COMP1911 22T2 (https://webcms3.cse.unsw.edu.au/COMP1911/22T2) **Code Examples from Lectures on 9-5_pointers** Introduction to Programming (https://webcms3.cse.unsw.edu.au/COMP1911/22T2)

arrayRep.c (https://cgi.cse.unsw.edu.au/~cs1911/22T2/lec/9-5_pointers/code/arrayRep.c)

An example showing how arrays are stored in one chunk of memory

And that the value of an array variable is really just the address of the beginning of the chunk of the array

```
#include <stdio.h>


int main(int argc, char * argv[]){
    //                p
    int nums[] = {1,2,3,4,5};
    int * p = &nums[2];
    int x = 99;
    nums = &nums[2];
    printf("%d %d %d\n",*p,p[0],p[2]);
    p = &x;
    printf("%d %d\n",*p,p[0]);

    return 0;
}
```

goodReturn.c (https://cgi.cse.unsw.edu.au/~cs1911/22T2/lec/9-5_pointers/code/goodReturn.c)

We can return pointers to variables that exist in the calling function (in this case the main function)

```
#include <stdio.h>
#include <math.h>

int * increment (int * n);


int main(int argc, char * argv[]){
    int i = 2;

    //We can nest calls to sqrt
    //because sqrt has input of type double
    //and output of type double
    double answer = sqrt(sqrt(16));
    printf("Answer %lf\n",answer);

    //Ignore the return value and just increment i
    increment(&i);


    //We can nest the calls by using the return
    //value to call increment again

    increment(increment(increment(&i)));

    printf("%d\n",i);
    return 0;
}


//By returning the pointer, we can nest calls
//since the output is the same type as the input
int * increment( int * n){
    *n = *n  + 1;
    // n is a pointer but it is not pointing
    //to a variable defined in this function
    //so it is ok
    return n;
}
```

pointerCompare.c (https://cgi.cse.unsw.edu.au/~cs1911/22T2/lec/9-5_pointers/code/pointerCompare.c)

Simple example illustrating use of pointers with array elements

Shows the different between comparing pointers vs the values they reference.

```
#include <stdio.h>

int main(void) {
                    //p1  p2          p3
    double nums[] = {100, 1.2, 0.9, 100};
    double *p1 = nums;
    double *p2 = &nums[1];
    double *p3 = &nums[3];

    printf("%p %p %p\n",p1, p2, p3 );
    printf("%d %d\n",p1 == p3, p2 > p1 );

    printf("%lf %lf %lf\n",*p1, *p2, *p3 );
    printf("%d %d\n",*p1 == *p3, *p2 > *p1 );


    return 0;
}
```

memoryAddress.c (https://cgi.cse.unsw.edu.au/~cs1911/22T2/lec/9-5_pointers/code/memoryAddress.c)

Demonstrating the address of & operator

Showing where variables are stored in relation to each other

Easier to see memory layout if we use gcc rather than dcc gcc -o memoryAddress memoryAddress.c

./memoryAddress

```
#include <stdio.h>



int main(void) {
    //These variables have all been declared and have addresses in memory
    //These variables will all be located next to each other in memory
    int a[4];
    int x;
    int y;
    double z;
    char c;

    //All  array elements are located next to each other in memory
    printf("Address of a    is %p\n", a);
    printf("Address of a[0] is %p\n",&a[0]);
    printf("Address of a[1] is %p\n",&a[1]);
    printf("Address of a[2] is %p\n",&a[2]);
    printf("Address of a[3] is %p\n",&a[3]);


    printf("Address of x is %p\n",&x);
    printf("Address of y is %p\n",&y);
    printf("Address of z is %p\n",&z);
    printf("Address of c is %p\n",&c);



    return 0;
}
```

pointerDemo.c (https://cgi.cse.unsw.edu.au/~cs1911/22T2/lec/9-5_pointers/code/pointerDemo.c)

