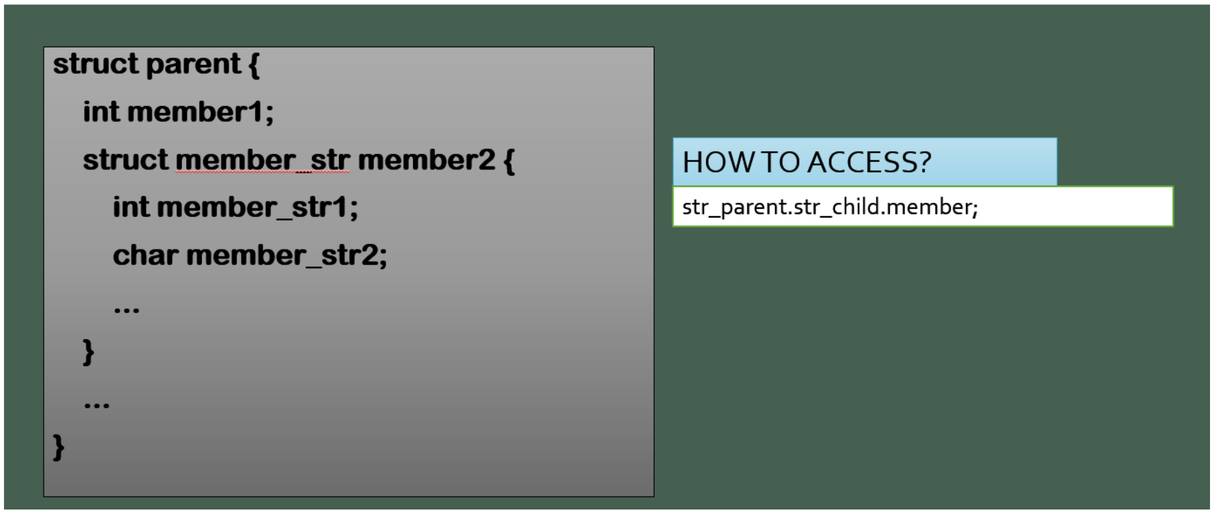
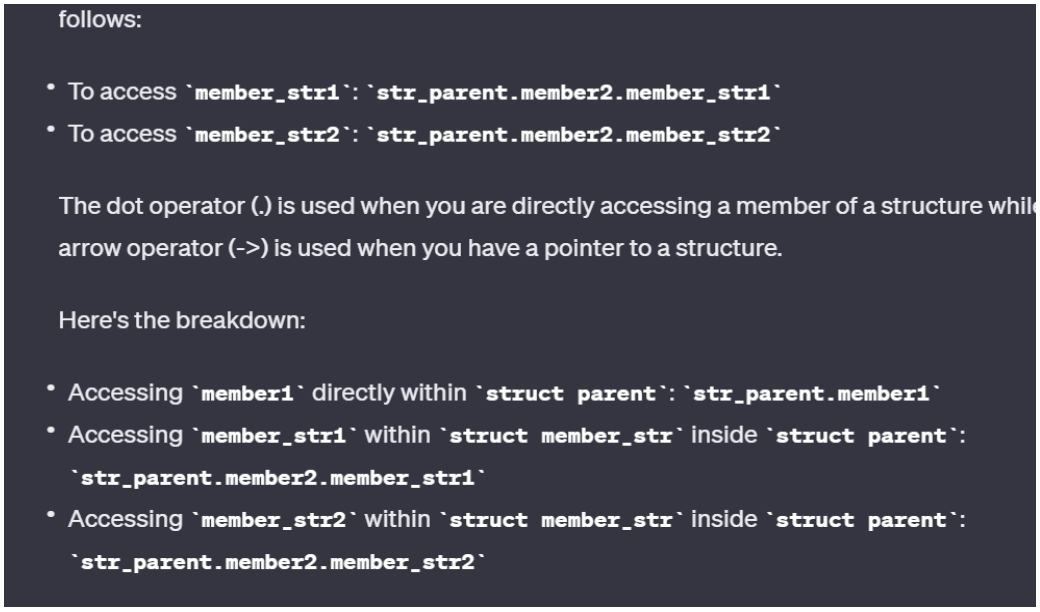


```
struct parent {
   int member1;
   struct member_str member2 {
      int member_str1;
      char member_str2;
      ...
   }
   ...
}
```

HOW TO ACCESS?

str_parent.str_child.member;



