**BTP100 Midterm Review Answers (in red)**

1. How many bits are in the following:
   1. A byte. 8 bits
   2. A word. 4 bytes or 32 bits
2. What are the following in hexadecimal:
   1. 8 0x8
   2. 16 0x10
   3. 24 0x18
   4. 32 0x20
3. Name the four most common type of the C language for performing arithmetic calculations.

char, int, float, double

1. Size specifiers adjust the size of the int and double types. Name three size specifiers.

short, long, long long

1. What type qualifier specifies that a variable is unmodifiable.

Const

1. Which variable type has a wider range of values, int or float? Why?

float. Floats use an exponent therefore have a wider range of values. The limits on a float and double depend on the execution environment.

1. What is casting?

The C language supports conversions from one type to another. To convert the type of an operand, we precede the operand with the target type enclosed within parentheses. We call such an expression a cast.

1. What is a structured program?

A structured program consists of sets of simple constructs, each of which has one entry point and one exit point.  Any programmer may replace one construct with an upgraded construct without affecting the other constructs in the program or introducing errors ("bugs").

1. Professionals in the field of human-computer interaction confirm that layout and arrangement affects comfort and accessibility.  Poorly laid out code frustrates and promotes misreadings. Name five layout tools at our disposal.

Indentation, line length, braces, spaces, and comments.

1. What are magic numbers and how do we avoid them?

We refer to values that appear out of nowhere in program code as *magic numbers*.  These may be mathematical constants, standard rates or default values.  We avoid magic numbers by identifying them with symbolic names and using those names throughout the code.  We set their value in either of two ways:

* using an unmodifiable variable (ie const int NUM=3;
* using a macro directive (ie #define NUM 3)

1. Compare and contrast syntactic errors with semantic errors.

Syntactic errors are errors that break the rules of the programming language whereas semantic errors are errors that fail to implement the intent and meaning of the program designer.

1. Compare and contrast black box tests with white box tests.

Black box tests are data driven. We run the executable and treat it as a black box where all internal logic has been hidden from view. External factors alone determine the success or failure of our tests. We test against the specifications. Our tests are input-output driven.

The complementary test to black box tests is a white box test. White box testing is logic driven. We treat the program as a glass box with all internal logic visible. Each white box test is path-oriented. In white box testing, we execute each possible path through the code at least once. The number of paths may be too large to test. To reduce this number and still cover all paths through the code at least once, we prepare flow graphs.

1. Compare and contrast a structure with an object of that structure.

A type describes how to interpret the information stored in a region of memory. In the C language, a type may be a primitive type or a derived type. A derived type is a collection of other types. The declaration of a derived type in the C language takes the form

struct Tag {

//... declarations here

};

where the keyword struct identifies a derived type or structure.

When we define an object of a structure, we allocate memory for that object. Our definition takes the form

struct Tag identifier;

where Tag is the name of the structure and identifier is the name of the object.

1. Compare and contrast a while loop with a do-while loop.

The while construct executes its sequence as long as the test condition is true. The do while construct executes its sequence at least once and continues executing it as long as the test condition is true.

1. Compare and contrast parallel arrays with structures.

A convenient way to store tabular information is through two parallel arrays. The index holds the key while the arrays hold the values. The arrays are parallel because the elements at the same index hold data that are related to the same entity.

A convenient alternative to parallel arrays for storing tabular information is an array of structures. The index holds the key, while the members of the structure hold the data.

1. Convert the following while-loop into a for-loop (see Loops.c):

int i = 0;

while (i < 100) {

printf("Hello\n");

++i;

}

for (i = 0; i < 100; ++i) {

printf("Hello\n");

}

1. Convert the following for-loop into a while-loop (see Loops.c):

for (i = 100; i > 0; --i) {

printf("Hello\n");

}

i = 100;

while (i > 0) {

printf("Hello\n");

--i;

}

1. Fill in the blanks in the following code (see bmi.c).

//bmi.c - Midterm review question for filling out the code

// This program asks the user for his/her name, height in meters

// and weight in kgs. It then calculates the body mass index (bmi)

// according to the formula: bmi = height/(weight\*weight)

// If the bmi is higher than 30.0, that person is considered obese.

// To end the program, enter 0 for name. The program then reports

// how many users there are and how many are obese.

