Computers in Engineering
Pseudocode and C Language Review

Pseudocode

- Pseudocode is an artificial and informal language that helps you develop algorithms.
- Pseudocode is similar to everyday English; it is convenient and user friendly although it is not an actual computer programming language.
- Pseudocode programs are not executed on computers. Rather, they merely help you "think out" a program before attempting to write it in a programming language such as C.
- Pseudocode consists purely of characters, so you may conveniently type pseudocode programs into a computer using an editor program.

Guide for Pseudocode

1. State your name
2. State the date
3. State the name of the subroutine.
4. Give a brief description of the function of the subroutine/program.
5. State the names of the input variables, their types, and give a brief description for each.
6. Each variable should be listed and the purpose of each should be briefly mentioned at the beginning of the pseudocode program.
7. Give a list of instructions with enough detail that someone can translate the pseudocode into a computer language such as C, C++, Java, etc. Note, your pseudocode should paraphrase your algorithm and not look identical to C code.
8. Your pseudocode should be indented just like when you write your program. You should use \{ or if/else/endif (for/endfor, while/endwhile, etc.) syntax to denote the start and end of blocks of statements.

Example Pseudocode

```plaintext
1. Programmer: Gary E. Christensen
2. Date: 8/3/06
3. Name: print_matrix
4. Function: Print matrix nicely to screen.
5. Input: a[] = single subscripted int array of size arow * acol = matrix to be printed.
6. arow = int = number of rows of matrix a.
7. acol = int = number of columns of matrix a.
8. MAXSIZE = global constant that specifies the maximum size of the array.
10. Algorithm:

   // Check for error conditions
   if (arow <= 0) or (acol <= 0) then print error message and return endif
   if (arow * acol > MAXSIZE) then print error message and return endif

   // Print the matrix
   k = 0
   for i = 0 to arow - 1
      print bar '|' right justified
      for j = 0 to acol - 1
         print a[k] right justified
      k = k + 1
   endfor
   print bar '|' followed by a newline
endfor
```

Documentation

- The main program and each subroutine must be documented.

```plaintext
    /**********************************************************
    *                                                            *
    * Programmer: <Your name goes here>                        *
    * Date: <Today’s date>                                      *
    * Name: <function name goes here>                          *
    * Function: <Description of what the function does goes here> *
    * Algorithm: <Describe how the function works>             *
    * Input: <Name each input variable to the function on a separate line and give a description of what it is> *
    * Output: <Name each output variable on a separate line and give a description of what it is> *
    */
```
/* Programmer: Gary Christensen  
* Date: 2/17/04  
* Name: quicksort  
* Function: This function sorts an array of integers from  
* smallest to largest value.  
* Algorithm: This subroutine sorts a list of integers using  
* recursion. The first element of the list is moved to  
* it's final position in the list and a sublist of numbers  
* less than this number and a sublist of numbers greater  
* than this number is created. Each sublist is then  
* sorted by passing it to quicksort again. The  
* termination condition for the recursion are as  
* follows: A one element list is sorted by definition and  
* is returned.  
* Input: ListOfInts - a pointer to an array of integers to be  
* sorted  
* size - the number of integers in the list to be sorted.  
* Output: ListOfInts - a pointer to an array of sorted integers */

```c
if (score >= 60)
    printf("You Passed
");
else
    printf("You Failed
");
```

Logical Expressions

```
((x <= 5)&&(x >= 0)) and (y != 5) are examples of logical expressions.
```

Relational Operators

```
== Equality (don't confuse with the assignment operator, =)
!= not equal
> greater than
< less than
>= greater than or equal
<= less than or equal
```

Compound Conditions

```
if ((score == 100)
    printf("Your grade is an A+\n");
else if (score >= 95)
    printf("Your grade is an A-\n");
else if (score >= 90)
    printf("Your grade is an A\n");
else if (score >= 85)
    printf("Your grade is lower than an A\n");
```

The Switch Statement

```
switch(score)
```

If statement

```
scanf("%d",&x);
if ((x < 0)||(x>5)){
    printf("Please enter an integer between 0 and 5.\n");
    printf("Try again\n");
    scanf("%d",&x);
}
```

Logical Operators:

```
&& Logical And
|| Logical Or
! Not (complement)
```

If-else statement

```
if (score == 100)
    printf("Your grade is an A+\n");
else if (score >= 95)
    printf("Your grade is an A-\n");
else if (score >= 90)
    printf("Your grade is an A\n");
else if (score >= 85)
    printf("Your grade is lower than an A\n");
```

Multiple-selection structure

```
- Every constant integral value tested
- Different actions for each value
```

Consists of

```
case labels and default case
```

Logical OR Operator

```
Logical AND Operator
Logical NOT Equal Operator
```
Example program to count number of A, B, and C grades

```c
/* Counting A, B, and C grades */
#include <stdio.h>
int main() {
    char grade;
    int aCount = 0, bCount = 0, cCount = 0;
    printf("Enter the letter grades A, B, or C. ");
    printf("Enter the 'return' character to end.\n");
    scanf("%c", &grade);
    (continued on next slide)
```

```c
while (grade != '\n') {
    switch (grade) {
        case 'a':
        case 'A': ++aCount;
        break; /* denotes end of this case */
        case 'B': ++bCount;
        break;
        case 'C': ++cCount;
        break;
        default: /* catch all other characters */
            printf("Incorrect input.\n");
    } /* end of switch */
    scanf("%c", &grade);
} /* end of while */
/* PRINT TOTALS HERE */
printf("Total of %d A's, %d B's %d C's
", aCount, bCount, cCount);
return 0;
} /* end main */
```