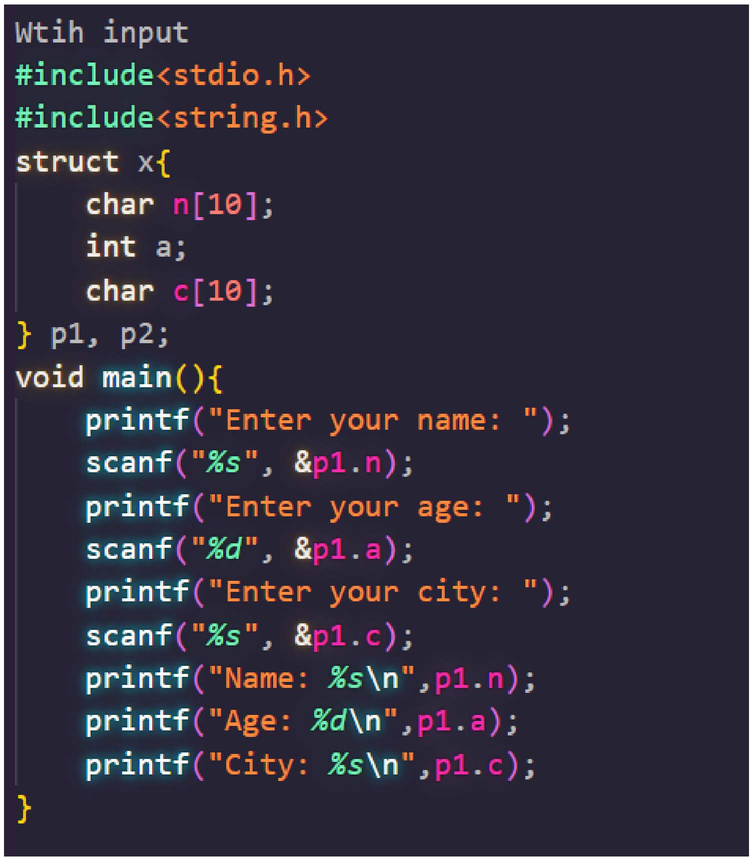
follows:

- To access `member_str1`: `str_parent.member2.member_str1`
- To access `member_str2`: `str_parent.member2.member_str2`

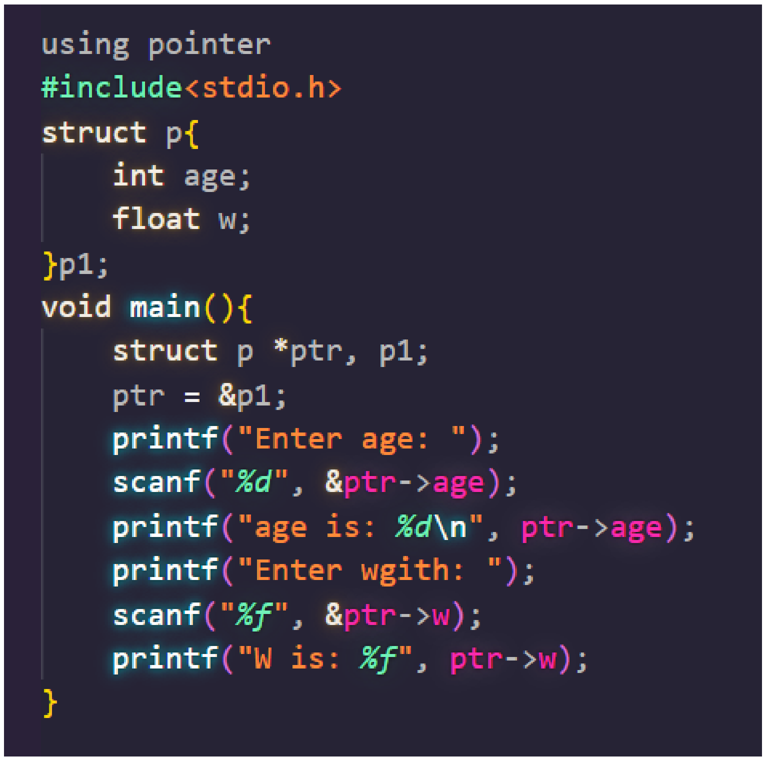
The dot operator (.) is used when you are directly accessing a member of a structure whil
arrow operator (->) is used when you have a pointer to a structure.

Here's the breakdown:

