1. Fill the blanks in the definitions below with a phrase from the following list:

dynamic memory allocation, class, typedef, array, object, struct, stack, linked list, new, address operator, dereference.

new: an operator used for dynamic memory allocation.

array: a collection of values of the same type, where each member of the collection can be accessed using an integer index.

class: a programmer-defined data type that includes both data and functions that operate on the data.

linked list: a collection of nodes that are connected by pointers.

typedef: a keyword used to create aliases for type names.

2. Circle T (True) or F (False) as applicable for the following statements.

F Dynamic memory allocation is done during compilation.

T new operator returns a memory address pointing to the allocated memory.

F Elements of a linked list can be accessed using pointer arithmetic.

T The size of a dynamic array is determined at runtime.

T A pointer is a memory address.
3. Consider the following code segment:

```c
char a[10] = { 'H', 'e', 'l', 'l', 'o', '\0'};
char *p1 = a;
char *p2 = &a[2];
```

(a) What is the value of *a ? H
(b) What is the value of *(a+2) ? l
(c) What is the value of *(p1+1) ? e
(d) What is the value of *p2 ? l
(e) What is the value of *(p2-1) ? e

4. Consider the following code segment:

```c
int a[3][3] = {
    { 1, 2, 3 },
    { 4, 5, 6 },
    { 7, 8, 9 }
};

int *pa[3] = { a[0], a[1], a[2] };
int *p = a[0];
```

(a) What is the value of a[0][2] ? 3
(b) What is the value of **a ? 1
(c) What is the value of *(*a+1)+1 ? 5
(d) What is the value of *pa[1] ? 4
(e) What is the value of p[1] ? 2
5. Consider the following code segment:

```c
int m[5][4] = { {10,20,30,40}, {31,41,51,61}, {52,62,72,82},
                {73,93,103,113}, {94,114,134,154} } ;
int *p ;
int *q ;
int *r ;
p = m[0] ;
q = m[1] ;
r = &m[2][2] ;
```

(a) What is the value of *(p+1) ? 20 
(b) What is the value of *(q+2) ? 51 
(c) What is the value of *q+2 ? 33 
(d) What is the value of *r ? 72 
(e) What is the value of *(r-1) ? 62 

6. Write the result of evaluating the expressions listed below after all the statements in the following code segment are executed. Write a question mark ‘?’ if the contents are not defined.

```c
struct Numbers {
    int first;
    int second;
    int rest[4];
};
Numbers myNumbers, *p;
p = &myNumbers;
myNumbers.first = 3;
p->second = 1;
myNumbers.rest[myNumbers.first] = 16;
p->rest[myNumbers.second] = 8;

/* Evaluate the expressions here */
```
7. Consider the following data structure and declarations:

```c
struct Account {
    int number;
    int balance;
    int past[12];
    int minpast;
    int maxpast;
};
struct Account myAccount, *pAccount = &myAccount;
int i;
```

(a) Write an assignment statement that sets the `number` field of `myAccount` to 160 using the `myAccount` variable.

```c
myAccount.number = 160;
```

(b) Write an assignment statement that sets the `balance` field of `myAccount` to 1000 using the `pAccount` pointer.

```c
pAccount->balance = 1000;
```

(c) Write a loop that finds the minimum and maximum values among all elements of the `past` field of `myAccount`, and assigns those values to the `minpast` and `maxpast` fields, respectively.

```c
pAccount->minpast = pAccount->maxpast = pAccount->past[0];
for (i = 1; i < 12; i++) {
    if (pAccount->minpast > pAccount->past[i])
        pAccount->minpast = pAccount->past[i];
    else if (pAccount->maxpast < pAccount->past[i])
        pAccount->maxpast = pAccount->past[i];
}
8. Consider the following data structure:

```c
struct Node {
    int data;
    Node *next;
};
```

and the following code:

```c
Node *n1, *n2;
n1 = new Node;
n2 = new Node;
n1->data = 5;
n2->data = 10;
n1->next = n2;
n2->next = n1;
n2->next->next->data = 7;
```

Write the values of the following expressions after the code above is executed.

<table>
<thead>
<tr>
<th>EXPRESSION</th>
<th>VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>n1-&gt;next-&gt;data</td>
<td>7</td>
</tr>
<tr>
<td>n1-&gt;next-&gt;next-&gt;data</td>
<td>5</td>
</tr>
<tr>
<td>n2-&gt;next-&gt;data</td>
<td>5</td>
</tr>
<tr>
<td>n2-&gt;next-&gt;next-&gt;data</td>
<td>7</td>
</tr>
<tr>
<td>n1-&gt;data + n2-&gt;data</td>
<td>12</td>
</tr>
</tbody>
</table>

9. Consider the following data structure used for implementing a linked list:

```c
struct Node {
    int data;
    Node *next;
};
Node *head;
```

Without declaring another variable, write a code segment that allocates three nodes for the linked list where the head points to the first node. The data for the first node should be assigned the value 10, the data for the second node should be assigned the value 20, the data for the third node should be assigned the value 30.

```c
head = new Node;
```
head->data = 10;
head->next = new Node;
head->next->data = 20;
head->next->next = new Node;
head->next->next->data = 30;
head->next->next->next = NULL;
10. Consider the following data structure used for implementing a linked list:

```c
struct Node {
    int data;
    Node *next;
};
```

Implement the function that takes a pointer to the first node of a linked list and an integer value as arguments. It returns -1 if the input value is not equal to the data value of any item in the list. Otherwise, it returns the location of an item in the list whose data value is equal to the input value. Assume that the location of the first item in the list is 0.

```c
int searchList(Node *head, int value) {
    Node *current;
    int position = 0;

    current = head;
    while (current != NULL) {
        if (current->data == value)
            return position;
        else {
            current = current->next;
            position++;
        }
    }

    return -1;
}
```