

Thinking Out Loud

While I was walking to the office this morning, I chatted briefly with an individual who was coming out of a garage. After mentioning the change in the weather (cooler than before) and how the current weather was lovely for sleeping, the person concluded the conversation with:

"Have a good one!"

Questions:

- How can a multiplicative unit be "good"?
- Why single out an individual number for being "good"?
 - Why not "Have a good two?" or some other number?
 - What about zero or negatives?
 - To avoid favoritism, should one state, "Have a good number?"
- Should this statement be restricted to integers? What about fractions or real numbers or complex numbers?

Announcements

Thursday Extras: CS Commons on Thursdays @ 4:00 pm

this week: Off-Campus Study for CS Majors: Why, How, Where?

Office hours:

Wednesday: 1:00-3:00 pm

Thursday: 1:00-3:00 pm

Friday: canceled

Sunday: 2:00-4:00 pm

Reminder: Supplemental Problem 2 due on Monday, but
no penalty if submitted by 5:00 pm on Tuesday

Today: in class: characters, strings, scanf

Friday: in class: Uninterpretable Dance Project

Reflections on for and while statements

Characters, strings, scanf

questions

clicker questions

Claim: Any for statement can be translated into code using a while loop.

- A. True always
- B. True usually (but not always)
- C. True in some cases
- D. Never True

Consider the basic syntax with for statements

```
for (initialization; condition; update(s))  
    {  
        loop body  
    }
```

This translates directly to

```
initialization  
while (condition)  
    {  
        loop body  
        update(s)  
    }
```

Claim: Any while statement can be translated into code using a for loop.

- A. True always
- B. True usually (but not always)
- C. True in some cases
- D. Never True

Consider the basic syntax with while statements

```
while (condition)
{
    loop body
}
```

This translates directly to

```
for ( ; condition; )
{
    loop body
}
```

what does a while statement highlight?

what does a for statement highlight?

Notes on Supplemental Problem 2

Basis computation for paying off a loan balance, with a given payment and annual interest rate

suppose

- `oldBalance` is the amount owed at the start of the month
- `annualRate` is the annual interest rate (as a percentage)
- `monthlyRate` is the monthly interest rate (as a decimal)
- `newBalance` is the amount owed at the end of the month
- `interestInMonth` is the amount of interest charged in a month
- `payment` is the amount paid by the customer each month

Then the basic monthly computation is

`monthlyRate = annualRate / 1200.`

`interestInMonth = oldBalance * monthlyRate`

`newBalance = oldBalance + interestInMonth - payment`

This computation proceeds each month, except in the last month,

`oldBalance + interestInMonth`

may be less than the payment, so the amount paid the last month may be lower than the usual monthly payment

Notes on Supplemental Problem 2

For loans of N months and monthly rate, the usual formula to compute the payment is

$$\text{payment} = \text{loanAmount} * \text{monthlyRate} / (1.0 - (1 + \text{monthlyRate})^{(-N)})$$

Computing this formula can be split into two parts:

- Compute $(1 + \text{monthlyRate})^{(-N)}$
- Plug this power into the full formula

Problem asks you to do the two formulae as two functions, where computing a value to a negative integral power will use a loop. (C has a power function, but it is quite inefficient in this context.)

Overall, the above formula will give an appropriate minimum payment to pay the loan off in N months.

With that amount, one can compute (and print, if desired) the monthly balance from the start of the loan until it is paid off. In the process, one can also determine the total amount of interest paid.

Notes on Supplemental Problem 2

But, what happens if one pays more than the minimum?

In this case, the monthly balance will go down faster, so the total interest charged will be less. (Many banks do not charge a penalty, for early payment, and fewer months will yield less interest paid altogether.)

The problem asks you to make three computations:

- one with the minimum amount paid each month
- two with additional payments (either at the start or every month).

With these computations, you can compute how much additional interest is paid in some cases over others.

Additional notes:

From now on:

- no global variables (although global constants allowed)
- all functions/procedures must state pre- and post-conditions

On our lab machines, the code segment

```
char ch1 = 'a';  
char ch2 = 'z';  
printf (" char = %c, code = %d\n", ch1, ch1);  
printf (" char = %c, code = %d\n", ch2, ch2);
```

prints the following output

```
char = a, code = 97  
char = z, code = 122
```

Which of the following segments prints the alphabet (in lower case) on all computers (internationally) running C?

A. `for (char ch = 'a'; ch <= 'z'; ch++)`

`printf ("%c", ch);`

B. `for (int i = 'a'; i <= 'z'; i++)`

`printf ("%c", i);`

C. `for (int i = 97; i <= 122; i++)`

`printf ("%c", i);`

A. A, B, and C

B. A and B

C. A and C

D. B and C

E. Something else

Suppose a is declared as an array of 10 characters:

```
char a [10]
```

and suppose the beginning address for this block of characters is 100. Which of the follow is true?