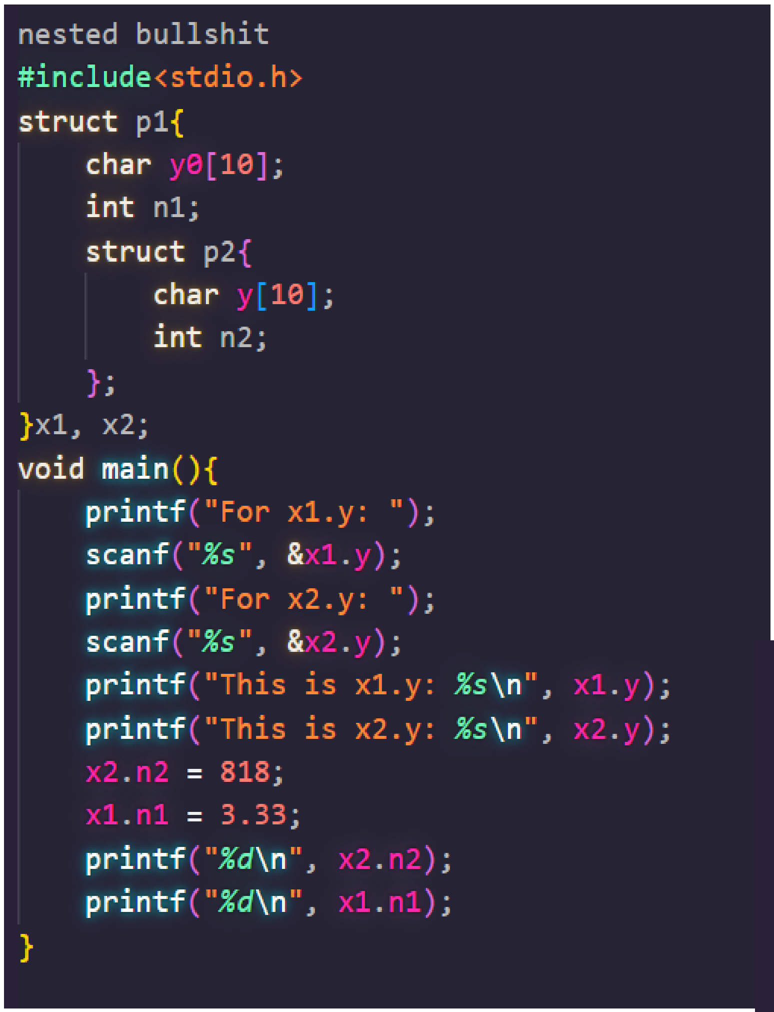
- Accessing `member1` directly within `struct parent`: `str_parent.member1`
- Accessing `member_str1` within `struct member_str` inside `struct parent`: `str_parent.member2.member_str1`
- Accessing `member_str2` within `struct member_str` inside `struct parent`: `str_parent.member2.member_str2`



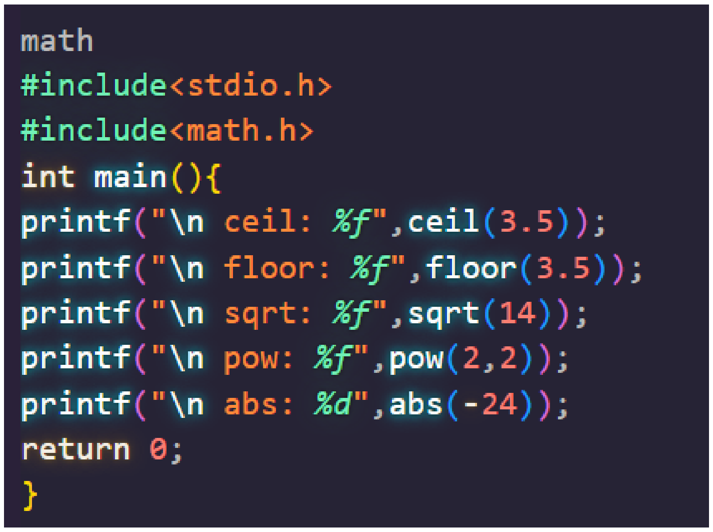
```
Wtih input
#include<stdio.h>
#include<string.h>
struct x{
    char n[10];
    int a;
    char c[10];
} p1, p2;
void main(){
    printf("Enter your name: ");
    scanf("%s", &p1.n);
    printf("Enter your age: ");
    scanf("%d", &p1.a);
    printf("Enter your city: ");
    scanf("%s", &p1.c);
    printf("Name: %s\n",p1.n);
    printf("Age: %d\n",p1.a);
    printf("City: %s\n",p1.c);
}
```



