approximate Precision	printf label
float	4 Bytes	6 decimal places	%f
double	8 Bytes	15 decimal places	%lf
long double	10 Bytes	19 decimal places	%llf

*Exact storage size varies by system. These are typical for 32bit architectures.

Compiling, and Running Code

```
gcc file.c -o execname
./execname
```

Pointers and Dynamic Memory

```
int *int_ptr;
int a = 50;
int b;

// Pointers reference other addresses
// Use & to get an address
int_ptr = &a;
// Use * to dereference a pointer and
// set value of address it points to
*int_ptr = 55;
// These 2 lines are identical
b = a;
b = *int_ptr;

// Use malloc to get heap memory
// Allocate enough space for int
int_ptr = (int*)malloc(sizeof(int));
// Change the value pointed at
*int_ptr = 123;
// Need to free data when done!
free(value);

// Can use pointers like arrays
int_ptr =
(int*)malloc(10*sizeof(int));
// Change first and last entries
int_ptr[0] = 0;
int_ptr[9] = 42;
```

Structures

```
// Declare a new type of struct
struct employee{
  char *name;
  double salary;
}; // Remember the semicolon!

int main() {
  // non-pointer struct syntax
  struct employee boss;
  boss.name = "B. Gates";
  boss.salary = 1000000000;

  // Pointer struct syntax
  struct employee* joe;
  joe = (struct employee*)
        malloc(sizeof(struct employee);
  joe->name = “Joe Schmoe”;
  joe->salary = 50000;
}
```

Stack and Heap Assumptions

Unless otherwise stated, assume:

- The Stack starts at address 10,000 and grows upwards
- The Heap starts at address 50,000 and grows downwards
- The heap will reuse freed memory to satisfy a new request if there is sufficient space
- int and int* consume 4 bytes