John Akujobi

Chapter 6 Review problems

CSC 150

**Q 1**

Write a function called **letter\_grade** that has a type int input parameter called points and returns through an output parameter **gradep** the appropriate letter grade using a straight scale (90–100 is an A, 80–89 is a B, and so on). Return through a second output parameter ( just\_missedp ) an indication of whether the student just missed the next higher grade (true for 89, 79, and so on).

void letter\_grade (int points, char \*gradep, int \*just\_missedp)

{

//In the straight scale, 90–100 is an A, 80–89 is a B, 70–79 is a C, 60–69 is a D, and 0–59 is an F.

//Use the else statement to check for invalid input.

\*just\_missedp = 0;

//checks for A grade (90-100)

if (points >= 90 && points <= 100)

{

\*gradep = 'A';

}

//checks for B grade (80 - 89)

else if (points >= 80 && points <= 89)

{

\*gradep = 'B';

}

//checks for C grade (70 - 79)

else if (points >= 70 && points <= 79)

{

\*gradep = 'C';

}

//checks for D grade (60 - 69)

else if (points >= 60 && points <= 69)

{

\*gradep = 'D';

}

//checks for F grade (0 - 59)

else if (points >= 0 && points <= 59)

{

\*gradep = 'F';

}

//Catches grades above 100

//Accepts the grade (for cases of extra points)

//But also informs the user about it

else if (points > 100)

{

\*gradep = 'A';

printf ("This student has a grade above 100 (Points = %d)\\n", points);

}

//Using this to catch any other values that are not in the range of 0-100

else

{

\*gradep = 'X';

printf("\\nOops! Invalid input.\\nPlease enter a number between 0 and 100.\\n");

exit (101);

}

//checks if the student just missed the next higher grade

//Didn't use (points % 10 == 9) in the case the student had 109 points brought by extra-points or 99 points

if (points == 89 || points == 79 || points == 69 || points == 59)

{

\*just\_missedp = 1;

}

return;

}

**Q 2**

Why would you choose to write a function that computes a single numeric or character value as a non void function that returns a result through a return statement rather than to write a void function with an output parameter?

**The values returned from a non void function can be used for more operations in the calling function but this can’t easily be done with a void function with an output operator.**

**Q 3**

Explain the allocation of memory cells when a function is called. What is stored in the function data area for an input parameter? Answer the same question for an output parameter.

**Memory cells are allocated for all the local variables and parameters of a function when it is called. This memory cells are temporarily given to the function while its being executed but are deallocated after the function has finished executing.**

**In the function data area for an input parameter, local copies are stored of the actual data from the caller.**

**As for output parameter, the function data area stores the value that will returned back to the caller/the calling function.**

### Q 4

Which of the functions in the following program outline can call the function grumpy? All function prototypes and declarations are shown; only executable statements are omitted.

**The main function [int main (void)] can call the function grumpy**

**Q 5**

Sketch the data areas of functions main and silly as they appear immediately before the return from the first call to silly in Quick-Check Exercise 8.

Diagram, timeline

Description automatically generated

void silly(int \*x);

int main(void)

{

int x, y;

x = 10; y = 11; //x is allocated as 10, while y is 11

silly(&x); //calls silly to act on x

silly(&y); //calls silly to act on y

/\* values here \*/

. . .

// before returning silly

//variable x contains 10

//variable y contains 11

}

void silly(int \*x)

{

int y; //creates integer y

y = \*x + 2; //y becomes the sum of \*x and 2

\*x = 2 \* \*x; //\*x is incremented by 2

//before returning

}

**Q 6**

Present arguments against these statements:

* It is foolish to use function subprograms because a program written with functions has many more lines than the same program written without functions.

True, for some solutions, it would be very inefficient to unnecessary to use functional subprograms as it would lead to unnecessarily large code files because of the large amounts of code needed.

For example, a program that calculates the area of a rectangle or prints Hello World. Using functions in these programs that do 1, 2 or 3 tasks will be much of an unneeded overkill.

However, aside small programs, C is also used to build very robust applications such as:

* + Windows XP, Unix and Google Chrome Operating systems
  + Mozzilla Firefox browser
  + MySQL Database Management system
  + Adobe Photoshop and Illustrator
  + Git version control system

These programs can carry out hundreds, if not ten thousands of tasks, especially the operating systems. Managing this complexity is made easier and more efficient with function subprograms.

* + Easier to build the features: For example, Photoshop has subprograms to read files, writing into various file formats, add contrast, resize images, calculating vector shapes and several more. These features can be written as their subprograms, allowing them call one another, exchange, processes data, and be activated only when needed. A big contribution to this is that using function subprograms, less code because they are reusable and reduces unnecessary repetition of code.
  + It makes it easier to read and maintain: Windows XP had 45 million lines of code. It would be mercilessly laborious to maintain or debug if all the code were mashed together into a single lump of a program. Using function subprograms, maintainability is made a lot easier and efficient.
  + Increases scalability: Git, for example, has been upgraded and released in more than 300 versions. Using function subprograms makes it possible and easier to add new, remove features without breaking a lot of other features.

To sum up, it would be a sort of overkill to use function subprograms for many small programs.

However, it becomes very necessary to use it for larger and more complex programs -which make up most real-life solutions-. Utilizing function subprograms would make it easier to build features, maintain and scale.

* b. The use of function subprograms leads to more errors because of mistakes in using argument lists.

It is possible to make some mistakes while using argument lists in function subprograms.

However, the use of function subprograms will greatly reduce errors overall in the long run. It breaks the programs into portions that are easier to debug, test and maintain. It also reduces the overall amount of code because is allows reusability of code and reduces repeating the same code. This is especially true for large and complex programs carrying out multiple tasks.