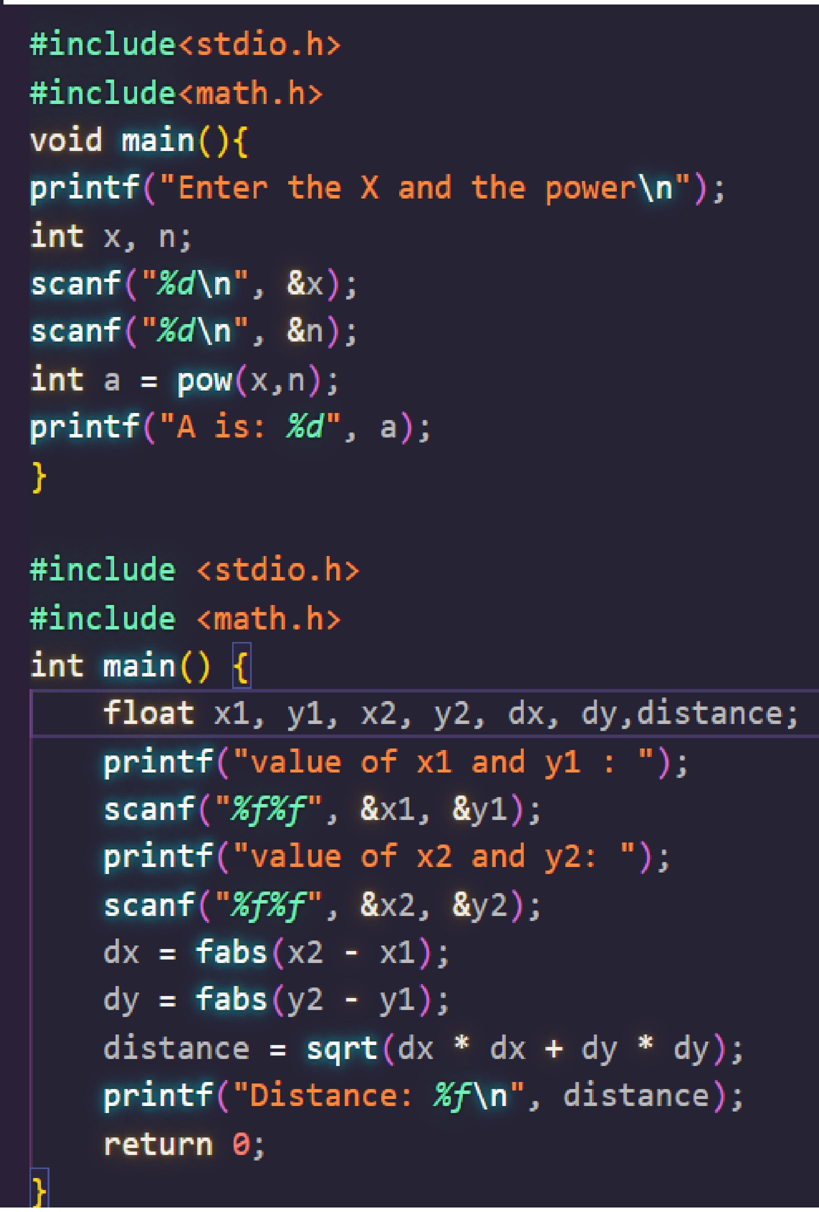
```
using pointer
#include<stdio.h>
struct p{
    int age;
    float w;
}p1;
void main(){
    struct p *ptr, p1;
    ptr = &p1;
    printf("Enter age: ");
    scanf("%d", &ptr->age);
    printf("age is: %d\n", ptr->age);
    printf("Enter wgith: ");
    scanf("%f", &ptr->w);
    printf("W is: %f", ptr->w);
}
```



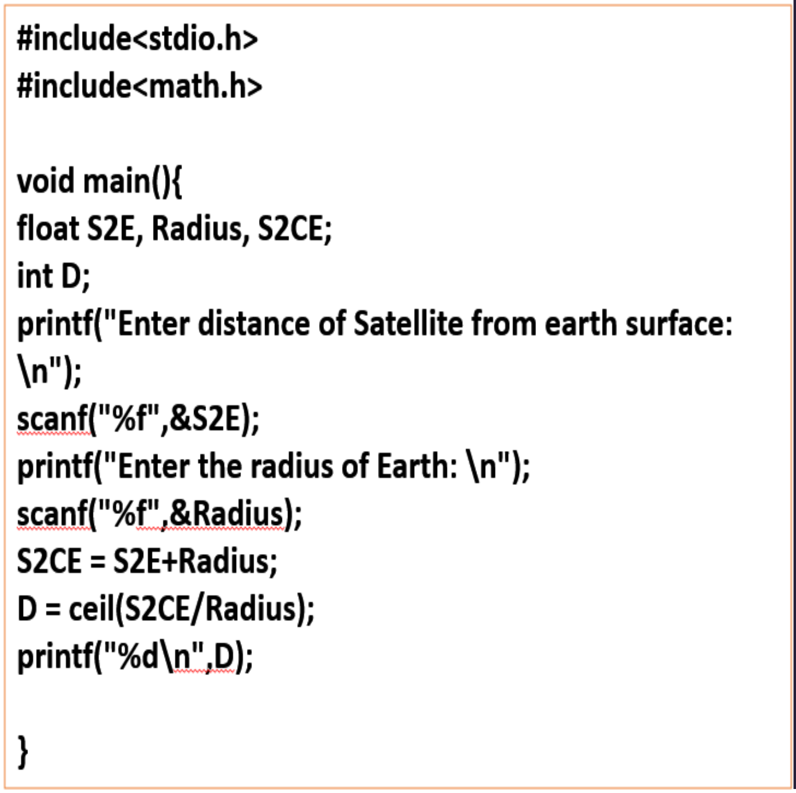
```
nested bullshit
#include<stdio.h>
struct p1{
    char y0[10];
    int n1;
    struct p2{
        char y[10];
        int n2;
    };
}x1, x2;
void main(){
    printf("For x1.y: ");
    scanf("%s", &x1.y);
    printf("For x2.y: ");
    scanf("%s", &x2.y);
    printf("This is x1.y: %s\n", x1.y);
    printf("This is x2.y: %s\n", x2.y);
    x2.n2 = 818;
    x1.n1 = 3.33;
    printf("%d\n", x2.n2);
    printf("%d\n", x1.n1);
}
```



