COMP1521 Tutorial 02

GCC Preprocessing

If the following program is in a file called `prog.c`:

```
 1. #define LIFE 42
 2. #define VAL  random()%20
 3. #define sq(x) (x*x)
 4. #define woof(y) (LIFE+y)
 5.
 6. int main()
 7. {
 8.     char s[LIFE];
 9.     int  i = woof(5);
10.     i = VAL;
11.     return (sq(i) > LIFE) ? 1 : 0;
12. }
```

then what will be the output of the following command:

```
$ gcc -E prog.c
```

You can ignore the additional directives inserted by the C pre-processor.

Function Scoping

Consider the following C program skeleton:

```
1. int  a;
2. char b[100];
3.
4. int fun1() { int c, d; ... }
5.
6. double e;
7.
8. int fun2() { int f; static int ff; ... fun1() ... }
9.
10. unsigned int g;
11.
12. int main(void) { char h[10]; int i; ... fun2() ... }
```

Now consider what happens during the execution of this program and answer the following:

a. Which variables are accessible from within `main()`?
b. Which variables are accessible from within `fun2()`?
c. Which variables are accessible from within `fun1()`?
d. Which variables are removed when `fun1()` returns?
e. Which variables are removed when `fun2()` returns?
f. How long does the variable `f` exist during program execution?
g. How long does the variable `g` exist during program execution?
h. How long does the variable `ff` exist during program execution?

Consider the following pair of variables

```
int  x;  // a variable located at address 1000 with initial value 0
int *p;  // a variable located at address 2000 with initial value 0
```

If each of the following statements is executed in turn, starting from the above state, show the value of both variables after each statement:

a. `p = &x;`

b. `x = 5;`

c. `*p = 3;`

d. `x = (int)p;`

e. `x = (int)&p;`

f. `p = NULL;`

g. `*p = 1;`

If any of the statements would trigger an error, state what the error would be

Pointers

Show what the following decimal values look like in 8-bit binary, 3-digit octal, and 2-digit hexadecimal:

a. 1
b. 8
c. 10
d. 15
e. 16
f. 100
g. 127
h. 200

How could I write a C program to answer this question?

Decimal, Octal and Hexadecimal

Bitwise Operations

Assume that we have the following 16-bit variables defined and initialised:

```
unsigned short a, b, c;
a = 0x5555;  b = 0xAAAA;  c = 0x0001;
```

What are the values of the following expressions:

a. `a | b` (bitwise OR)
b. `a & b` (bitwise AND)
c. `a ^ b` (bitwise XOR)
d. `a & ~b` (bitwise AND)
e. `c << 6` (left shift)
f. `a >> 4` (right shift)
g. `a & (b << 1)`
h. `b | c`
i. `a & ~c`

Give your answer in hexadecimal, but you might find it easier to convert to binary to work out the solution.

Device Flags

Consider a scenario where we have the following flags controlling access to a device

```
#define READING   0x01
#define WRITING   0x02
#define AS_BYTES  0x04
#define AS_BLOCKS 0x08
#define LOCKED    0x10
```

The flags are contained in an 8-bit register, defined as:

```
unsigned char device;
```

Write C expressions to implement each of the following:

a. mark the device as locked for reading bytes
b. mark the device as locked for writing blocks
c. set the device as locked, leaving other flags unchanged
d. remove the lock on a device, leaving other flags unchanged
e. switch a device between reading and writing, leaving other flags unchanged

Word Reversing

Given the following type definition

```
typedef unsigned int Word;
```

Write a function

```
Word reverseBits(Word w);
```

Which reverses the order of the bits in the variable `w`.

For example: If `w == 0x01234567`, the underlying bit string looks like:

```
0000 0001 0010 0011 0100 0101 0110 0111
```

which, when reversed, looks like:

```
1110 0110 1010 0010 1100 0100 1000 0000
```

which is `0xE6A2C480` in hexadecimal.

Consider a `Stack` data type like the one defined in lectures:

```
1. // Interface to Stack data type
2.
3. #define MAX_STACK 1000
4.
5. typedef char Item;
6.
7. typedef struct _stack {
8.    int  top;
9.    Item items[MAX_STACK];
10. } Stack;
11.
12. void initStack(Stack *s);
13. int  pushStack(Stack *s, Item val);
14. Item popStack(Stack *s);
15. int  isEmptyStack(Stack s);
16. void showStack(Stack s);
```

Some of the functions have a parameter which is defined as `Stack *s`, while others have a `Stack s` parameter.

a. Why might we define the parameters differently like this?

b. Assuming that the stack parameter's name is `s`, how would you refer to the `top` field within the function `initStack()` and within the function `isEmptyStack()`?

c. Are there any disadvantages to the parameter type used by `isEmptyStack()` and `showStack()`?

Stack ADT