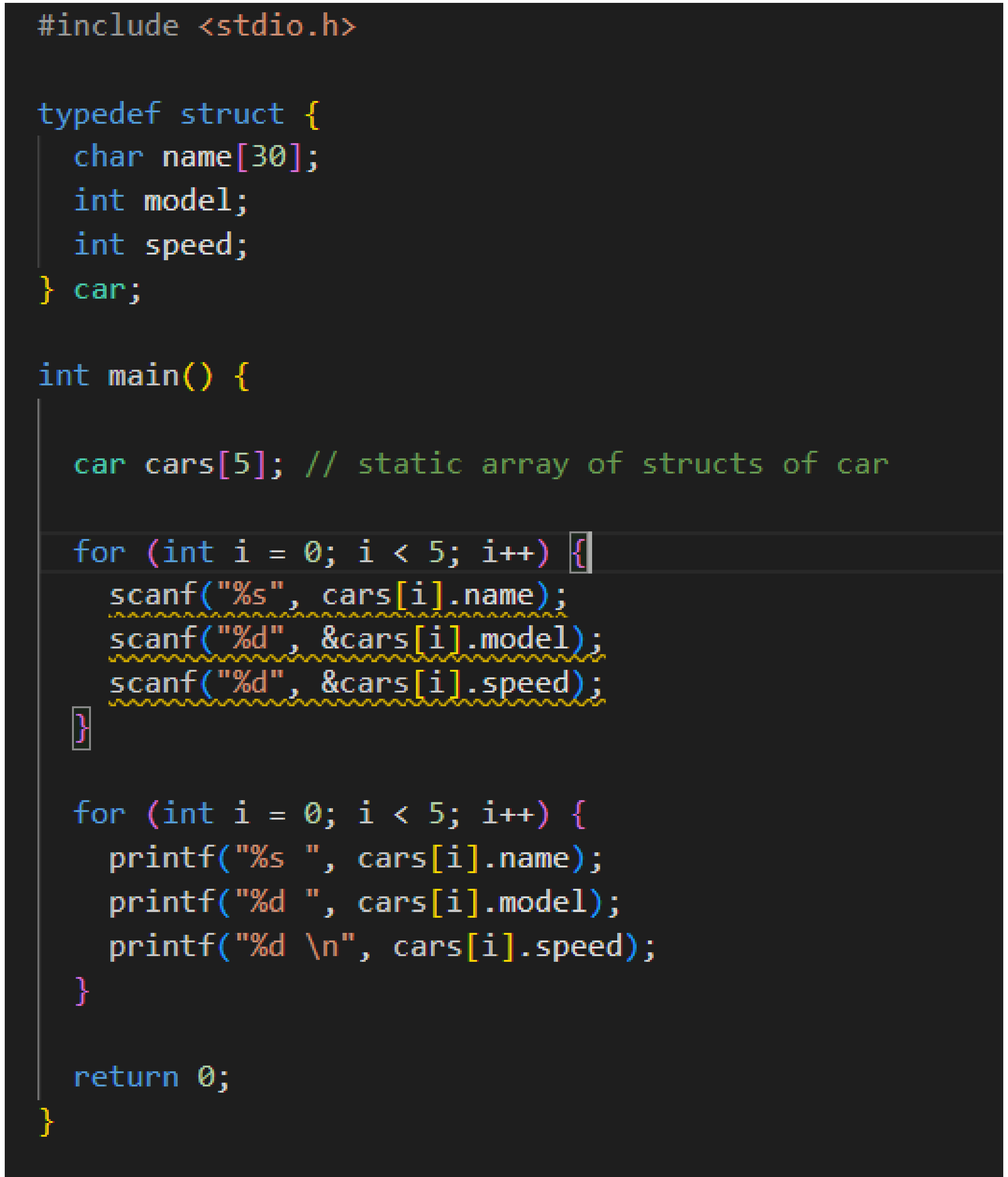
Session 13 & 14

Mostafa Akram

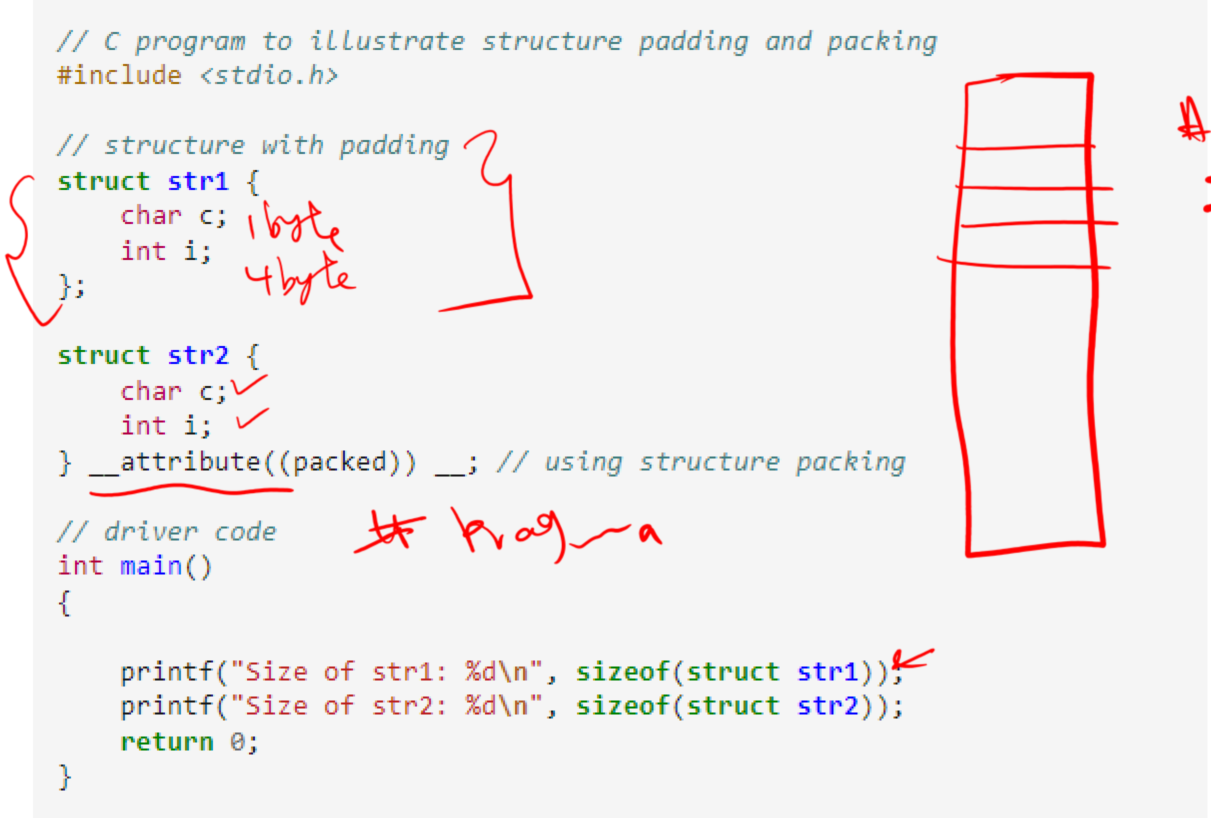
Array of structs



```c
#include <stdio.h>

typedef struct {
  char name[30];
  int model;
  int speed;
} car;

int main() {

  car cars[5]; // static array of structs of car

  for (int i = 0; i < 5; i++) {
    scanf("%s", cars[i].name);
    scanf("%d", &cars[i].model);
    scanf("%d", &cars[i].speed);
  }

  for (int i = 0; i < 5; i++) {
    printf("%s ", cars[i].name);
    printf("%d ", cars[i].model);
    printf("%d \n", cars[i].speed);
  }

  return 0;
}
```

#define #pragma

C Structure Padding and Packing



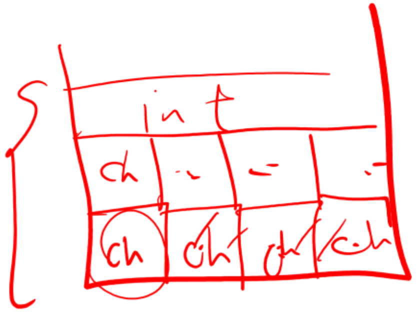
```c
// C program to illustrate structure padding and packing
#include <stdio.h>

// structure with padding
struct str1 {
    char c;
    int i;
};

struct str2 {
    char c;
    int i;
} __attribute((packed)) __; // using structure packing

// driver code
int main()
{

    printf("Size of str1: %d\n", sizeof(struct str1));
    printf("Size of str2: %d\n", sizeof(struct str2));
    return 0;
}
```