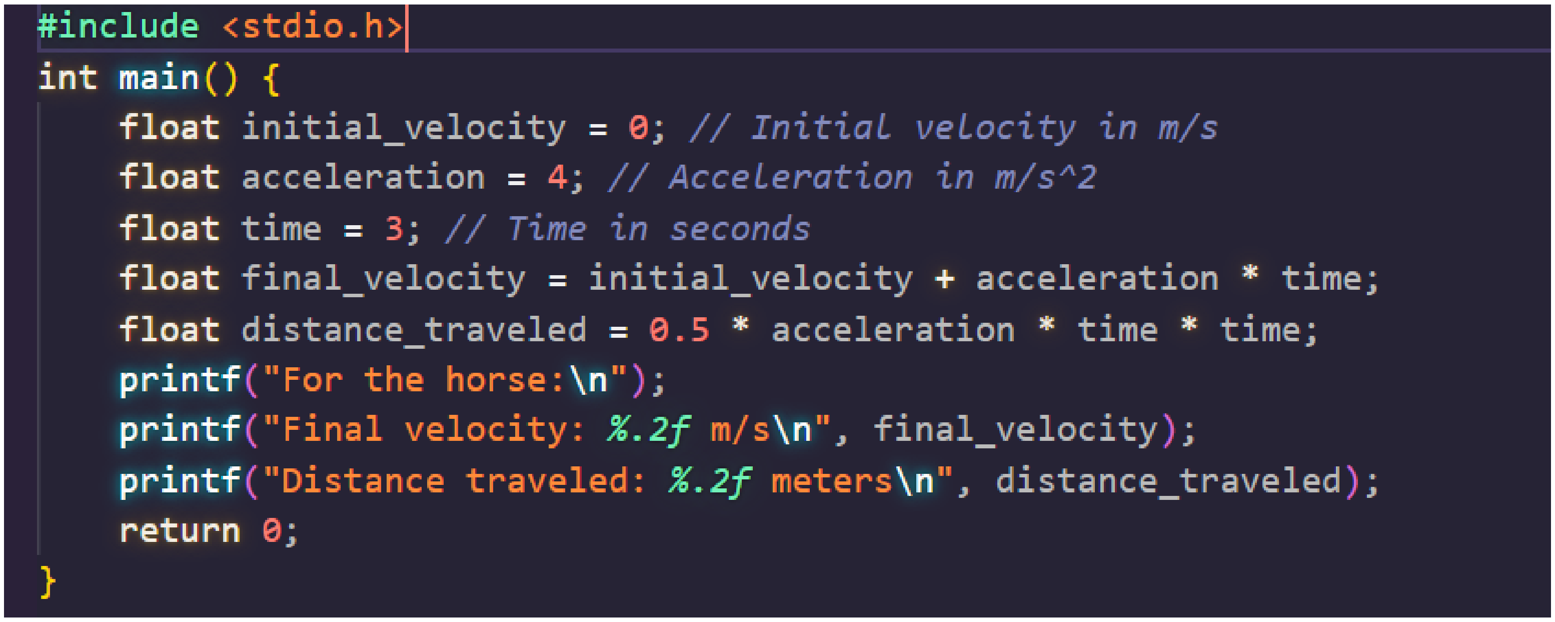
```
math
#include<stdio.h>
#include<math.h>
int main(){
printf("\n ceil: %f",ceil(3.5));
printf("\n floor: %f",floor(3.5));
printf("\n sqrt: %f",sqrt(14));
printf("\n pow: %f",pow(2,2));
printf("\n abs: %d",abs(-24));
return 0;
}
```



