Reference for C programming language:
N.K. "C Programming : A modern approach", Norton, 1996.

Today

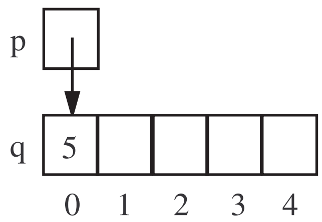
- Pointers and arrays (Ref. King)
- Strings (Ref. King)
- Classes and objects in C++ (Ref. Folk, Zoellick and Riccardi, Section 1.5)
- A useful example to help with your assignment

Pointers and Arrays

1) Pointers can point to array elements and can be used for array processing

```
int a[5], *p, *q;

p = &a[0]; // makes p points to a[0]
*p=5;      // puts 5 in variable pointed by p
```



Pointer Arithmetic

```
p = &a[2]; // p points to a[2]
p+ = 2;    // p points to a[4]
```

Adding 2 to p does not move p 2 bytes, but moves p 2 "int's" ahead since p is a pointer to `int`.

```
q = p - 3;
*q = 10;  // place 10 in a[1]
```

The following program sums the elements of an array using pointers :

```
int a[10],sum,*p;

  ⋮

sum = 0;
for (p = &a[0]; p < &a[10]; p++) {
    sum += *p;
}
```

2) Array name as a pointer
The name of an array can be used as a pointer to the first element in the array.

```
int a[10];

*a = 7;         // stores 7 in a[0]
*(a+1) = 12;    // stores 12 in a[1]
```

The "for-loop" in the previous example can be written as :

```
for (p = a; p < a+N; p++) {
    sum += *p;
}
```

Important : An array can be used as a pointer but it is not possible to assign it a new value.

```
while (*a!=0) {
    a++;   // wrong!!!
}
```

The correct way is

```
p = a;
while (*p!=0) {
    p++;
}
```

3) Array arguments
When passed to a function, an array name is always treated as a pointer.

```
int find_largest(int a[], int n) {
    int i,max;
    max = a[0];
    for (i = 1; i < n; i++) {
        if (a[i] > max)  max = a[i];
    }
    return max;
}

void store_zeros (int a[], int n) {
    int i;
    for (i=0;i<n;i++)
        a[i] = 0;
}
```

To indicate that an array parameter won't be changed (like in find_largest) we can include const in its declaration.

```
int find_largest (const int a[], int n) {
    ...
}
```

No copy of the array constants is done when its passed as argument to a function; only pointers are copied.

The call:

```
    largest = find_largest(b,N);
```

makes a pointer to the first element of `b` (a pointer to `b[0]`) to be assigned to `a`.

String variables

- There is no basic type String in C
- Array of characters may be used as strings

The string “abc” is stored in a array s of four characters as follow :

s	a	b	c	\0

The empty string “ ” is stored as

s	\0

You are allowed to initialize the `char` array with a string literal, but not do an assignment of the string literal to an array.

```
char s[4] = "abc";   //OK

char s[4]
s = "abc";  // Wrong!

s[0] = ‘a’; s[1] = ‘b’; s[2] = ‘c’;
s[3] = ‘\0’;   //Correct
```

In order to move strings around more easily you need to use the C string library :

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

Useful string manipulation functions :

String copy (`strcpy`)

Prototype : `char *strcpy(char *s1, const char *s2);`

```
strcpy(s,"abc");
strcpy(r,s); //Now r contains "abc"
```

If you had done :

```
r = s;
```

Pointer `s` would be copied into `r`, but if `r` was declared as `char r[4]`, the pointer assignment would fail.

Other useful functions :

String concatenation (`strcat`)

```
strcat(str1,str2);
```

Appends `str2` to the end of `str1`.

