2035 E2 Cheat Sheet

by theninja111 via cheatography.com/18862/cs/1982/

Random Range

rand() % 4 = 0~3

rand() % 11+1 = 1~11

Bubble Sort

7 23 5 78 22

5 7 23 22 78

5 7 22 23 78

5 7 22 23 78

if (list[walker] < list[walker - 1]) {

temp = list[walker]; / *exchange elements* /

list[walker] = list[walker - 1];

list[walker - 1] = temp; }

Initialize 4x5 Array

for(i = 0; i < 4; i ++) { table[i][0] = i * 20;

for(j = 1; j < 5; j ++) { table[i][j] = table[i][j - 1] + 1; }; };

MIPS Subroutine

(2) (14 points) The following MIPS subroutine computes the average of all the odd numbers in an integer array starting at location labeled `Array`. Registers are used as follows:

$1	Array index	$5	Running sum of odd numbers; output average
$2	Size of the array in bytes	$7	Running count of odd numbers
$3	Current array element	$8	Predicate register

Complete the routine by adding MIPS code to preserve registers before the jal by pushing them on the stack and to restore them after the subroutine call.

OddAvg:	addi $1, $0, 0	# init Array index
	addi $2, $0, 400	# init Array size
	addi $5, $0, 0	# init Array index
	addi $7, $0, 0	# init Array index
Loop:	lw $3, Array($1)	# load in current element
	addi $29, $29, -12	**# adjust SP: make room for 3 words**
	sw $1, 0($29)	**# push $1 (preserve on stack)**
	sw $2, 4($29)	**# push $2**
	sw $31, 8($29)	**# push return address**
	jal CountOdd	# in: $3, $5, $7; out: $5, $7
	lw $1, 0($29)	**# pop $1 (restore from stack)**
	lw $2, 4($29)	**# pop $2**
	lw $31, 8($29)	**# pop return address**
	addi $29, $29, 12	**# readjust SP: deallocate 3 words**
	addi $1, $1, 4	# inc Array index
	bne $1, $2, Loop	# if end not reached, loop
	div $5, $7	# odd running sum / odd count
	mflo $5	# put odd number avg in $5
	jr $31	# return to caller

Function Stack

int Foo (int a, int b) { int x; int y = 10;

x = ay+b; return x;}

Complete AF

Foo:	**add $30, $29, $0**	**# set base of FP**
	addi $29, $29, -8	**# alloc 2 ints: x, y**
	addi $1, $0, 10	**# initialize y**
	sw $1, -8($30)	**# push init val of y**

Recursive to Iteration

long factorial(int n) { if (n == 0) {

return 1; }

else { return (n * factorial (n-1)); }; }

long factorial (int n) { int i; long fact;

if (n == 0) return 1; else {

for (i = 1, fact = 1; i <= n; i ++) {

fact = fact * i; } return fact; } }

Copy Text

int main(void) {

FILE +fp1, +fp2; int c;

if (!(fp1 = fopen("aaa.dat", "r"))) {

printf("could not open file for reading. \n"); exit; }

if (!(fp2 = fopen("bbb.dat", "w"))) {

printf("could not open file for writing. \n"); exit;}

while ((c = fgetc(fp1)) != EOF) { fputc(c, fp2); }

fclose(fp1); fclose(fp2); return 0; }

Insertion Sort

7 23 5 78 22

7 23 5 78 22

5 7 23 78 22

5 7 23 78 22

if (temp < list[walker]) { list[walker + 1] = list[walker]; walker --; }

else { located = TRUE; };

AF Variables

(1) (20 points) Consider the following C code fragment. Foo is called by main.

```
int main() {
    float       n = 3.14;
    int         a = 5, b[] = {8, 19, 60}, c;
    float       *p = &n;
    <more code here>
    c = Foo(n, p, a, b, &a);
    <more code here>
}

int Foo (float x, float *y, int z, int *v, int *w) {
    <Foo's code goes here>
}
```

For each statement from Foo below, list the resulting value. If the result is an address, just list "address". For return statements, list the returned value. Also determine if it affects Foo's activation frame variables, main's activation frame variables, or both. Consider each statement **independently** and **non-cumulatively** (i.e., as if it is the only statement in Foo).

statement in Foo	result (*assigned value*)	modify Foo's AF variables		modify main's AF variables	
X += 2.0;	**5.14**	**Yes**	No	Yes	**No**
y++;	**address**	**Yes**	No	Yes	**No**
Z += *w;	**10**	**Yes**	No	Yes	**No**
V[1] += 1;	**20**	Yes	**No**	**Yes**	No
*w = v[2] + 1;	**61**	Yes	**No**	**Yes**	No

Complete AF Cont.

(2) (10pts) Compute the "x = ay+b;" statement by accessing a, y, and b from the stack. Be sure to update locals on the stack.

	lw $3, 4($30)	**# $3: a**
	lw $2, 8($30)	**# $2: b**
	lw $1, -8(30)	**# $1: y (optional, already in $1)**
	mult $3, $1	**# ay**
	mflo $3	**# $3: ay**
	add $3, $3, $2	**# $3: ay+b**
	sw $3, -4($30)	**# update x = ax+b**

(3) (6pts) Finish the implementation of Foo by implementing its return to its caller.

	lw $3, -4($30)	**# $3: x (optional, already in $3)**
	sw $3, 0($30)	**# store x in return value slot**
	addi $29, $30, $0	**# pop off locals**
	jr $31	

Cheatographer

theninja111
cheatography.com/theninja111/

Dates

This cheat sheet has not been published yet and was last updated on 28th April, 2014.