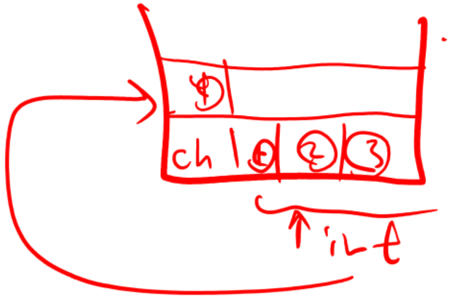
1byte

4byte

pragma

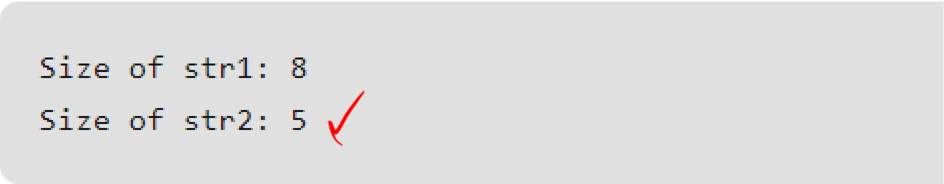
#Pragma attribute Padding





Packing

Output



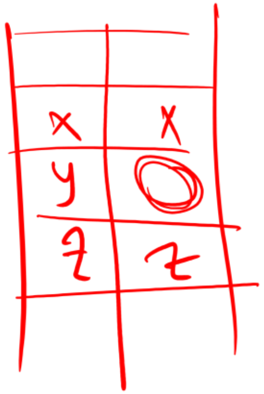
```
Size of str1: 8
Size of str2: 5 ✓
```

① Ⓐ Packing i's the actual size of variables in the struct.

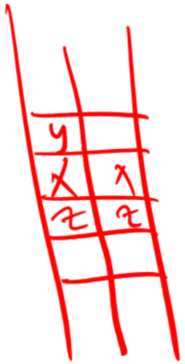
② Ⓐ packing done by using → # Pragma
→ attribute

③ Ⓐ Padding is the size of the biggest variable multiplied by the number of variables

```
Short x;
char y;   } 6 byte
Short z;
```

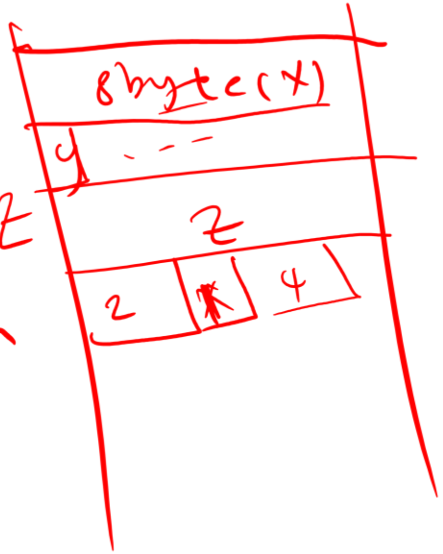


```
char y;
Short x;
Short z;
```



6 byte

```
{ long long x
  char j
  long long z
  Short m
  char x2
  int n
```



Example

```
// structure A
typedef struct structa_tag {
             char c;
             short int s;
} structa_t;

// structure B
typedef struct structb_tag {
             short int s;
             char c;
             int i;
} structb_t;

// structure C
typedef struct structc_tag {
             char c;
             double d;
             int s;
} structc_t;

// structure D
typedef struct structd_tag {
             double d;
             int s;
             char c;
} structd_t;
```

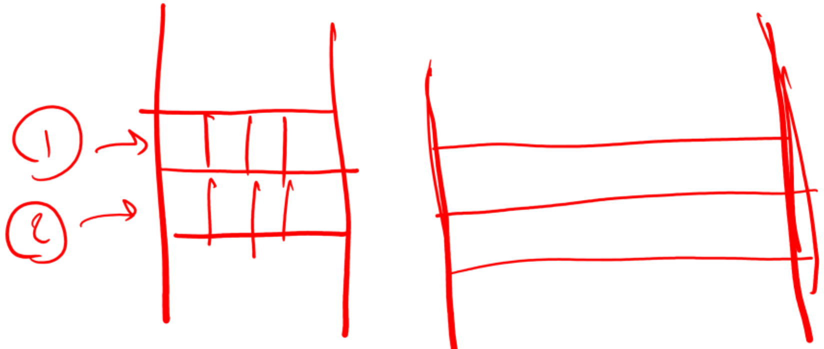
C Structure Padding and Packing

- Structure padding is the concept of adding multiple empty bytes in the structure to naturally align the data members in the memory. It is done to minimize the CPU read cycles to retrieve different data members in the structure.
- Structure Packing. C language provides two ways for structure packing:
    1. Using #pragma pack(1)
    2. Using __attribute((packed))__
    3. .. Note if struct declared using typedef so use __attribute((packed)) “struct name”