String comparison (`strcmp`)

```
strcmp(str1,str2);
```

returns value	< 0	if `str1 < str2`
	$= 0$	if `str = str2`
	> 0	if `str1 > str2`

Comparison in lexicography :

```
"abcd" <  "abce"
"abc"  <  "abcd"
```

String length (`strlen`)

`strlen(str)` returns the length of the string, not counting the extra null character \0.

```
int len;
char str1[10];

len strlen("abc");    // len is 3
len = strlen("");     // len is now 0
strcpy(str1, "abc");
len = strlen(str1);   // len is now 3
```

Using Objects in C++

Reference : Folk, Zoellick and Riccardi. Sections 1.5.

- Read the book section fore more details

An example of a very simple C++ class is Person, as given below.

```
class Person
{ public :
    // data members
    char LastName[11], FirstName[11], Address[16];
    char City[16], State[3], ZipCode[10];
    // method
    Person();   // default constructor
};
```

- `LastName`, `FirstName`, ... are members
- Object `p` of class `Person` is declared :

  ```
  Person p;
  ```

- `p.LastName` refers to its `LastName` member.
- Levels of access :

  `public`

  `private`

  `protected`

C++ includes special methods called constructors which guarantee that objects are property initialized.

A constructor has no return type and the same name as the class: `Person()` in the example

There are two ways of having objects created :

by declaration of variable :

```
Person p;  // automatic creation
```

by declaration of pointer + dynamic creation using new operator :

```
Person *p-ptr = new Person;     // dynamic creation
```

Also ok :

```
Person *p-ptr;
...
p-ptr = new Person;
```

In this case, access to members can be done as follow :

```
(*p-ptr).LastName       or
p-ptr -> LastName       (the second is most used)
```

Either object's creation includes the execution of `Person`'s constructor.

- The symbol :: is the scope resolution operator, telling that `Person()` is a member of `Person` class
- Note that inside the member code, the member can be used without the dot(.) operator.
- Every call of a member function has a hidden argument which is a pointer to the object: `this`.
  `this` → `LastName` is the same as `LastName` inside a method's code.

The code for Person constructor is provides as follow :

```
Person::Person()
{  // set each field to an empty string
    LastName[0]=0; FirstName[0]=0; Address[0]=0;
    City[0]=0; State[0]=0; ZipCode[0]=0;
}
```

Example: manipulating fixed length records in C++

The following program will be discussed during this tutorial/lab.. It contains elements/ideas that are useful for assignment#1.

```
// readrec.cpp
#include <fstream.h>
#include <string.h>

#define MAX 100

// a simplified version of Person class
class Person {
   public:
   char LastName[6];
   char FirstName[6];
   char State[3];
   Person();
};

Person::Person() {
   LastName[0]='\0'; FirstName[0]='\0'; State[0]='\0';
}

// Read from stream a fixed length record and places in p
// Fields have sizes: 5, 5 and 2 respectively

int ReadRecPerson(fstream & stream, Person & p) {
   stream.getline(p.LastName,6); // reads 5 characters or default
   stream.clear();               // delimiter '\n'
                                 // when delimiter not found "fail"
                                 // flag is set this clears "fail" flag
   if (strlen(p.LastName)==0) return 0;
   stream.getline(p.FirstName,6); stream.clear();
   stream.getline(p.State,3); stream.clear();
   return 1;
}
```

```
// Write to stream the Data in p
int WriteRecPerson(fstream & stream, Person & p) {
   stream << p.LastName << p.FirstName << p.State << endl;
}

// Read records from "in.txt" and write in "out.txt" in reverse order
// (first record last, last record first)
int main() {
   fstream infile;
   fstream outfile;

   infile.open("in.txt",ios::in);
   outfile.open("out.txt",ios::out);

   Person people[MAX]; int i,n=0;

   if (infile.fail()) {  // if file does not exist, abandon program
      cerr <<"File open failed!\n";
      return 0;
   }

   while ((ReadRecPerson(infile,people[n]) != 0) && (n<MAX))
      n++;

   for (int i=n-1; i>=0 ; i--)
      WriteRecPerson(outfile,people[i]);

   infile.close();
   outfile.close();

   return 1;
}
```

After you understand this, you may look at appendixes: D.5, D.6, D.6, D.8 in order to see how we could do similar reading/writing tasks by "overloading" `operator>>` and `operator<<`.