CMPUT 350 - C Refresher

1 Hello World

Create file `hello.c` containing this program (copy & paste):

```c
// this program prints "hello world" to standard output
// let the compiler know about standard I/O functions
#include <stdio.h>
int main() {
    printf("hello world\n");
    return 0;
}
```

Then issue

```
gcc -o hello hello.c
```

which generates executable file `./hello` from source file `hello.c`. The program prints ”hello world” when being invoked by issuing `./hello<ENTER>`. `gcc` is the C compiler in the GNU Compiler Collection.

In CMPUT 350 we will be using its cousin g++ for compiling C++ programs. For projects involving StarCraft you will have to install Windows and Visual. C++ on your computer (or in a virtual machine if you use Linux or Mac OS).

2 Important gcc/g++ command line paramters

- -O, -O2, -O3 : optimize executable code at (-O3 fastest)
- -Wall -Wextra -W -Wundef -Wconversion -Wsign-conversion: switch on useful compiler warnings
- -g: generate debug information (for debugger gdb)

To learn more about command line options issue: `man g++` or `man gcc`.

3 C Types and Variables

- basic: `char`, `short`, `int`, `long int`, `float`, `double`
- arrays:

```c
int foo[100];        // defines array foo that contains 100 ints
                     // on the stack, valid indicies are 0.99
int x = foo[42];     // stores array element with index 42 into variable x
```

- pointers:

```c
int x = 5;  // defines int x (on stack)
int *px = &x;    // address of x is assigned to int pointer px
                 // px no points to x
*px = 0;         // px points to x, *px therefore refers to x.
                 // This sets x to 0
                 // & is the address of operator, * is the pointer dereference operator
```

- structures: Store data items together using a common name. Access by component name:

```c
struct X {
    int a, b;           // structure X with components a,b,c,d
    float c, d;
};

struct X x;             // variable definition (x has type X)
x.a = 5;                // assign 5 to structure component a
printf("%d", x.b);      // print x's b component to stdout
```

4 Control Flow

- Conditionals:

```c
if (x > 0) {
    // executed if x > 0
} else if (x < 0) {     // [optional]
    // executed if x < 0
}
else {                  // [optional]
    // executed if x == 0
}
```

- Loops:

```c
while (x > 0) {     // execute as long as x > 0
    --x;            // decrease x by 1
}

int i;
for (i=0; i < n; ++i) {     // execute for i = 0 .. n-1
    printf("%d\n", i);
}

// is equivalent to:
int i = 0;
while (i < n) {
    printf("%d\n", i);
    ++i;
}
```

- Functions: parameters are passed by value (except for arrays, see below)

```c
// function that returns an int, takes int parameter x, and returns x+1
int foo(int x) { return x + 1; }

int y = foo(5); // y = 6 after executing this line

// function that takes int array and number of elements
// as parameter and returns sum of array elements
int sum(int A[], int n) {
    int sum = A[0];
    int i;  // in C++ can be moved inside the for loop
    for (i=1; i < n; ++i) {
        sum += A[i];
    }
    return sum;
}
```

In C, arrays are passed by reference. What is actually passed is just a pointer to the first element. Consequently, array parameters don't know the size of the array. So, the size has to be passed separately.

Equivalent function definition (but harder to understand: is it a pointer we pass or an array?)

```c
int sum(int *A, int n) {
    // ...
}
```

5 Memory Allocation

`malloc` allocates m consecutive bytes on heap (requires `stdlib.h`). Since `malloc` returns a `void*`, it needs to be cast to the appropriate type:

```c
int *p = (int*) malloc(n * sizeof(int));
```

allocates array of n ints on heap and assigns address of first int to `p`. `(int*)` is a type cast that convinces the compiler that the returned value is indeed a pointer to int.

`free(p)` returns memory `p` to the operating system. Using `*p` after this call is a logical error.