Very first pointer example

```c
#include <stdio.h>


int main(void) {

    int x = 10;
    //The variable p is of type int *
    //which we also call int pointer or pointer to an int
    int *p = &x;
    int *p2 = NULL; //NO ADDRESS
    //int *p3 = 99;     //This is illegal. 99 is not an address of an int
    printf("Value   of p is %p\n",p);  //p and &x should be the same address
    printf("Address of x is %p\n",&x);

    printf("Address of p is %p\n",&p);

    printf("x is %d\n",x);   //10

    printf("*p is %d\n",*p); //10
    *p = *p + 1;             //x = x+1

    printf("x is %d\n",x);   //11
    printf("*p is %d\n",*p); //11


    //printf("*p2 is %d\n",*p2);  // NULL POINTER PROBLEM Don't dereference
                                  // a NULL pointer.


    return 0;
}
```

what3.c (https://cgi.cse.unsw.edu.au/~cs1911/22T2/lec/9-5_pointers/code/what3.c)

Try to follow through what this program does and then run to confirm

An example of being able to have a pointer point midway through a string

```c
#include <stdio.h>


int main(int argc, char * argv[]){
    //              p
    char s[] = "I am hungry";
    char *p = s;


    printf("s %s\n",s); //I am hungry
    printf("s:\t%p\t*s:\t%c\n\n",s,*s);//Address of s
                                       //I

    printf("\np %s\n",p);   //I am hungry
    printf("p:\t%p\t*p:\t%c\n\n",p,*p); //Address of s
                                        //I

    p = &s[5];
    printf("\np %s\n",p); //hungry
    printf("p:\t%p\t*p:\t%c\n\n",p,*p); //Address of s[5],
                                        //h

    return 0;
}
```

mystrcmp.c (https://cgi.cse.unsw.edu.au/~cs1911/22T2/lec/9-5_pointers/code/mystrcmp.c)

A demonstration of how to write and test our own version of the string.h strcmp function

```
#include <string.h>
#include <stdio.h>

#define MAX 100

int mystrcmp(char * s1, char * s2);

int main(void) {
    // ant zoo
    // bird ant
    // hello hell
    // \n h
    // "" ""
    //hello helloooo
    char str1[MAX];
    char str2[MAX];

    fgets(str1,MAX,stdin);
    fgets(str2,MAX,stdin);


    int comparison = mystrcmp(str1,str2);

    if(comparison == 0){
        printf("%s == %s\n",str1,str2);
    } else if (comparison < 0){
        printf("%s < %s\n",str1,str2);
    } else if (comparison > 0){
        printf("%s > %s\n",str1,str2);
    }

    return 0;
}
//       i
//s1 Hello\0
//s2 Hellz\0

int mystrcmp(char * s1, char * s2){
   int i = 0;
   while( s1[i] != '\0' && s1[i] == s2[i]){
       i = i + 1;
   }
          //'o'   'z'
   return s1[i] - s2[i];
}
```

passByReference1.c (https://cgi.cse.unsw.edu.au/~cs1911/22T2/lec/9-5_pointers/code/passByReference1.c)

First example of pass by reference with a primitive type

We send the address of n into the function so the function can change the original memory that holds n rather than just receiving a copy of the value that n stores.

```
#include <stdio.h>
#include <stdlib.h>

void increment(int *x);


int main(int argc, char * argv[]){
    int n = 10;

    printf("before increment is %d\n",n);

    //pass in the address of n
    increment(&n);

    printf("after increment is %d\n",n);

    return EXIT_SUCCESS;
}

// x contains the address of the variable n
// so the function can go to that address and
// modify n
void increment(int *x){
   *x = *x + 1;
}
```

replace.c (https://cgi.cse.unsw.edu.au/~cs1911/22T2/lec/9-5_pointers/code/replace.c)

Print lines from file after replacing specified pattern

Bug: What if I run out of room in newString?