1 byte = 8 bits

Data Alignment in Memory

like padding long double → 16 byte

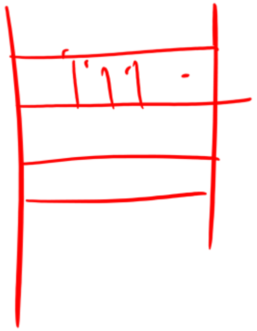


- A variable’s ***data alignment*** deals with the way the data is stored in these banks. For example, the natural alignment of ***int*** on a 32-bit machine is 4 bytes. When a data type is naturally aligned, the CPU fetches it in minimum read cycles.
- Note that a **double** variable will be allocated on an 8-byte boundary on a 32-bit machine and requires two memory read cycles. On a 64-bit machine, based on a number of banks, a **double** variable will be allocated on the 8-byte boundary and requires only one memory read cycle.

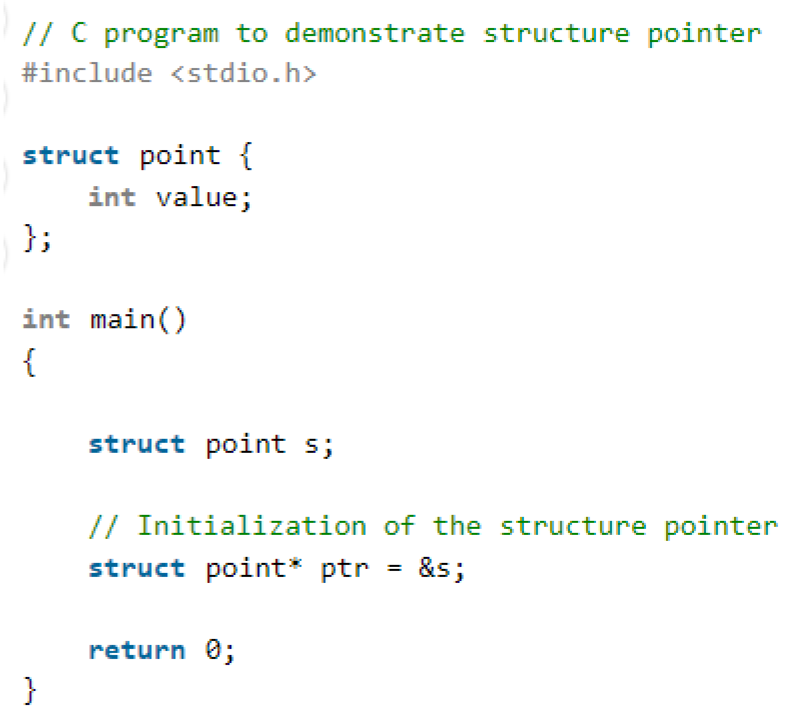
8 byte

4 byte → 1 sec

8 byte → 1 sec



Pointer to struct

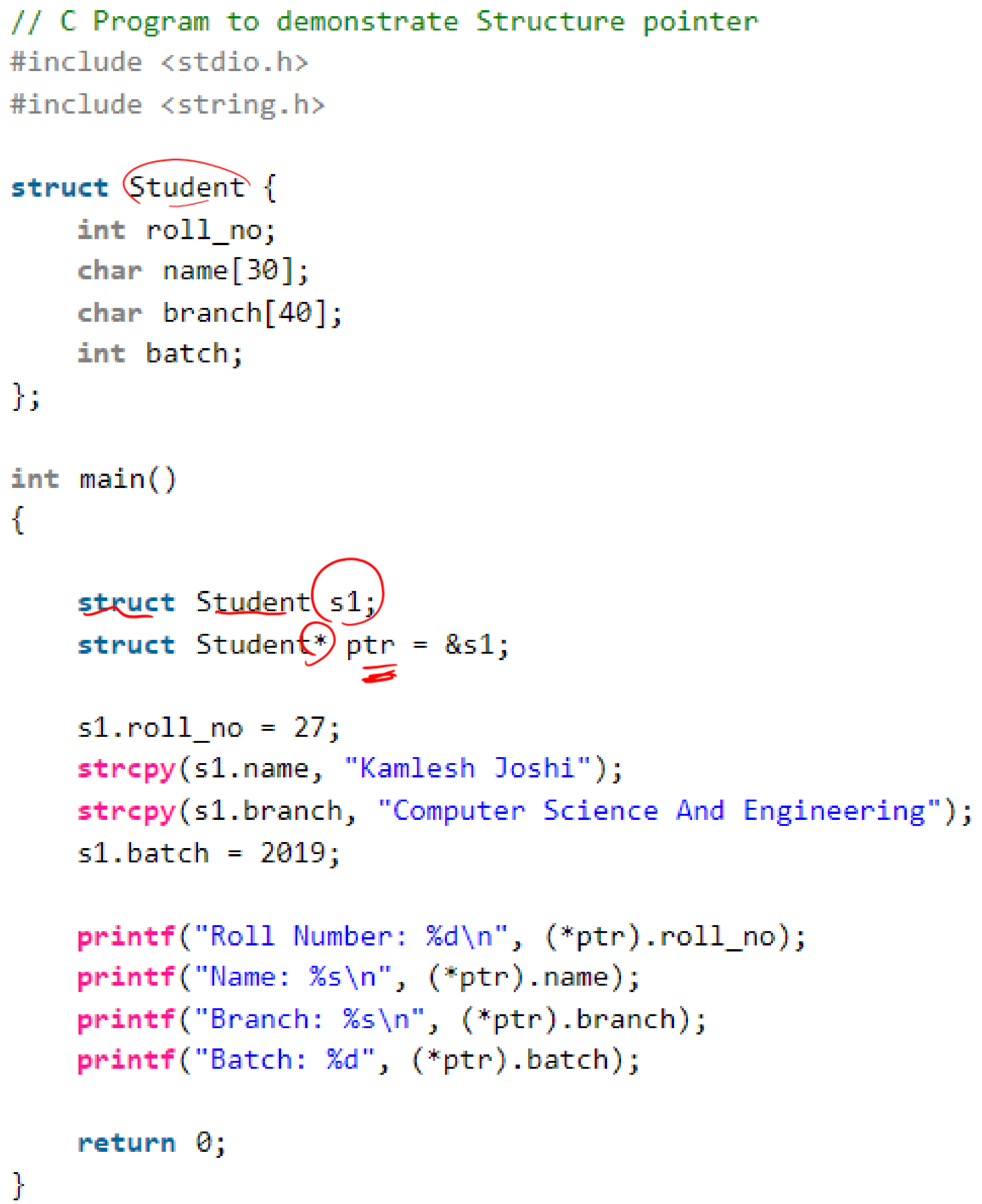


```c
// C program to demonstrate structure pointer
#include <stdio.h>

struct point {
    int value;
};

int main()
{

    struct point s;

    // Initialization of the structure pointer
    struct point* ptr = &s;

    return 0;
}
```