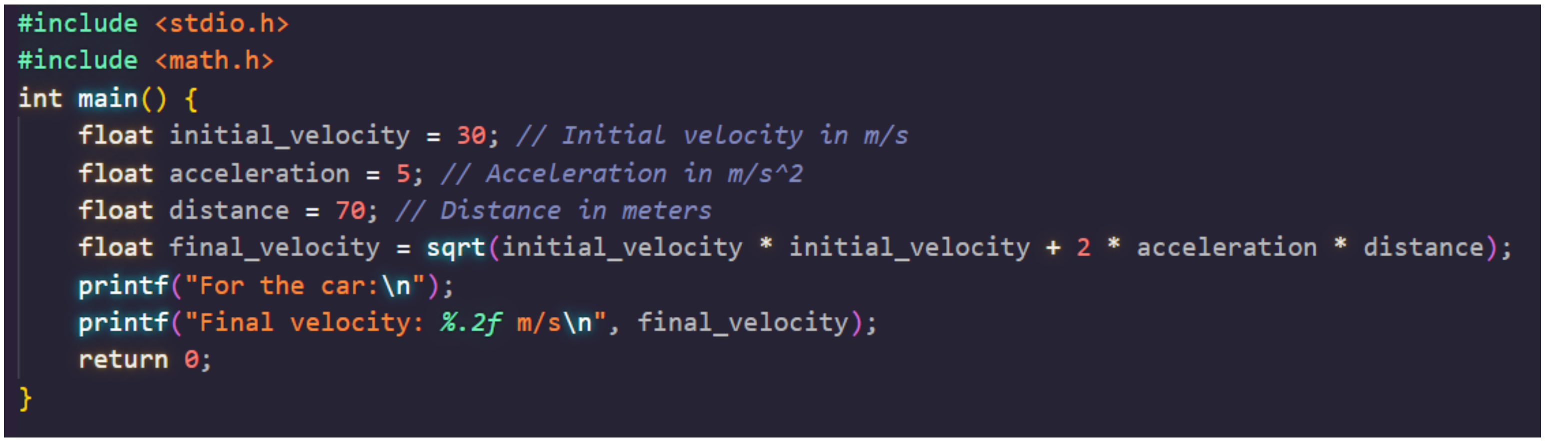
```
#include<stdio.h>
#include<math.h>
void main(){
printf("Enter the X and the power\n");
int x, n;
scanf("%d\n", &x);
scanf("%d\n", &n);
int a = pow(x,n);
printf("A is: %d", a);
}

#include <stdio.h>
#include <math.h>
int main() {
    float x1, y1, x2, y2, dx, dy,distance;
    printf("value of x1 and y1 : ");
    scanf("%f%f", &x1, &y1);
    printf("value of x2 and y2: ");
    scanf("%f%f", &x2, &y2);
    dx = fabs(x2 - x1);
    dy = fabs(y2 - y1);
    distance = sqrt(dx * dx + dy * dy);
    printf("Distance: %f\n", distance);
    return 0;
}
```



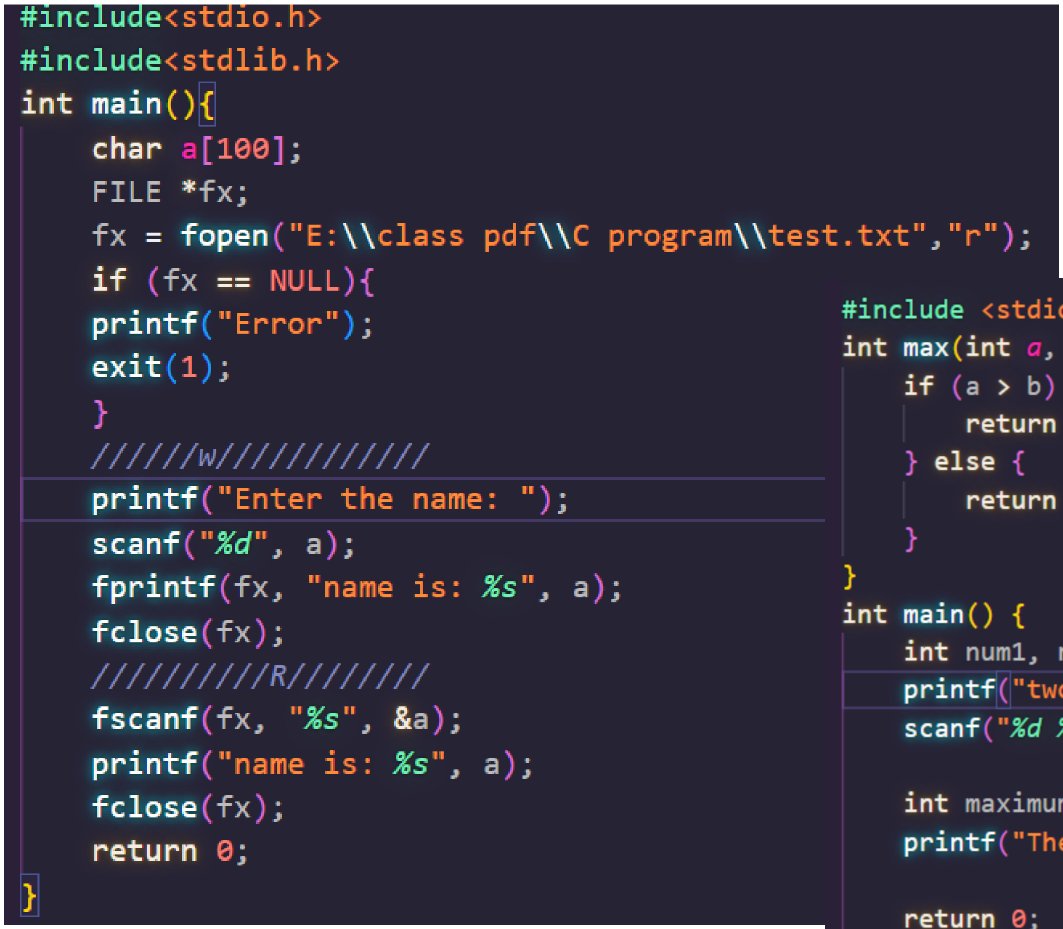
```
#include<stdio.h>
#include<math.h>

void main(){
float S2E, Radius, S2CE;
int D;
printf("Enter distance of Satellite from earth surface:
\n");
scanf("%f",&S2E);
printf("Enter the radius of Earth: \n");
scanf("%f",&Radius);
S2CE = S2E+Radius;
D = ceil(S2CE/Radius);
printf("%d\n",D);

}
```



