Array

Allows data of the same type to be kept under a single variable name.The number of elements is specified when defining.

Identification Examples:

Example 1:
```
int array1 [12];
```

Example 2:
```
char array2 [20];
```

Assign elements to arrays examples:

Example 1:
```
int array1[5] = {1,2,3 ,4,5};
```

Example 2:
```
int array2[5] = {6,7,8}; //last two elements is 0
```

Example 3:
```
int array3[5], i;
for (i = 0; i<5; i++){
array3[i] = i;
printf ("%d \n", array3 [i]);}
```

Multidimensional arrays examples:

Example 1:
```
int array[ 3][2] = {{1,2},{3,4},{5,6}};
int i,j;
for(i = 0;i<3; i++){
for(j=0; j<2; j++){
printf ("%d \t", array[ i][ j]);}
printf ("\n ");}
```

Example 2:
```
int array[ 3][ 2][2] = {1,2,3 ,4, 5,6 ,7, 8,9 ,10 ,11 ,12};
int i,j,k;
for(i = 0; i<3; i++){
for(j = 0; j<2; j++){
for(k = 0; k<2; k++){
printf ("%d \t", array[ i][ j][k]);}
printf ("\n ");}
printf ("\n ");}
```

Stacks

It is a data structure in which the data is kept in a linear way and addition and subtraction are made from the top point. Last in first out rule is valid.

Stack With Array:
```
int Stack[10];
int top = -1;
void Push(int a){
if(top == 9){
printf ("No room in the stack. \n");}
else{
Stack[ ++top] = a;
printf ("%d added into the stack. \n", a);}}
int pop(){
```

Stacks (cont)

```
if (top<0){
printf ("No data in the stack. \n");
return -1;}
else{
int a = Stack[ top--];
printf ("%d removed from the stack. \n", a);
return a;}}`
int peek(){
if (top<0){
printf ("No data in the stack. \n");
return -1;}
else{
printf ("%d is at the top of the stack. \n", Stack[ top]);
return Stack[ top];}}
```

Stack With Linked List:

```
struct Node {
int data;
struct Node* next;};
struct Node* top = NULL;
void Push(int a) {
struct Node* t = (struct Node*) malloc (si zeo f(s truct Node));
t->data = a;
if(top == NULL){
top = t;
top->next = NULL;}
else {
t->next = top;
top = t;}
printf ("%d added into the stack. \n", a);}
int Pop() {
if(top == NULL) {
printf ("No data in the stack. \n");
return -1;}
else {
struct Node* t = top;
int a = t-> data;
top = top->next;
printf ("%d removed from the stack. \n", a);
free(t);
return a;}}
int Peek() {
```

Stacks (cont)

```
if(top == NULL) {
printf ("No data in the stack. \n");
return -1;}
else {
printf ("%d is at the top of the stack. \n", top->d ata);
return top->d ata;}}
```

Linked List

They are structures in which objects of the same type are stored in a linear order and interconnected. The objects in the linked list are called nodes, and nodes are connected to each other by pointers that point to the next node. There is also a head pointer that points to the beginning of the list.Nodes consist of two parts.

Singly linked list: Navigation in the list is forward only.

Doubly linked list: Navigation in the list is forward and back.

Circular linked list: The next pointer of the last node points to the first node of the list.

Create Node:

```
struct Node {
int data;
struct Node* next;
};
struct Node* head = NULL;
```

Insertion:

```
void Insert(int a){
struct Node t = (struct Node) malloc (si zeo f(s truct Node));
t->data = a;
t->next = head;
head= t;
}
```

Deletion:

```
void Delete(){
if(hea d!= NULL){
struct Node *t = head;
head = head-> next;
free(t);
}}
```

Traversal:

```
void traver se(){
struct Node*t = head;
while( t!= NULL){
printf ("%d ", t-> data);
t = t-> next;
}}
```

Example:

```
int i, j;
```

Linked List (cont)

```
for(i = 1; i<=10; i++){
Insert(i);}
for(j = 1; j<=4; j++){
Delete();}
traver se();
```

Queue

It ensures that the data is kept in linear order. They are referred to by the first in first out (FIFO) rule.

Queue With Array:

```
void Enqueu e(int a){
if(count == 6){
printf ("No place in the queue. \n");}
else{
queue[ rear] = a;
rear++;
if(rear == 6) rear = 0;
count++;
printf ("%d added in the queue. \n", a);}}
void Dequeue(){
if(count == 0){
printf ("No data in the queue. \n");}
else{
int a = queue[ front];
front++;
if(front == 6) front = 0;
count--;
printf ("%d removed from the queue. \n", a);}}
```

Queue With Linked List:

```
struct Node {
int data;
struct Node* next;};
struct Node* front = NULL;
struct Node* rear = NULL;
void Enqueu e(int a){
struct Node* t = (struct Node*) malloc (si zeo f(s truct Node));
t->data = a;
t->next = NULL;
if(front == NULL && rear == NULL){
front = rear = t;}
else{
rear->next = t;
rear=t;}
```

Queue (cont)

```
printf ("%d added in the queue. \n", a); }
void Dequeue(){
if(front == NULL){
printf ("No data in the queue. \n");}
else{
struct Node* t= front;
if(front == rear){
front = rear = NULL;}
else{
front = front- >next;}
printf ("%d removed from the queue. \n", t-> dat a);}}
```

By **alicanpayasli**

cheatography.com/alicanpayasli/

Not published yet.
Last updated 11th October, 2023.