

Learn C in 15 minutes (if you already know C++)

C is essentially a subset of C++. Here are some differences between the two languages.

- C has no classes or anything to do with classes (e.g.,
 - no constructors
 - no destructors
 - no inheritance
 - no operator overloading
 - no templates

Although there is no concept of classes, there is a `struct`. A struct is like a class which can have data members, but no member functions, and everything is public. (A struct is exactly like a pascal record.) For example:

```
struct Student {
    char firstname[32];
    char lastname[32];
    int ID;
    double GPA;
};
.....
int main()
{
    struct Student Suzy;
    strcpy(Suzy.lastname, "Creamcheese");
    Suzy.ID = 234567;
    ...
}
```

Notice that whenever you declare a structure, you have to precede it with the word `struct`.

- All variables must be declared before the first executable statement in a function or block.

```
{
    struct Student Suzy;
    strcpy(Suzy.lastname, "Creamcheese");
    struct Student Monica; /* wrong */
    int i; /* wrong */
    ...
}
```

Here is how this code would have to have been written in C.

```
{
    struct Student Suzy;
    struct Student Monica;
    int i;
    strcpy(Suzy.lastname, "Creamcheese");
    ...
}
```

- Comments must be enclosed in `/*` and `*/`. The C++ style comment delimiter `//` is not allowed.
- Instead of using `new`, you must use the system call `void* malloc(int n)`
This works like `new` in that it gets new memory from the heap and returns a pointer. The argument `n` is the number of bytes of memory. You have to cast the function to the correct type. Here are some examples;

```
char *cptr;
double *dblptr;
struct Student *stuptr;

cptr = (char *)malloc(100); /* equiv to cptr = new char[100]; */
dblptr = (double *)malloc(sizeof(double) * n);
    /* equiv to dblptr = new double[n]; */
stuptr = (struct Student *)malloc(sizeof(struct Student) * 47);
    /* equiv to stuptr = new Student[47]; */
```

- Input and output are done quite differently in C. You cannot use the `<<` and `>>` operators and `cin` and `cout` are not defined. You must use the library functions `scanf()` and `printf()`. These functions take variable numbers of arguments. The first argument is a format string (`char *`), and the remaining arguments are variables to write from in `printf` or read to in `scanf`.

In the format string for `printf`, all characters are printed as themselves except those following the percent character (`%`). The percent characters indicate that the value of a variable should be printed. The percent sign should be followed immediately by a letter indicating the data type, `d` for integer, `c` for a character, `s` for a character string, and `f` for a float or double. There should be one argument for each `%` in the format string.

Here are some examples of `printf`

```
printf("Hello World\n"); /* prints Hello World followed by a carriage
```

```

return, known in Unix as a newline char */
int x, y;;
char c = 'A';
char *s="Hello World";
x = 17;
printf("The value of x is %d", x); /* prints The value of x is 17 */
y = 430;
printf("The sum of %d and %d is %d\n",x,y,x+y);
/* prints The sum of 17 and 430 is 447 */
printf("The ascii value of %c is %d\n",c,c);
/* prints The ascii value of A is 65 */
printf("%s\n",s); /* prints Hello World */

```

The function `scanf()` reads data from the keyboard, and works the same way.

```

char name[32];
int age;
printf("Please enter your name: ");
scanf("%s",name);
printf("Please enter your age: ");
scanf("%d",&age);

```

Note that you must pass the address of the variable name except in the case of strings for `scanf` variables.

There are also a number of functions which read from standard input (the keyboard by default)

`char *gets(char *buffer)` reads a line into buffer up to a newline character. This is a dangerous function to use because it is easy to overflow the buffer if the line read in is longer than the buffer size.

`int getchar()` reads a single character from standard input. It returns the character.

`int putchar(int c)` writes the character `c` to standard output (the terminal by default)

`int puts(char *s)` writes the string `s` to standard output

- There are no reference variables in function arguments in C. To simulate call by reference, pass the address of a variable to the function. Here is an example:

C program

```
void fctn(int x, int *y)
{
    x = 17;
    *y = 46;
}

int main()
{
    int a = 12;
    int b = 3;
    fctn(a,&b);
    printf("%d %d\n",a,b);
    return 0;
}
```

C++ equivalent

```
void fctn(int x, int& y)
{
    x = 17;
    y = 46;
}

int main()
{
    int a = 12;
    int b = 3;
    fctn(a,b);
    cout << a << ' ' << b << endl;
    return 0;
}
```

Both of these examples print the same output 12 46

- The include file for I/O is `stdio.h`, not `iostream.h`
- The suffix on C source files is `.c`, not `.cpp`. This is important, because many compilers (including Microsoft Visual Studio) invoke a different compiler depending on the suffix of the source code.