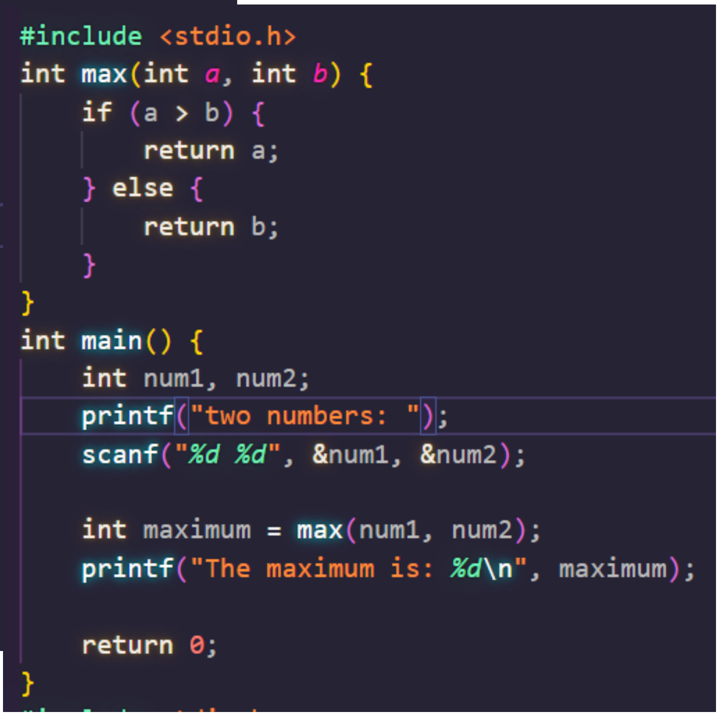
```c
#include <stdio.h>
int main() {
    float initial_velocity = 0; // Initial velocity in m/s
    float acceleration = 4; // Acceleration in m/s^2
    float time = 3; // Time in seconds
    float final_velocity = initial_velocity + acceleration * time;
    float distance_traveled = 0.5 * acceleration * time * time;
    printf("For the horse:\n");
    printf("Final velocity: %.2f m/s\n", final_velocity);
    printf("Distance traveled: %.2f meters\n", distance_traveled);
    return 0;
}
```



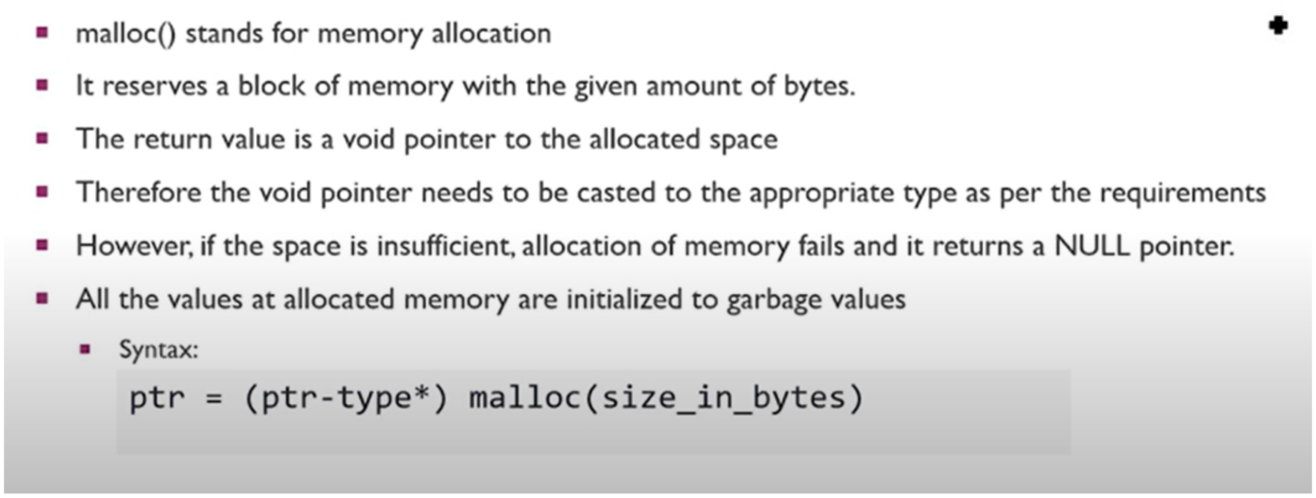
```c
#include <stdio.h>
#include <math.h>
int main() {
    float initial_velocity = 30; // Initial velocity in m/s
    float acceleration = 5; // Acceleration in m/s^2
    float distance = 70; // Distance in meters
    float final_velocity = sqrt(initial_velocity * initial_velocity + 2 * acceleration * distance);
    printf("For the car:\n");
    printf("Final velocity: %.2f m/s\n", final_velocity);
    return 0;
}
```



```c
#include<stdio.h>
#include<stdlib.h>
int main(){
    char a[100];
    FILE *fx;
    fx = fopen("E:\\class pdf\\C program\\test.txt","r");
    if (fx == NULL){
    printf("Error");
    exit(1);
    }
    //////w////////////
    printf("Enter the name: ");
    scanf("%d", a);
    fprintf(fx, "name is: %s", a);
    fclose(fx);
    //////////R////////
    fscanf(fx, "%s", &a);
    printf("name is: %s", a);
    fclose(fx);
    return 0;
}
```

