02_10_ C Structure

C Structure

- **C Structure** is a collection of different data types which are grouped together and each element in a C structure is called member.
- If you want to access structure members in C, structure variable should be declared.
- Many structure variables can be declared for same structure and memory will be allocated for each separately.
- It is a best practice to initialize a structure to null while declaring, if we don’t assign any values to structure members.

Difference between C variable, C array and C structure

- A normal C variable can hold only one data of one data type at a time.
- An array can hold group of data of same data type.
- A structure can hold group of data of different data types
- Data types can be int, char, float, double and long double etc.

Datatype	C variable		C array		C structure	
	Syntax	Example	Syntax	Example	Syntax	Example

int	`int a`	`a = 20`	`int a[3]`	`a[0] = 10`
				`a[1] = 20`
				`a[2] = 30`
				`a[3] = '\0'`
					`struct student`
					`{`
					`int a;`
					`char b[10];`
					`}`
						`a = 10`
						`b = "Hello"`

char	`char b`	`b='Z'`	`char b[10]`	`b="Hello"`		

Concepts in C structure

- How to declare a C structure?
- How to initialize a C structure?
- How to access the members of a C structure?

Type	Using normal variable	Using pointer variabe

Syntax	`struct tag_name`
	`{`
	`data type var_name1;`
	`data type var_name2;`
	`data type var_name3;`
	`};`
		`struct tag_name`
		`{`
		`data type var_name1;`
		`data type var_name2;`
		`data type var_name3;`
		`};`

Example	`struct student`
	`{`
	`int  mark;`
	`char name[10];`
	`float average;`
	`};`
		`struct student`
		`{`
		`int  mark;`
		`char name[10];`
		`float average;`
		`};`

Declaring structure variable	`struct student report;`	`struct student *report, rep;`
Initializing structure variable	`struct student report = {100, "Mani", 99.5};`	`struct student rep = {100, "Mani", 99.5};` `report = &rep;`

Accessing structure members	`report.mark`
	`report.name`
	`report.average`
		`report -> mark`
		`report -> name`
		`report -> average`

C structure

- This program is used to store and access “id, name and percentage” for one student. We can also store and access these data for many students using array of structures.

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

struct student
{
            int id;
            char name[20];
            float percentage;
};

int main()
{
            struct student record = {0}; //Initializing to null

            record.id=1;
            strcpy(record.name, "Raju");
            record.percentage = 86.5;

            printf(" Id is: %d \n", record.id);
            printf(" Name is: %s \n", record.name);
            printf(" Percentage is: %f \n", record.percentage);
            return 0;
}
```

```
Id is: 1
Name is: Raju
Percentage is: 86.500000
```

Another way of declaring C structure

- In this program, structure variable “record” is declared while declaring structure itself. In above structure example program, structure variable “struct student record” is declared inside main function which is after declaring structure.

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

struct student
{
            int id;
            char name[20];
            float percentage;
} record;

int main()
{

            record.id=1;
            strcpy(record.name, "Raju");
            record.percentage = 86.5;

            printf(" Id is: %d \n", record.id);
            printf(" Name is: %s \n", record.name);
            printf(" Percentage is: %f \n", record.percentage);
            return 0;
}
```

```
Id is: 1
Name is: Raju
Percentage is: 86.500000
```

C structure declaration in separate header file

- In above structure programs, C structure is declared in main source file. Instead of declaring C structure in main source file, we can have this structure declaration in another file called “header file” and we can include that header file in main source file as shown below.

Header file name – `structure.h`

- Before compiling and executing below C program, create a file named “structure.h” and declare the below structure.

```
struct student
{
int id;
char name[20];
float percentage;
} record;
```

Main file name – `structure.c`

- In this program, above created header file is included in "`structure.c`" source file as `#include` "`Structure.h`".
- So, the structure declared in "`structure.h`" file can be used in "`structure.c`" source file.

```
// File name - structure.c
#include <stdio.h>
#include <string.h>
#include "structure.h"   /* header file where C structure is declared */

int main()
{

   record.id=1;
   strcpy(record.name, "Raju");
   record.percentage = 86.5;

   printf(" Id is: %d \n", record.id);
   printf(" Name is: %s \n", record.name);
   printf(" Percentage is: %f \n", record.percentage);
   return 0;
}
```

```
Id is: 1
Name is: Raju
Percentage is: 86.500000
```

Uses of C structures

- C Structures can be used to store huge data. Structures act as a database.
- C Structures can be used to send data to the printer.
- C Structures can interact with keyboard and mouse to store the data.
- C Structures can be used to clear output screen contents.
- C Structures can be used to check computer's memory size etc.

C – Array of Structures

- As you know, C Structure is collection of different datatypes (variables) which are grouped together.

