C Exam 17.08.23 idan hava

1) int FlipDigitsInNum(int)

Pseudo code:

Check if num < 0.
if so → num = -1 * num., sign = -1

As long as (num != 0)
{
store LSB digit
divide num by base 10
Build new num by multiplying
}
return res;

res variable

LSB * 10 + LSB

int FlipDigitsInNum(int num)
{
int res = 0; //result
int LSB = 0;
int sign = 1;
if(num < 0)
{
sign *= -1;
}

Implementation

DELETED

```
while (num != 0)
{
    lsb = num % 10;          // algo
    lsb /= 10;
    res = res * 10 + lsb;
}

res *= sign;                 // restore sign
return res;
}
// End of FlipDigitsInNum
```

2) Hexadecimal representation → divide to nibbles:

1100 1110 1001 1001 1101 1000 1111 0111

C E 9 9 d 8 f 7

result: 0xCE99d8f7

3) uninitialized variable:

auto local → created during stack-frame creation, contains GARBAGE

Static local → created & allocated with compilation, DATA seg. → value = 0!

global → BSS seg. → value = 0

static global → BSS seg → value = 0

* globals may be stored in "Common" without initialization to Zero if "-fno-common" is used for compilation.

4) char ByteMirror (char byte) // Flip
{
// nibbles
byte = ((byte & 0xf0) >> 4) | ((byte & 0x0f) << 4);
// pairs
byte = ((byte & 0xcc) >> 2) | ((byte & 0x33) << 2));
// singles
byte = ((byte & 0xAA) >> 1) | ((byte & 0x55) << 1);
return Byte;
}

// The above function does not use loops, but Bitwise operations - which are the fastest. Solution is O(1);

5) int FlipBit (int val, unsigned int n)
{
// xoring a bit with 1, effectively flips the bit:
unsigned int mask = 1U;
mask <<= n; // prepare mask
val ^= mask; // flipping
return val;
} // Done with O(1);

// undefined B.: n > 32 (int bits size)

6) Implementation:

```
unsigned int CountSetBits (char byte)
{
    unsigned int mask1 = 1 U;
    unsigned int count = 0 U;
    while (0 != byte)
    {
        Count != byte & mask1;
        byte >>= 1;
    }
    return count;
}                                   // o(n) comp.
```

O(1) Solution by Agbaria:

```
size_t countBitsOn(char num)
{
    num = ((num>>1) & 0x55) + (num & 0x55);
    num = ((num>>2) & 0x33) + (num & 0x33);
    num = ((num>>4) & 0x0f) + (num & 0x0f);
    return (num);
}
```

We can send each byte in the short or double to the function and make the same

or

We can add a line for the short :-

```
num = (num>>8) + num;
```

and for the double :-

```
num = (num>>16) + num;
```

4) size_t is a data type defind in <stddef.h> library. it is Architecture-dependent:

+64 systems: unsigned long long

+32 systems: unsigned long

size_t is used to reresent sizes and indices - it can only be positive, and can store the largets possible value the system supports.

* In contrast, INT max value is reduced because of the MSB bit Representing a signed number, so it is not effective for sizes, indices and maximum values.

8) Char RotateLeft(Char byte, unsigned int nbits)

#define NUM_OF_BITS = 8

```
{
    Char mask1 = 1U;
    Char msb_holder = 0U;
    size_t i = 0;

    While ( i < nbits)
    {
        msb_holder = (byte >> NUM_OF_BITS -1) & mask1;  //store msb

        byte <<= 1;  //shift msb left

        byte = byte | msb_holder;  //bring last msb back from the right.
    ++i;
    }
    return byte;  // O(n)
}
```

O(1) Solution by Agbaria only for n<7:

8) char RotateLeft (char byte, unsigned int nbits)

```
{
    byte = (byte << nbits) | (byte >> (8-nbits));
    return (byte);
}
```

9) I can come up with 2 possible reasons for general different results given by reading the binary file in a different computer: 1. Endianess of system
2. WORD SIZE of System

1. Endianess is a property of a system arch. which determines the way memory is read & accessed with regards to Most & Least significand Bytes of the data.

for example int a = 256 will be stored as follows:

	in ~~Little~~ Big Endian sys.	in ~~Big~~ Little Endian sys.
address 0x00	00000000	00000001
0x08	-0-	-0-
0x0A	-0-	-a-
0x0F	00000001	00000000

LSB is in Higher Address | LSB is in ~~Higher~~ Lower Address

2. In our case, the Binar file contains only 0-100 numbers so they should not exceed more than ~~one~~ 1-byte for each if the Serialization & deserialization functions take into account the need to read the same size of data always.

⇓

So finally only the Endianess of the system may cause the fabrication of the data.

10) Void SwapPointers (int*** a, int*** b)

{ // if *a == *b → undefined.

if (*a != *b)

{

a = **a ^ (b);

b = **a ^ (b);

a = **a ^ (b);

}

}

// the above function subs XOR operation to swap values just the same as done for classic swap of values. This time the values are pointers themselves, so the arguments should be ** of pointers

// parameters are ***params, and can point to pointers.

12) unsigned long GetNFibonacciElement(unsigned int n)

{ size_t prv = 1; size_t nxt = 2;

size_t cur = 1; size_t i =0;

If (n < 2) return current;.