- A. a stores the address 100; that is, $a == 100$
- B. the address of a is 100; that is, $\&a == 100$
- C. the address of a[0] is 100; that is $\&a[0] == 100$
- D. exactly two of the above
- E. all of the above

Suppose a program contains the following declarations:

```
char a [10];  
char * b = a;
```

Suppose further that the beginning address for the a array is 100, and supposed b has address 200. Which of the follow is true?

- A. `b == 100`
- B. `&b == 200`
- C. `&b[0] == 100`
- D. exactly two of the above
- E. all of the above

Consider the following declarations and initializations

```
/* declaration and initialization */  
int arr1 [5] = {3, 1, 4};      // line 1  
int arr2 [ ] = {2, 7, 1, 8, 2}; // line 2  
int arr3 [ ] = arr2;           // line 3  
int arr4 [5];                  // line 4  
arr4 = arr2;                   // line 5  
int * arr5;                    // line 6  
arr5 = arr1;                   // line 7  
int arr6 [5] = arr5;           // line 8
```

In 4 votes, which lines compile (e.g., no errors)?

Vote 1	Vote 2	Vote 3	Vote 4
1. 1 and 2	1. 3 and 4	1. 5 and 6	1. 7 and 8
2. 1 not 2	2. 3 not 4	2. 5 not 6	2. 7 not 8
3. 2 not 1	3. 4 not 3	3. 6 not 5	3. 8 not 7
4. neither 1 nor 2	4. neither 3 nor 4	4. neither 5 nor 6	4. neither 7 nor 8

A character usually requires 1 byte of storage.

- `sizeof` (from `stdlib.h`) indicates the storage required.
- `strlen` (from `string.h`) indicates the string's logical length.

What is printed by the following code?

```
char str [ ] = "computer";  
printf ("size %d, strlen %d, str[2] %c\n",  
        (int) sizeof (str), (int) strlen (str), str[2]);
```

A. size 8, strlen 8, str[2] o

B. size 8, strlen 8, str[2] m

C. size 9, strlen 9, str[2] o

or

size 9, strlen 9, str[2] m

D. size 9, strlen 8, str[2] o

E. size 9, strlen 8, str[2] m

Consider the following code segment, and suppose the str2 array is stored immediately **before** the str1 array in main memory:

```
char str1 [16] = " is exciting!!!";
char str2 [16] = {'c', 'o', 'm', 'p', 'u', 't', 'e', 'r',
                  ' ', 's', 'c', 'i', 'e', 'n', 'c', 'e'};
printf ("A-str2: %s, str1: %s\n", str2, str1);
str1[0] = 0;
printf ("B-str2: %s, str1: %s\n", str2, str1);
```

What is printed?

- A. A-str2: computer science, str1: is exciting!!!
B-str2: computer science, str1: is exciting!!!
- B. A-str2: computer science, str1: is exciting!!!
B-str2: computer science, str1:
- C. A-str2: computer science is exciting!!!, str1: is exciting!!!
B-str2: computer science, str1: ??????
- D. A-str2: computer science is exciting!!!, str1: is exciting!!!
B-str2: computer science, str1:
- E. none of the above

Suppose input will contain a name in the first 40 characters of a line, followed by a number.

```
char line [101]; //leaving room for null character
char name [41];
int number;
```

Two approaches are proposed reading a line

1. `scanf ("%41s %d", name, &number);`
2. `fgets (line, 101, stdin);`
 `strncpy (name, line, 40);`
 `name[40] = 0;`
 `number = atoi (line+40);`

Which of these approaches works reliably?

- A. Approaches 1 and 2
- B. Approach 1, not 2
- C. Approach 2, not 1
- D. Neither 1 nor 2

Suppose you want to read an integer at the start of one line, skip anything else on that line, and then read a double at the start of the next line.

```
int one;  
double two;  
#define max 100  
char line [max+1];
```

Two approaches are proposed:

1. `scanf ("%d", &one);`
 `while (getchar () != '\n') ;`
 `scanf ("%lf", &two);`
2. `fgets (line, max+1, stdin);`
 `one = atoi (line);`
 `while (line[strlen(line)-1] != '\n')`
 `fgets (line, max+1, stdin);`
 `fgets (line, max+1, stdin);`
 `two = atof (line);`

Which of these approaches works reliably?

- A. Approaches 1 and 2
- B. Approach 1, not 2
- C. Approach 2, not 1
- D. Neither 1 nor 2