- Whereas, array of structures is nothing but collection of structures. This is also called as **structure array in C**.

- **Example program for array of structures in C:**

- This program is used to store and access “id, name and percentage” for 3 students. Structure array is used in this program to store and display records for many students. You can store “n” number of students record by declaring structure variable as ‘struct student record[n]“, where n can be 1000 or 5000 etc.

```c
#include <stdio.h>
#include <string.h>

struct student
{
     int id;
     char name[30];
     float percentage;
};

int main()
{
     int i;
     struct student record[3];

     // 1st student's record
     record[0].id=1;
     strcpy(record[0].name, "Raju");
     record[0].percentage = 86.5;

     // 2nd student's record
     record[1].id=2;
     strcpy(record[1].name, "Surendren");
     record[1].percentage = 90.5;

     // 3rd student's record
     record[2].id=3;
     strcpy(record[2].name, "Thiyagu");
     record[2].percentage = 81.5;

     for(i=0; i<3; i++)
     {
         printf(" Records of STUDENT : %d \n", i+1);
         printf(" Id is: %d \n", record[i].id);
         printf(" Name is: %s \n", record[i].name);
         printf(" Percentage is: %f\n\n",record[i].percentage);
     }
     return 0;
}
```

```
Records of STUDENT : 1
Id is: 1
Name is: Raju
Percentage is: 86.500000
Records of STUDENT : 2
Id is: 2
Name is: Surendren
Percentage is: 90.500000
Records of STUDENT : 3
Id is: 3
Name is: Thiyagu
Percentage is: 81.500000
```

Declaring many structure variable in C

- In this program, two structure variables "**`record1`**" and "**`record2`**" are declared for same structure and different values are assigned for both structure variables.

- Separate memory is allocated for both structure variables to store the data.

```c
#include <stdio.h>
#include <string.h>

struct student
{
     int id;
     char name[30];
     float percentage;
};

int main()
{
     int i;
     struct student record1 = {1, "Raju", 90.5};
     struct student record2 = {2, "Mani", 93.5};

     printf("Records of STUDENT1: \n");
     printf("  Id is: %d \n", record1.id);
     printf("  Name is: %s \n", record1.name);
     printf("  Percentage is: %f \n\n", record1.percentage);

     printf("Records of STUDENT2: \n");
     printf("  Id is: %d \n", record2.id);
     printf("  Name is: %s \n", record2.name);
     printf("  Percentage is: %f \n\n", record2.percentage);

     return 0;
}
```

```
Records of STUDENT1:
 Id is: 1
 Name is: Raju
 Percentage is: 90.500000
Records of STUDENT2:
 Id is: 2
 Name is: Mani
 Percentage is: 93.500000
```

C – Passing struct to function

- A structure can be passed to any function from main function or from any sub function.
- Structure definition will be available within the function only.
- It won't be available to other functions unless it is passed to those functions by value or by address(reference).
- Else, we have to declare structure variable as global variable. That means, structure variable should be declared outside the main function. So, this structure will be visible to all the functions in a C program.

Passing structure to function in C

- It can be done in below 3 ways.

    1. Passing structure to a function by value
    2. Passing structure to a function by address(reference)
    3. No need to pass a structure – Declare structure variable as global

Example program – passing structure to function in C by value:

- In this program, the whole structure is passed to another function by value. It means the whole structure is passed to another function with all members and their values.

- So, this structure can be accessed from called function.

```c
#include <stdio.h>
#include <string.h>

struct student
{
            int id;
            char name[20];
            float percentage;
};

void func(struct student record);

int main()
{
            struct student record;

            record.id=1;
            strcpy(record.name, "Raju");
            record.percentage = 86.5;

            func(record);
            return 0;
}

void func(struct student record)
{
            printf(" Id is: %d \n", record.id);
            printf(" Name is: %s \n", record.name);
            printf(" Percentage is: %f \n", record.percentage);
}
```

```
Id is: 1
Name is: Raju
Percentage is: 86.500000
```

Passing structure to function in C by address

Example program – Passing structure to function in C by address

- In this program, the whole structure is passed to another function by address. It means only the address of the structure is passed to another function.

- The whole structure is not passed to another function with all members and their values. So, this structure can be accessed from called function by its address.

```c
#include <stdio.h>
#include <string.h>

struct student
{
            int id;
            char name[20];
            float percentage;
};

void func(struct student *record);

int main()
{
           struct student record;

           record.id=1;
           strcpy(record.name, "Raju");
           record.percentage = 86.5;

           func(&record);
           return 0;
}

void func(struct student *record)
{
           printf(" Id is: %d \n", record->id);
           printf(" Name is: %s \n", record->name);
           printf(" Percentage is: %f \n", record->percentage);
}
```

```
Id is: 1
Name is: Raju
Percentage is: 86.500000
```

Declare a structure variable as global in C

Example program to declare a structure variable as global in C:

- Structure variables also can be declared as global variables as we declare other variables in C.