A sample run is as follows:

Enter the user's name: Luke

Enter Luke's height in meters: 1.80

Enter Luke's weight in kg: 90

Enter the user's name: Lorelei

Enter Lorelei's height in meters: 1.70

Enter Lorelei's weight in kg: 80

Enter the user's name: Mike

Enter Mike's height in meters: 1.80

Enter Mike's weight in kg: 110

Mike is obese

Enter the user's name: Ming

Enter Ming's height in meters: 1.60

Enter Ming's weight in kg: 70

Enter the user's name: 0

There is/are 4 users and 1 is/are obese

//bmi.c - Midterm review question for filling out the code

// This program asks the user for his/her name, height in meters

// and weight in kgs. It then calculates the body mass index (bmi)

// according to the formula: bmi = height/(weight\*weight)

// If the bmi is higher than 30.0, that person is considered obese.

// To end the program, enter 0 for name. The program then reports

// how many users there are and how many are obese.

#define \_CRT\_SECURE\_NO\_WARNINGS

#include <stdio.h>

#define MAX\_STR 32

int main(void)

{

int numUsers = 0;

int numObeseUsers = 0;

char name[MAX\_STR];

float height, weight, bmi;

int done = 0;

do {

printf("Enter the user's name: ");

scanf("%s", name);

if (name[0] == '0') { //end the program if '0' is entered

done = 1;

}

else {

//prompt the user for the height and weight

printf("Enter %s's height in meters: ", name);

scanf("%f", &height);

printf("Enter %s's weight in kg: ", name);

scanf("%f", &weight);

//bmi equals the height divided by the weight squared

bmi = weight / (height \* height);

if (bmi > 30) {//test for obesity, bmi greater than 30

printf("%s is obese\n", name);

++numObeseUsers;//increment the obesity count

}

++numUsers;//increment the user count

printf("\n"); }

} while (done == 0);

printf("There is/are %d users and %d is/are obese\n", numUsers, numObeseUsers);

return 0;

}

1. Evaluate the following (see Evaluation.c):

const double e = 2.71828;

printf("The constant e is %d %.2lf\n", (int)e, e);

The constant e is 2 2.72

1. Evaluate the following (see Evaluation.c):

const int x = 16;

const int y = 5;

printf("x/y is %d\n", x / y);

x/y is 3

1. Evaluate the following (see Evaluation.c):

const int z = 16;

const int w = 4;

printf("z/w is %.3f\n", (float)z / w);

z/w is 4.000

1. Evaluate the following (see Evaluation.c):

int num1 = 92, num2 = 94;

int result = (num2 < num1) ? (num2 - num1) \* 3 : (num2 - num1) \* 2;

printf("result is: %d\n", result);

result is: 4

1. Evaluate the following (see Evaluation.c):

int num3 = 2, num4 = 4;

int res = (num4 > num3) ? num4 - num3 \* 3 : num4 - num3 \* 2;

printf("res is: %d\n", res);

res is: -2

1. Evaluate the following (see Evaluation.c):

const char college[] = "Seneca College";

printf("%c%c%c\n", college[3], college[4], college[5]);

eca

1. What is the exact output of the following program (see Walkthrough.c)?
   1. Record your work using the walkthrough table provided on the next page for the below code.
   2. Put your output answer in the “Screen Output” box on the next page.

1//Walkthrough.c - Midterm review on walk-through tables

2

3//Miguel Watler

4//miguel.watler@senecacollege.ca

5//1234567890

6//Section XXY

7#define \_CRT\_SECURE\_NO\_WARNINGS

8#include <stdio.h>

9#define NUM 5

10#define PASS\_GRADE 50

11

12int main(void)

13{

14 //Midterm marks for five courses

15 int midTerm[NUM] = { 23, 34, 41, 33, 40 };

16 //Final exam marks for five courses

17 int finalExam[NUM] = { 26, 30, 40, 12, 45 };

18 int finalMark[NUM] = { 0 };

19 int numFailed = 0;

20 int numPassed = 0;

21

22 for (int i = 0; i < NUM; ++i) {

23 finalMark[i] = midTerm[i] + finalExam[i];

24 if (finalMark[i] < PASS\_GRADE) {

25 ++numFailed;

26 }

27 else {

28 ++numPassed;

29 }

30 }

31

32 printf("This student has passed %d course(s) and failed %d course(s).\n",

numPassed, numFailed);

33

34 return 0;