Pointer to struct

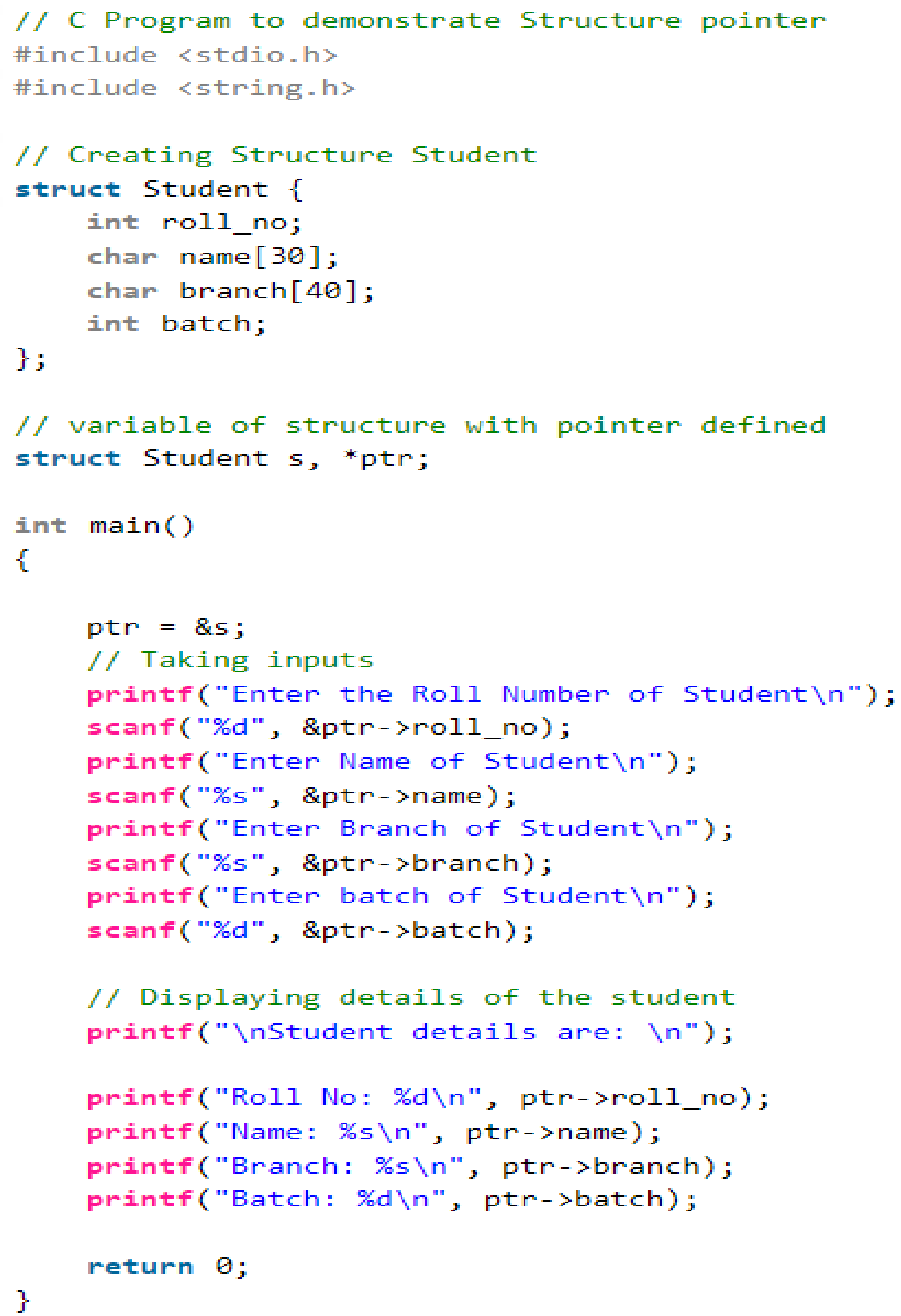
Char x;

char * ptr;



```c
// C Program to demonstrate Structure pointer
#include <stdio.h>
#include <string.h>

struct Student {
    int roll_no;
    char name[30];
    char branch[40];
    int batch;
};

int main()
{

    struct Student s1;
    struct Student* ptr = &s1;

    s1.roll_no = 27;
    strcpy(s1.name, "Kamlesh Joshi");
    strcpy(s1.branch, "Computer Science And Engineering");
    s1.batch = 2019;

    printf("Roll Number: %d\n", (*ptr).roll_no);
    printf("Name: %s\n", (*ptr).name);
    printf("Branch: %s\n", (*ptr).branch);
    printf("Batch: %d", (*ptr).batch);

    return 0;
}
```

Pointer to struct



```
// C Program to demonstrate Structure pointer
#include <stdio.h>
#include <string.h>

// Creating Structure Student
struct Student {
    int roll_no;
    char name[30];
    char branch[40];
    int batch;
};

// variable of structure with pointer defined
struct Student s, *ptr;

int main()
{

    ptr = &s;
    // Taking inputs
    printf("Enter the Roll Number of Student\n");
    scanf("%d", &ptr->roll_no);
    printf("Enter Name of Student\n");
    scanf("%s", &ptr->name);
    printf("Enter Branch of Student\n");
    scanf("%s", &ptr->branch);
    printf("Enter batch of Student\n");
    scanf("%d", &ptr->batch);

    // Displaying details of the student
    printf("\nStudent details are: \n");

    printf("Roll No: %d\n", ptr->roll_no);
    printf("Name: %s\n", ptr->name);
    printf("Branch: %s\n", ptr->branch);
    printf("Batch: %d\n", ptr->batch);

    return 0;
}
```

Task

- Search:
  - Bit field using struct

- Example of code on bit field using stuct

Links