

The Australian National University Faculty of Engineering & Information Technology  
Department of Computer Science

COMP2300-2007-02

Notes for Tutorial/Laboratory Session 03: C I/O, Pointers and Structures

No preparation exercise are required for this session. Note however there is submittable work which is due by 10 am Tuesday 20 March (week 5), which will contribute up to 1% of your assessment (in the Tute/Lab mark).

### 1 Tutorial Questions

Work through the following questions in your session. Finish any uncompleted exercises for homework.

1. Consider the declaration `int a[8], b[4];`. How many bytes of memory would be allocated by this declaration on a 32-bit machine? Draw a diagram to illustrate this. Would the assignment statement `b = a;` be legal, and if so describe what effect it would have.
2. **Pointers:** true or false? (and why?)
  - (a) A variable in a C program, regardless of its type, must have an address.
  - (b) A variable of any type can be used to store the address of a variable.
  - (c) The address of a variable is expressed in hexadecimal form; therefore, only an integer type variable can be used to store the address of a variable.
  - (d) A pointer variable of any type can be used to store the address of a `double` variable.
  - (e) Pointer variables are more difficult to use than non-pointer variables because the programmer needs manually to find the addresses of variables to be stored in the pointer variables.
  - (f) The indirection operator `*` may appear only on the left side of an assignment statement.
  - (g) The indirection operator `*` may appear on both sides of an assignment statement.
  - (h) The address of operator `&` may appear only on the left side of an assignment statement.
  - (i) The address of operator `&` may appear on both sides of an assignment statement.
  - (j) For an `int` variable `aa`, its address `&aa` is a constant, not a variable.
  - (k) Given `char aa, *bb;` and noting `sizeof(aa)` is 1 byte, then `sizeof(bb)` is also 1 byte.

3. Consider the following code:

```
#include <stdio.h>
int main(void) {

    char str[80];
```

```

char *p;

printf("Enter a string:\n");
scanf("%80s", str);
p = str;

while(*p) *p++ -= 32;
printf("%s\n", str);

return 0;
}

```

Rewrite it using array notation instead of pointers, i.e. the `p[]` notation. Now state what the program does.

What function from `<ctype.h>` could you use to achieve the same effect as `*p++ -= 32;` ?

4. In the following code:

```

#include <stdio.h>
void funct(int *p);

int main(void) {
    int a[] = {4, 8, 10, 7, 9, -6, 3, 0, 7, 22, 1};

    funct(a+1);
    funct(a+2);
    funct(a+8);

    return 0;
}

void funct(int *p){
    int x = 0, y = 0;
    int *pstart, *pend;
    pstart = p;

    while (*p >= 0){
        x += *p++;
        y++;
    }

    pend = p;
    printf("y=%d x=%d pend-pstart=%d\n", y, x, pend-pstart);
}

```

- What kind of argument is passed to `funct()`?
- What kind of information is returned by `funct()`?
- What information is actually passed to `funct()`?
- What is the purpose of the `while` loop that appears within `funct()`?
- What values are displayed by the `printf()` statement within `funct()`?

## 2 Laboratory Exercises

Execute the command `cp -pr /dept/dcs/comp2300/public/lab3 .` to copy over this lab's C program in your directory.

1. Do tests number 10 and 11 (pointers, and pointers and structures) from the [Interactive Tests](http://www.cs.ntu.edu.au/sit/resources/cprogram/onlinet.htm). (<http://www.cs.ntu.edu.au/sit/resources/cprogram/onlinet.htm>).
2. The programs `string.c` and `funct.c` are from Tutorial Questions 3 and 4 respectively. Compile and run these programs, to verify your answers to those questions.
3. The program `readchars.c` aims to read in from the keyboard two lines that contain two characters on each line. Inspect its code.

It should produce the following, where AB and CD are the lines typed at the keyboard:

```
Enter two characters (without spaces), then press return:
```

```
AB
```

```
First Character A Second Character B
```

```
Enter two MORE characters (without spaces), then press return:
```

```
CD
```

```
Third Character C Fourth Character D
```

However the code does not print this for the third and fourth characters. Why is this? Fix the program so it works correctly.

4. Write a program `readstring.c` that inputs a character string. The program should define the input character string as all entered characters from the first non-white space character to the next white space character. If the character string contains more than 20 characters, it should be truncated to contain just the first 20 entered characters. The program then prints out the string (with a `"%s\n"` format) and terminates. Note that the program prints nothing else.

Example input/output line pairs:

```
MYCHARACTERSTRING PLUS SOME OTHER DATA
MYCHARACTERSTRING
```

```
ANOTHER AND SOMETHING ELSE
ANOTHER
```

```
A_VERY_LONG_STRING_THAT_IS_TRUNCATED_AT_20_CHARACTERS
A_VERY_LONG_STRING_T
```

5. The program `struct.c` defines a structure and initialises it. Add two `printf()` calls to print the components of `me` firstly using `me` and then using `*pme`.

6. Some of the C library functions and many Unix system library functions return pointers to structures. An example of this is the Unix `gettimeofday()` function. (Do `man gettimeofday` to read all about it). This function has the following prototype, that is also defined in the header file `<sys/time.h>`:

```
int gettimeofday(struct timeval *tv, struct timezone *tz);
```

The program `timeloop.c` uses `gettimeofday()` to measure the elapsed time required to execute a small loop. It will print a system error message if the call to `gettimeofday()` fails.

Inspect the code of the program, then compile it and try running it, supplying a largish value for the 1st parameter, e.g. `timeloop 9000000`. You will probably find the time taken is substantially less than one second. From the man page you will see that the second component of the `timeval` structure also contains the number of microseconds.

Before the final `printf()` statement, compute the precise time (accurate to microseconds) that the loop took to execute as a floating point value, and assign it to `loopTime`. Now the program should print this time to 4 decimal places. Note: do not alter any of the `printf()` statements, nor add any others.

Once you are satisfied with the exercises, check your programs' correctness using the command:

```
previewAutoMark lab2 readstring.c timeloop.c
```

When all is working correctly, submit it using:

```
submit comp2300 lab3 readstring.c timeloop.c
```

before **10 am Tuesday 20 March** (the deadline is strict).