35}

**Walkthrough Table**

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Line # | i | midterm[i] | finalExam[i] | finalMark[i] | numFailed | numPassed |
| 22 | 0 | 23 | 26 | 0 | 0 | 0 |
| 23 | 0 | 23 | 26 | 49 | 0 | 0 |
| 25 | 0 | 23 | 26 | 49 | 1 | 0 |
| 22 | 1 | 34 | 30 | 0 | 1 | 0 |
| 23 | 1 | 34 | 30 | 64 | 1 | 0 |
| 28 | 1 | 34 | 30 | 64 | 1 | 1 |
| 22 | 2 | 41 | 40 | 0 | 1 | 1 |
| 23 | 2 | 41 | 40 | 81 | 1 | 1 |
| 28 | 2 | 41 | 40 | 81 | 1 | 2 |
| 22 | 3 | 33 | 12 | 0 | 1 | 2 |
| 23 | 3 | 33 | 12 | 45 | 1 | 2 |
| 25 | 3 | 33 | 12 | 45 | 2 | 2 |
| 22 | 4 | 40 | 45 | 0 | 2 | 2 |
| 23 | 4 | 40 | 45 | 85 | 2 | 2 |
| 28 | 4 | 40 | 45 | 85 | 2 | 3 |
|  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |

**Screen Output:**

This student has passed 3 course(s) and failed 2 course(s).

1. A Complete Program (see nhlStats.c).

Create a structure called PlayerInfo for the Stanley Cup champions St. Louis Blues. The structure should contain the following:

* 1. Name (up to 31 characters plus the NULL character)
  2. Goals (integer)
  3. Assists (integer)
  4. Points (integer)
  5. Games played (integer)
  6. Points per game (float)

Prompt the user to enter the last name, the number of goals scored, the number of assists, and the number of games played for four St. Louis Blues. The program will then calculate the number of points for that player according to the formula: **Points = Goals + Assists**, and then calculate the points per game for that player according to the formula: **PointsPerGame = Points/Games**. The statistics for some players from last season are:

// Name Games Goals Assists Points

//O'Reilly 82 28 49 77

//Tarasenko 76 33 35 68

//Schenn 72 17 37 54

//Perron 57 23 23 46

//Pietrangelo 71 13 28 41

//Bozak 72 13 25 38

Once the data has been entered, the program will display each player’s last name, the number of games played, the number of points, and the points per game to three decimal places. Then the program will display the name of the player with the maximum points per game.

Format all your printf’s and scanf’s according to the sample run on the following page.

Enter the player's last name: O'Reilly

Enter the number of goals: 28

Enter the number of assists: 49

Enter the number of games: 82

Enter the player's last name: Tarasenko

Enter the number of goals: 33

Enter the number of assists: 35

Enter the number of games: 76

Enter the player's last name: Schenn

Enter the number of goals: 17

Enter the number of assists: 37

Enter the number of games: 72

Enter the player's last name: Perron

Enter the number of goals: 23

Enter the number of assists: 23

Enter the number of games: 57

O'Reilly played 82 games with 77 points and 0.939 points per game

Tarasenko played 76 games with 68 points and 0.895 points per game

Schenn played 72 games with 54 points and 0.750 points per game

Perron played 57 games with 46 points and 0.807 points per game

O'Reilly has the most points per game with 0.939

#define \_CRT\_SECURE\_NO\_WARNINGS

#include <stdio.h>

#define SIZE 4

#define MAX\_STR 32

struct PlayerInfo {

char name[MAX\_STR];

int goals;

int assists;

int points;

int games;

float pointsPerGame;

};

int main(void) {

struct PlayerInfo player[SIZE] = { 0 };

float maxPointsPerGame = 0.0;

int indexMaxPointsPerGame = -1;

for (int i = 0; i < SIZE; ++i) {

printf("Enter the player's last name: ");

scanf("%s", player[i].name);

printf("Enter the number of goals: ");

scanf("%d", &player[i].goals);

printf("Enter the number of assists: ");

scanf("%d", &player[i].assists);

printf("Enter the number of games: ");

scanf("%d", &player[i].games);

player[i].points = player[i].goals + player[i].assists;

player[i].pointsPerGame = (float)player[i].points / player[i].games;

printf("\n");

}

for (int i = 0; i < SIZE; ++i) {

printf("%s played %d games with %d points and %.3f points per game\n",

player[i].name, player[i].games, player[i].points, player[i].pointsPerGame);

maxPointsPerGame = player[0].pointsPerGame;

indexMaxPointsPerGame = 0;

if (player[i].pointsPerGame > maxPointsPerGame) {

maxPointsPerGame = player[i].pointsPerGame;

indexMaxPointsPerGame = i;

}

}

printf("\n");

printf("%s has the most points per game with %.3lf\n",

player[indexMaxPointsPerGame].name, player[indexMaxPointsPerGame].pointsPerGame);

return 0;

}