More about the switch Construct

- case labels must evaluate to an integer constant
- All of the labels must be unique (but can be in any order)
- Statement-list can contain zero or more statements
- Control passes to next label unless there is a break statement.
- break exits the enclosing switch

C Loop constructs

- Permit an action to be repeated multiple times
- Three loop constructs
  - while
  - do/while
  - for
- Example (pseudo-code):
  While there are more homework problems to do:
  work next problem and cross it off the list
  endwhile

While Loop Example

- Problem: Find the first power of 2 larger than 1000
- Pseudo-code:
  Initialize value to 2
  while the value is less than 1000:
    Multiply the value by two
  endwhile

```c
product = 2;
while (product <= 1000) {
    product = product * 2;
}
```

While Loop Example Flowchart
Class Average Example

- Problem statement: A class of ten students took a quiz. The grades (integers in the range 0 to 10) for this quiz are available to you. Determine the class average on the quiz.
- Class average = (sum of grades / total students)
- Main algorithm:
  - input each of the grades
  - perform the averaging calculation
  - print the result

Pseudo-code Algorithm

Set total to zero
Set grade counter to one
while (grade counter is less than or equal to ten):
  Input the next grade
  Add the grade into the total
  Add one to the grade counter
endwhile
Set the class average to the total divided by ten
Print the class average
Note: This is an example of a counter-controlled loop (loop is executed a fixed number of times, controlled by a counter)

C Program For the Example

```c
/* average program, counter controlled repetition */
#include <stdio.h>

int main() {
    int counter, grade, total, average;
    total = 0;
    counter = 1;
    while (counter <= 10) {
        printf("Enter grade: ");
        scanf("%d", &grade);
        total = total + grade;
        counter = counter + 1;
    }
    average = total / 10;
    printf("Class average is %d
", average);
    return 0;
}
```

Sentinel-controlled loops

- Sentinel value: a special input value to indicate the end of data entry
- Also called a flag value or signal value
- The user must enter the sentinel value to indicate that all data items have been entered
- Used in cases of indefinite repetition
- Number of data items to be processed is not known before the loop begins executing

Class Average Example Using a Sentinel (Pseudo-code)

- Initialize total to zero
- Initialize counter to zero
- Input the first grade
- while (the user has not yet entered the sentinel):
  - Add this grade into the running total
  - Add one to the grade counter
- Input the next grade (possibly the sentinel)
- endwhile
- If the counter is not equal to zero:
  - Set the average to the total divided by the counter
  - Print the average
- else
  - Print "No grades were entered"
endif

C Program For the Example Using a Sentinel (Pseudo-code)

```c
/* average program, sentinel-controlled repetition */
#include <stdio.h>

int main() {
    double average = 0.0;
    int counter = 0;
    int grade = 0;
    total = 0;
    while (grade != -1) {
        printf("Enter grade, -1 to end: ");
        scanf("%d", &grade);
        total = total + grade;
        counter = counter + 1;
    }
    average = ((double) total) / counter;
    printf("Class average is %.2f
", average);
    return 0;
}
```
The for Loop Construct

- Handles some of the counter-controlled repetition details
- Example:
  ```
  for (counter = 0; counter < 10; counter++) {
    printf("%d\n", counter);
  }
  ```
- for loop has three parts:
  - initializer
  - condition
  - update
- Any for loop could be re-written as a while loop (counter-controlled repetition).

for loop versus while loop

```c
for (expression_1; expression_2; expression_3) {
  statement;
}
```

is the same as:

```c
while (expression_1) {
  statement;
  expression_3;
}
```

More for Loop Examples

How many times does each loop run?
1. for (j = 1; j <= 100; j++)
2. for (j = 100; j >= 1; j--)
3. for (j = 7; j <= 77; j += 7)
4. for (j = 20; j >= 2; j -= 2)
5. for (j = 2; j <= 20; j += 3)
6. for (j = 99; j >= 0; j -= 11)

do/while Loop Construct

- Tests loop-continuation at the end of loop body
- Syntax:
  ```
  do {
  statement1;
  statement2;
  } while (condition);
  ```
- Example: What does this print?
  ```c
  int counter = 1;
  do {
    printf("%d", counter);
    counter++;
  } while (counter <= 10); /*Note semicolon*/
  ```

do/while Versus while Loop

```c
#define NUM 5
main(){
  int i, a[NUM]={4, 2, 7, 3, 9};
  /* Print out the elements of array a[] */
  for(i=0; i<NUM; i++) {
    printf("a[%d]=%d\n", i, a[i]);
  }
  /* Read in integers into the array a[] */
  i=0;
  while (i<NUM) {
    scanf("%d", &a[i]);
    i++;
  }
  /* Print out elements of array a[] */
  i=0;
  do {
    printf("a[%d]=%d\n", i, a[i]);
    i++;
  }while(i<NUM);
}
```

for & while statements

```c
#define NUM 5
main(){
  int i, a[NUM]={4, 2, 7, 3, 9};
  /* Print out the elements of array a[] */
  for(i=0; i<NUM; i++) {
    printf("a[%d]=%d\n", i, a[i]);
  }
  /* Read in integers into the array a[] */
  i=0;
  while (i<NUM) {
    scanf("%d", &a[i]);
    i++;
  }
  /* Print out elements of array a[] */
  i=0;
  do {
    printf("a[%d]=%d\n", i, a[i]);
    i++;
  }while(i<NUM);
}
```