EXAM ANSWERS

1.

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

unsigned int binaryToDecimal(char* binary) {
    int len = strlen(binary);
    unsigned int result = 0;
    int i;
    for (i = 0; i < len; i++) {
        if (binary[i] == '1') {
            result = (result << 1) + 1;
        } else if (binary[i] == '0') {
            result = result << 1;
        } else {
            printf("Error: input string contains non-binary characters\n");
            return 0;
        }
    }
    return result;
}

```

```c
int main() {
    char binary[] = "1100110011001100";
    unsigned int decimal = binaryToDecimal(binary);
    printf("Binary: %s\nDecimal: %u\n", binary, decimal);
    return 0;
}
```

The binaryToDecimal function takes a pointer to a string containing a binary integer and returns its decimal value as an unsigned integer. It works by iterating over the characters of the string from left to right, shifting the result to the left by one bit for each '0' encountered and adding 1 before shifting for each '1' encountered. If the string contains a character other than '0' or '1', the function prints an error message and returns 0. In the main function, we define a sample binary string, call binaryToDecimal to convert it to decimal, and print both values to the console.

2.

```c
#include <stdio.h>
#include <string.h>
#include <ctype.h>

int hexToInt(char *hexStr) {
    int result = 0;
    int len = strlen(hexStr);
    int power = 1;

    // Iterate over the string in reverse order
    for (int i = len - 1; i >= 0; i--) {
        char c = toupper(hexStr[i]);
        int digit = 0;

        // Convert the hex digit to an integer
        if (isdigit(c)) {
            digit = c - '0';
        } else if (c >= 'A' && c <= 'F') {
            digit = c - 'A' + 10;
        } else {
            // Invalid character in the string
            return -1;
        }
```

```
57          // Add the digit to the result
58          result += digit * power;
59          power *= 16;
60      }
61
62      return result;
63  }
64
```

This function takes a string hexStr containing a 32-bit hexadecimal integer, and returns its integer value. It first calculates the length of the string, and then iterates over the string in reverse order. For each hex digit in the string, it converts it to an integer value (0-15) and adds it to the result, multiplying by the appropriate power of 16 based on the digit's position in the string. Note that this implementation assumes that the input string contains only valid hexadecimal digits (0-9, A-F, a-f). If the input string contains any other characters, the function returns -1 to indicate an error.

3.

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

// Returns the binary representation of a number as a string
char *int_to_bin(int num) {
    static char bin[33];  // buffer to store binary string
    int i;

    // Convert number to binary string, bit by bit
    for (i = 31; i >= 0; i--) {
        bin[31-i] = ((num >> i) & 1) ? '1' : '0';
    }

    // Add null terminator to end of string
    bin[32] = '\0';

    return bin;
}

// Example usage
int main() {
    int num = 42;
    char *bin_str = int_to_bin(num);
    printf("Binary representation of %d is %s\n", num, bin_str);
    return 0;
}
```

In this implementation, the int_to_bin function takes an integer num and returns a pointer to a static buffer bin containing the binary representation of num. The binary string is constructed by iterating over the bits of the integer, starting with the most significant bit, and adding the appropriate character ('0' or '1') to the string based on the value of the bit. Finally, a null terminator is added to the end of the string. The static buffer is used to avoid having to dynamically allocate memory for the string, but it means that the function is not reentrant (i.e., can't be called safely from multiple threads at the same time).

4.

```c
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

char* intToHex(int num) {
    // calculate maximum length of the hexadecimal string
    int length = snprintf(NULL, 0, "%X", num);
    // allocate memory for the string
    char* hexStr = (char*) malloc(length + 1);
    if (hexStr == NULL) {
        printf("Memory allocation failed.");
        exit(1);
    }
    // convert integer to hexadecimal string
    snprintf(hexStr, length + 1, "%X", num);
    return hexStr;
}

int main() {
    int num = 305441741; // example integer
    char* hexStr = intToHex(num);
    printf("%s\n", hexStr);
    free(hexStr); // free memory allocated for the string
    return 0;
}
```

This function first calculates the maximum length of the hexadecimal string using snprintf() and

then allocates memory for the string using malloc(). It then converts the integer to a hexadecimal string using snprintf() again and returns the string. Finally, the memory allocated for the string is freed using free().