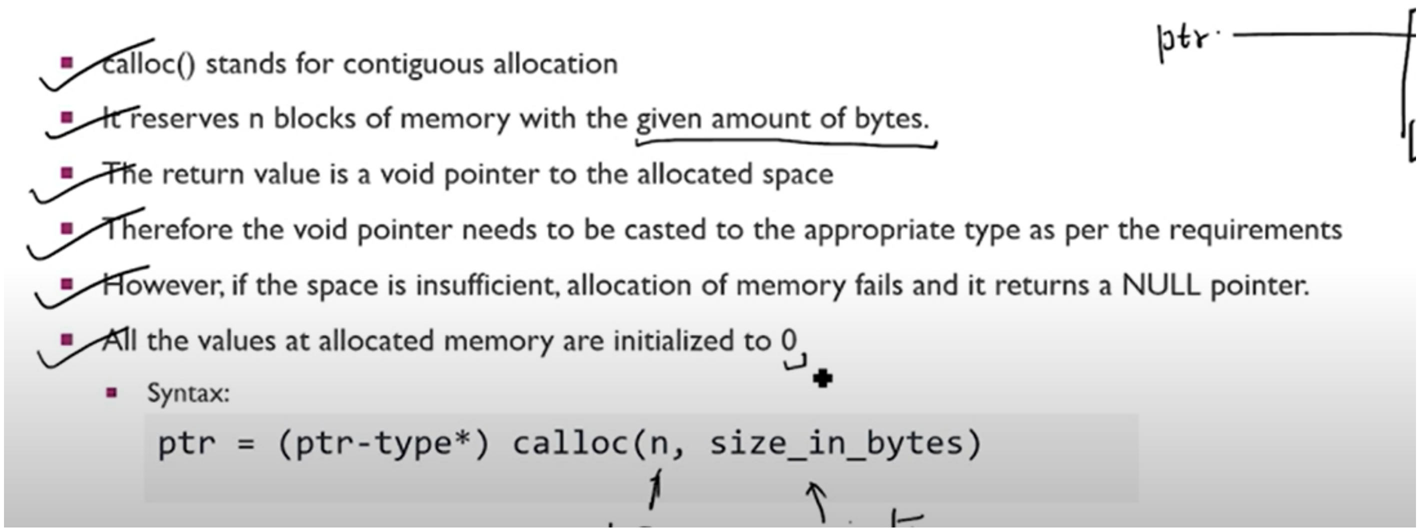


```c
#include <stdio.h>
int max(int a, int b) {
    if (a > b) {
        return a;
    } else {
        return b;
    }
}
int main() {
    int num1, num2;
    printf("two numbers: ");
    scanf("%d %d", &num1, &num2);

    int maximum = max(num1, num2);
    printf("The maximum is: %d\n", maximum);

    return 0;
}
```



- malloc() stands for memory allocation
- It reserves a block of memory with the given amount of bytes.
- The return value is a void pointer to the allocated space
- Therefore the void pointer needs to be casted to the appropriate type as per the requirements
- However, if the space is insufficient, allocation of memory fails and it returns a NULL pointer.
- All the values at allocated memory are initialized to garbage values
  - Syntax:

```
ptr = (ptr-type*) malloc(size_in_bytes)
```



- calloc() stands for contiguous allocation
- It reserves n blocks of memory with the given amount of bytes.
- The return value is a void pointer to the allocated space
- Therefore the void pointer needs to be casted to the appropriate type as per the requirements
- However, if the space is insufficient, allocation of memory fails and it returns a NULL pointer.
- All the values at allocated memory are initialized to 0
  - Syntax:

```
ptr = (ptr-type*) calloc(n, size_in_bytes)
```

Text Files

- Text files are the normal .txt files. You can easily create text files using any simple text editors such as Notepad.
- When you open those files, you'll see all the contents within the file as plain text. You can easily edit or delete the contents.
- They take minimum effort to maintain, are easily readable, and provide the least security and takes bigger storage space.

Binary Files

- Binary files are mostly the .bin files in your computer.
- Instead of storing data in plain text, they store it in the binary form (0's and 1's).
- They can hold a higher amount of data, are not readable easily, and provides better security than text files.