# COMP 2401
## Test #2

**[out of 50 marks]**

1. [4 marks]
   **Answer**:
   -- parameter values
   -- local variables
   -- return address in calling function
   **Marking**:
   -- 2 marks for parameter values
   -- 2 marks for local variables
   -- Bonus: 1 mark for return address

2. [4 marks]
   **Answer**:
   -- dynamically allocated memory
   **Marking**:
   -- 4 marks, all or nothing

3. [4 marks]
   **Answer**:
   -- recompiles only the files that have changed since the last compile
   -- one compilation command rather than several
   **Marking**:
   -- 2 marks for **each** of the above

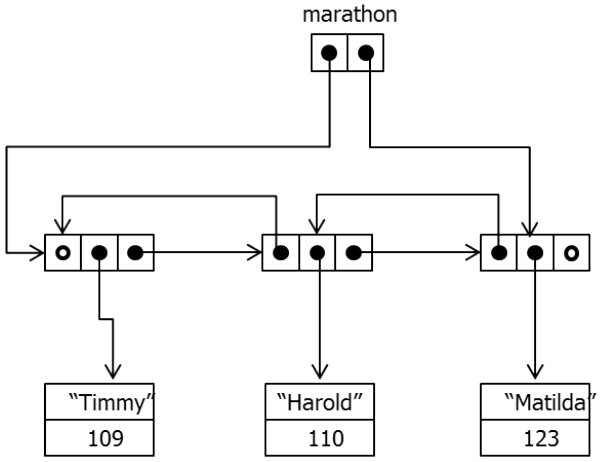
4. [8 marks]

```
void  foo ( int **p )
{
   *p  =  ( int  * )  malloc ( sizeof ( int ) ) ;
   **p =  20 ;
}
int main()
{
  int * x = NULL;
  foo ( &x ) ;
  printf("%d\n", *x); free(x);
  return 0;
}
```

   **Answer**:
   -- see code above
   **Marking**:
   -- 2 marks for making foo parameter a double pointer
   -- 1 mark for dereferencing p in foo's first statement
   -- 1 mark for dereferencing p twice in foo's second statement
   -- 2 marks for passing address of x to foo from main
   -- 2 marks for freeing x after printf

5. [30 marks]

a. [7 marks]
**Answer**:




**Marking:**
-- 1 mark for correct pointer to the head
-- 1 mark for correct pointer to the tail
-- 1 mark for first node's previous set to zero
-- 1 mark for last node's next set to zero
-- 1 mark for correct next pointers (both of them)
-- 1 mark for correct prev pointers (both of them)
-- 1 mark for correct pointers to data, in correct order

b. [23 marks] **Note**: this question is actually worth 24 marks; the extra mark is a bonus

```
void addRunner(RunnerType *runner, ListType *list)
{
  NodeType *newNode, *currNode;

// 2 marks for node allocation
//  -- 1 mark for malloc of correct size
//  -- 1 mark for typecasting to NodeType*
  newNode = (NodeType *) malloc(sizeof(NodeType));

// 2 marks for initializing values
//  -- 1 mark for initializing data
//  -- 1 mark for initializing prev and next
// Note:  if node is incorrectly dereferenced, deduct 1 out of 2
  newNode->data = runner;
  newNode->prev = NULL;
  newNode->next = NULL;

  currNode = list->head;
```

```
// 2 marks for correct looping over existing list
// 2 marks for correct testing for finishTime
  while (currNode != NULL) {
    if (newNode->data->finishTime < currNode->data->finishTime)
      break;
    currNode = currNode->next;
  }

// 4 marks for empty list case and correct pointers changes
// -- 2 marks for setting head
// -- 2 marks for setting tail
  if (currNode == NULL) {
    if (list->head == NULL) {  // empty list
      list->head = newNode;
      list->tail = newNode;
    }
// 4 marks for last position case and correct pointers changes
// -- 1 mark for setting tail's next to new node
// -- 1 mark for setting new node's prev to tail
// -- 2 marks for setting list tail to new node
    else {                     // add to last position
      list->tail->next = newNode;
      newNode->prev    = list->tail;
      list->tail       = newNode;
    }
  }
  else {
// 4 marks for first position case and correct pointers changes
//    1 mark for setting new node's next to head
//    1 mark for setting head's prev to new node
//    2 marks for setting list head to new node
    if (currNode == list->head) {  // add to first position
      newNode->next    = list->head;
      list->head->prev = newNode;
      list->head       = newNode;
    }
// 4 marks for mid-position case and correct pointers changes
//    1 mark for setting new node's prev to curr node's prev
//    1 mark for setting curr/new node's prev's next to new node
//    1 mark for setting new node's next to curr node
//    1 mark for setting curr node's prev to new node
    else {                         // add to mid position
      newNode->prev       = currNode->prev;
      newNode->prev->next = newNode;
      newNode->next       = currNode;
      currNode->prev      = newNode;
    }
  }
}
```