Name: _____ CS 23200

Date: _____ More Pointer Practice

Strings and pointers

If valid, what does each code snippet print?

1.
```
char *pString = “Hello.”;
printf(pString);
strncpy(pString, “Goodbye.”, strlen(“Hello.”));
printf(pString);
```

2.
```
char pString[200] = “Hello.”;
printf(pString);
strncpy(pString, “Goodbye.”, 200);
pString[199] = ‘\0’;
printf(pString);
```

3.
```
char pString[200];
char pStringTwo[200];
pString = “Hello.”;
pStringTwo = pString;
```

4.
```
char pStringTwo[200] = “Hello.”;
char *pString = pStringTwo;
strncpy(pStringTwo, “Goodbye.”, 200);
pStringTwo[199] = ‘\0’;
printf(pString);
printf(pStringTwo);
```

Pointers with const

Are these snippets valid?

1.
```
void swap(const int *pOne, const int *pTwo){
    int temp = *pOne;
    *pOne = *pTwo;
    *pTwo = temp;
}
```

2.
```
char* findSpace(const char* str){
    while(*str != ‘\0’ && *str != ‘ ‘)
        str++; /* same as "str = &(str[1]);" */
    return str;
}
```

3.
```
void doubleString(const char* str){
    const int n = strlen(str);
    int i;
    for(i=0; i<n; i++){
        str[i + n] = str[i];
    }
    str[2 * n] = ‘\0’;
}
```

Pointers to Pointers, Multi-dimensional Arrays, etc.

Determine if the following code snippets are valid

1.
```
char **str;
str = “Hello.”;
```

2.
```
char **str[2];
str[0] = “Hello.”;
str[1] = “Goodbye.”;
```

3.
```
int *nums[3];
nums[0] = 7;
nums[1] = 19;
nums[2] = 25;
```

4.
```
int i;
int **nums = malloc(sizeof(int) * 10);
for(i=0; i<10; i++){
    nums[i] = malloc(sizeof(int) * 5);
}
nums[9][2] = 17;
```

5.
```
char *names[] = {“Smith”, “Jones”, “Thompson”};
names[0][0] = ‘s’;
names[1][0] = ‘j’;
names[2][0] = ‘t’;
```

4.
```
void doubleString(const char* str){
    const int n = strlen(str);
    const char* pSrc;
    char* pDest;

    for(pSrc = str, pDest = str + n; pSrc < str + n;
            pSrc++, pDest++){
        *pDest = *pSrc;
    }
    *pDest = '\0';
}
```

5.
```
void doubleString(char* str){
    const int n = strlen(str);
    const char* pSrc;
    char* pDest;

    for(pSrc = str, pDest = str + n; pSrc < str + n;
            pSrc++, pDest++){
        *pDest = *pSrc;
    }
    *pDest = '\0';
}
```

6.

```
int nums[10][5];
int i,j;

for(i=0; i<5; i++){
     for(j=0; j<10; j++){
          nums[i][j] = 0;
     }
}
```

7.

```
double **matrix;
int row;
for(row=0; row<10; row++){
     matrix[row] = malloc(sizeof(double) * 5);
}
matrix[9][4] = -1.0;
```

8.

```
char** names;
names = malloc(sizeof(char*) * 3);
names[0] = “Smith”;
names[1] = “Jones”;
names[2] = “Thompson”;
```

9.

```
const int numEvents = 4;
const int numContestants[4]
     = {5, 7, 6, 10};
double *times[numEvents];
int eventIndex;

for(eventIndex=0; eventIndex < numEvents;
     eventIndex++){
     int j;

     times[eventIndex] = malloc(sizeof(double) *
          numContestants[eventIndex]);
     for(j=0; j<numContestants[eventIndex]; j++){
          times[eventIndex][j] = 0.0;
     }
}
```

Command-line Arguments

Fill in the following function so that joinedStr will contain the concatenation of the command line arguments that follow the program name. If the command line arguments' total length exceeds MAX_LENGTH, store the first MAX_LENGTH characters of the concatenated string in joinedStr.

```
int main(int argc, char* argv[]){
    const int MAX_LENGTH = 500;
    char joinedStr[MAX_LENGTH + 1];

    /* fill in here */









    printf("The concatenated string is %s\n.", joinedStr);
    return 0;
}
```