We have not considered this!

```
#include <stdio.h>
#include <string.h>
#include <stdlib.h>

#define MAX_LINE 1024
#define MAX_REPLACEMENT_LINE 32768

void replace(char string[], char target[], char replacement[],
             char newString[], int newStringLen);

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

    if (argc < 4) {
        fprintf(stderr,
                "Usage: %s <target> <replacement> <files>\n",
                argv[0]);
        return EXIT_FAILURE;
    }

    char * targetString = argv[1];
    char * replacementString = argv[2];
    FILE *stream = fopen(argv[3], "r");
    if (stream == NULL) {
        perror(argv[3]);
        return EXIT_FAILURE;
    }


    char line[MAX_LINE];
    while (fgets(line, MAX_LINE, stream) != NULL) {
        char changedLine[MAX_REPLACEMENT_LINE];
        replace(line, targetString, replacementString,
                changedLine, MAX_REPLACEMENT_LINE);
        printf("%s", changedLine);
    }
    fclose(stream);
    return EXIT_SUCCESS;
}

// copy string to new_string replacing all instances of target with replacement

void replace(char string[], char target[], char replacement[],
             char newString[], int newStringLen) {

    int targetLen = strlen(target);
    int replacementLen = strlen(replacement);
    int i = 0;
    int newStringIndex = 0;
    while ( string[i] != '\0'){
        if( strncmp(&string[i],target,targetLen) == 0){
            strcpy(&newString[newStringIndex],replacement);
            i = i + targetLen;
            newStringIndex = newStringIndex + replacementLen;
        } else {
            newString[newStringIndex] = string[i];
            i = i + 1;
            newStringIndex = newStringIndex + 1;
        }
    }
    newString[newStringIndex] = '\0';
}
```

passByReference2.c (https://cgi.cse.unsw.edu.au/~cs1911/22T2/lec/9-5_pointers/code/passByReference2.c)

Simple example illustrating use of pointers to return more thatn 1 value from a function

```c
#include <stdio.h>


void powers(double x, double *square, double *cube) ;

int main(void) {
    double s, c;

    powers(42,&s,&c);

    printf("42^2 = %lf\n", s);
    printf("42^3 = %lf\n", c);
    return 0;
}
                      //Address of s  //Address of c
void powers(double x, double *square, double *cube) {
    *square = x*x;
    *cube = x*x*x;
}
```

what1.c (https://cgi.cse.unsw.edu.au/~cs1911/22T2/lec/9-5_pointers/code/what1.c)

Try to follow through what this program does and then run to confirm

```c
#include <stdio.h>

int main(void) {

    int x = 10;
    int y = 99;
    int *p = &y;

    int z = *p * x;

    *p = *p + 1;

    p = &z;

    printf("%d %d %d %d\n",x,y,z,*p);

    return 0;
}
```

what2.c (https://cgi.cse.unsw.edu.au/~cs1911/22T2/lec/9-5_pointers/code/what2.c)

Try to follow through what this program does and then run to confirm

```c
#include <stdio.h>

int main(void) {
                        n
    int nums[6] = {5,6,7,8,9,10};
    int *n;
    n = nums;
    printf("n[0]=%d n[1]=%d n[2]=%d \n", n[0], n[1], n[2]);

    n = &nums[0];
    printf("n[0]=%d n[1]=%d n[2]=%d \n", n[0], n[1], n[2]);

    n = &nums[3];
    printf("n[0]=%d n[1]=%d n[2]=%d \n", n[0], n[1], n[2]);

    //nums = &nums[3]; //ILLEGAL
    return 0;
}
```

badReturn.c (https://cgi.cse.unsw.edu.au/~cs1911/22T2/lec/9-5_pointers/code/badReturn.c)

An example of attempting to return the address of a local variable.

Once f has finished, the memory for the function f is popped from the stack and re-used by other functions.

```c
#include <stdio.h>

int * f (void);

int main(int argc, char * argv[]){
    int * n = f();
    printf("n is %d\n",*n);
    return 0;
}

int * f(void){
    int x = 3;
    //Trying to return the address of a local variable
    //BAD
    return &x;
}
```