While (i < n – 2)

{

prv = nxt;

nxt += cur;

cur = prv;

}

return cur;

}

11) Strlen:

```
size_t Strlen(const char *str)
{
    size_t i=0;
    assert(NULL != str);   // Should appear in all imps. only once here.
    while('\0' != *str)
    {
        ++i;
        ++str;
    }
    return i;
}
```

StrCmp:

```
int StrCmp(const char *str1, const char *str2)
{ // while any current element not found
    while((*str1 == *str2) && ('\0' != *str1))
    {
        ++str1;
        ++str2;
    }
    return *str1 - *str2;
}
```

Strcat:

```
char *Strcat(char *dest, const char *src)
{
    StrCpy(dest + Strlen(dest), src);
    return dest
}
```

Strcpy:

```
Char *strcpy(char *dest, const char *src)
{
    size_t i=0;
    while ('\0' != *(src+i))  //as long as not end
    {
        *(dest+i) = *(src+i);
        ++i;
    }
    *(dest+i) = EOS;

    return dest;

}
```

Strncpy:

```
Char *strncpy(char *dest, const char *src, size_t num)
{
    size_t i=0;

    while ('\0' != *(src+i) && i<num)
    {
        *(dest+i) = *(src+i);
        ++i;
    }
    while(i < Strlen(dest))
    {
        *(dest+i) = '\0';
        ++i;
    }
    return dest;
}
```

13) void IntToString (int num, char* str) // for base 10

```
{
#define ASCII_offset 48
    size_t i = 0;
    size_t lsb_digit;
    size_t sign = 1;

    if (num < 0)
    {
        sign = -1;
        num *= sign
    }

    while (num != 0)                    //algo
    {
        lsb_digit = num % 10;
        num /= 10;                      //build string
        str[i] = (char) (lsb_digit + ASCII_offset);
        ++i;
    }

    if (sign < 0)
    {
        str[i] = '-';                   // add minus sign
        ++i;
    }
    str[i] = '\0';

    MirrorString (str);                 // helper function
}
```

~~void MirrorString (Str)~~ MirrorString // Helper function

```
void MirrorString (char *str)
{
                                    //undefined B.: NULL STR
    size_t len = strlen (str);
    size_t i = 0;

    while ( i < len/2)
    {
        char temp = str[i]
        str[i] = str[len-i-1];
        str[len-i-1] = temp;
    }
}
```

14) Multiplying by 8 is effectively like multipling by 2^3.
Shifting an unsigned number a single time to left multiplies it by 2^1 for n times: multiplies it by 2^n;

```
Void Mul By 8 (int * num)
{
    int* num_to_mul = *num;

    *num_to_mul <<= 3;
    *num = num_to_mul;
}
```

15) void SWAP (int *a, int *b) ① // using temp

k. {
int c = *a;
*a = *b;
*b = c;
}

ר. void SWAP (int *a, int *b) ② // using xor by value
{
if (a != b)
{
*a = *a ^ *b;
*b = *a ^ *b;
*a = *a ^ *b;
}
}

ב. See Question #10 for another solution using Xoring by address.
Solutions ② & ③ are both O(1)

16) Prints: 8 6 5 5
sizeof pointer — sizeof string* (char*) = 1 — sizeof string — sizeof string

Segmentation fault (s1 is Read only)

Strlen does not count '\0'

17) Val[arr3] same as arr3[val] →

PRints: 3.

18) Static global: used for storing a value to be accessed by all functions in the same c.file.
if initialized → in Data seg.
if not, =0 → " BSS seg

Static local: used within a function / Block, keeps its value outside of its scope.
initialized to 0, Data seg.

Static function: used for helper function or "hiding" functions so they can only be accessed from same c.file. Stored in CODE seg.

Static Library: a library containing .o. files
So if included, Linker can use its implementations and then to the final .out file of the using app.

Static link: Linker can only about globals
~~residing in the same file.~~
libraries that were available during linking process. final .out is created after resolving all unresolved symbols.
in contrast: In dynamic linking the final .out file is created after execution, when all libraries are loaded into memory.

InfinityLabs R&D

19) auto – the default status of a C variable.
"auto local variable"

register – a HW component implemented with transistors, enabling storing data in a binary representation, also refered to as a "storer of data"

Volatile – ~~a variable~~ a qualifier informing the compiler that the variable it is appended to – can change in any time, meaning its not advisable to use it if usage is Re-entrant.

default – a Precondition for all cases determined other than these specified in implementation.

extern – a statement allowing to import variables/functions residing in other files, assuming they are not static.

20) Size of (int)
x32: 4 Byte
x64: 4 Byte

21)

```
struct s    {
int *i;             /*8 bytes */
char c;             /*1 bytes */
short s;            /*2 bytes */
int *j;             /*8 bytes */
/*total: 8*3 = 24 bytes */    }s_ty;
```

```
typedef struct size
        {
        unsigned int a:1;
        unsigned int b:31;
        unsigned int c:1;
        } mystruct;
```

With int b:31, the output is 8.
With int b:1, the output is 4.
With int b:32, the output is 12.

22)

```
u int i:2
u char c:1
u short s:3
int *j
u char x:2
```

```
struct card
        {
         unsigned int i : 2;          /* 4 bytes occupied*/
         unsigned char c : 1;         /* uses i space */
         unsigned short s : 3;        /* uses i space */
         int *j ;                     /* not considered */
         unsigned char x : 2;         /* uses i space */
                                      /*total: 4 bytes */
         }c_ty;
```

23)

```
Union
{
  char array [7];   //
  int i;            //   (x64) → Largest
  short s;          //  2
} jack              // jack is a variable

Size of (jack) =

System:   x64  32
```

7 bytes-< =8 (largest)

8 bytes