- So, When a structure variable is declared as global, then it is visible to all the functions in a program. In this scenario, we don’t need to pass the structure to any function separately.

```c
#include <stdio.h>
#include <string.h>

struct student
{
            int id;
            char name[20];
            float percentage;
};
struct student record; // Global declaration of structure

void structure_demo();

int main()
{
            record.id=1;
            strcpy(record.name, "Raju");
            record.percentage = 86.5;

            structure_demo();
            return 0;
}

void structure_demo()
{
            printf(" Id is: %d \n", record.id);
            printf(" Name is: %s \n", record.name);
            printf(" Percentage is: %f \n", record.percentage);
}
```

```
Id is: 1
Name is: Raju
Percentage is: 86.500000
```

C – Structure using Pointer

- C structure can be accessed in 2 ways in a C program. They are,

  1. Using normal structure variable
  2. Using pointer variable

- Dot(.) operator is used to access the data using normal structure variable and
- arrow (->) is used to access the data using pointer variable.

- You have learnt how to access structure data using normal variable in C – Structure topic.

- So, we are showing here how to access structure data using pointer variable in below C program.

C – Structure using Pointer

Example program for C structure using pointer:

- In this program, "`record1`" is normal structure variable and "`ptr`" is pointer structure variable.

```c
#include <stdio.h>
#include <string.h>

struct student
{
     int id;
     char name[30];
     float percentage;
};

int main()
{
     int i;
     struct student record1 = {1, "Raju", 90.5};
     struct student *ptr;

     ptr = &record1;

         printf("Records of STUDENT1: \n");
         printf("  Id is: %d \n", ptr->id);
         printf("  Name is: %s \n", ptr->name);
         printf("  Percentage is: %f \n\n", ptr->percentage);

     return 0;
}
```

```
Records of STUDENT1:
Id is: 1
Name is: Raju
Percentage is: 90.500000
```

C – Structure using Pointer

Example program to copy a structure in C

- There are many methods to copy one structure to another structure in C.

  1. We can copy using direct assignment of one structure to another structure or
  2. we can use C inbuilt function “memcpy()” or
  3. we can copy by individual structure members.

```c
#include <stdio.h>
#include <string.h>

struct student
{
    int id;
    char name[30];
    float percentage;
};

int main()
{
    int i;
    struct student record1 = {1, "Raju", 90.5};
    struct student record2, *record3, *ptr1, record4;

    printf("Records of STUDENT1 - record1 structure \n");
    printf("  Id : %d \n  Name : %s\n  Percentage : %f\n",
            record1.id, record1.name, record1.percentage);

    // 1st method to copy whole structure to another structure
    record2=record1;
    printf("\nRecords of STUDENT1 - Direct copy from " \
           "record1 \n");
    printf("  Id : %d \n  Name : %s\n  Percentage : %f\n",
            record2.id, record2.name, record2.percentage);

// 2nd method to copy using memcpy function
    ptr1 = &record1;
    memcpy(record3, ptr1, sizeof(record1));

    printf("\nRecords of STUDENT1 - copied from record1 " \
           "using memcpy \n");
    printf("  Id : %d \n  Name : %s\n  Percentage : %f\n",
           record3->id, record3->name, record3->percentage);
```

```c
// 3rd method to copy by individual members
    printf("\nRecords of STUDENT1 - Copied individual " \
           "members from record1 \n");
    record4.id=record1.id;
    strcpy(record4.name, record1.name);
    record4.percentage = record1.percentage;

    printf("  Id : %d \n  Name : %s\n  Percentage : %f\n",
            record4.id, record4.name, record4.percentage);

     return 0;
}
```

```
Records of STUDENT1 - record1 structure
Id : 1
Name : Raju
Percentage : 90.500000
Records of STUDENT1 - Direct copy from record1
Id : 1
Name : Raju
Percentage : 90.500000
Records of STUDENT1 - copied from record1 using memcpy
Id : 1
Name : Raju
Percentage : 90.500000
Records of STUDENT1 - Copied individual members from record1
Id : 1
Name : Raju
Percentage : 90.500000
```

C – Nested Structure

- Nested structure in C is nothing but structure within structure.
- One structure can be declared inside other structure as we declare structure members inside a structure.
- The structure variables can be a normal structure variable or a pointer variable to access the data. You can learn below concepts in this section.

    1. Structure within structure in C using normal variable
    2. Structure within structure in C using pointer variable

Structure within structure in C using normal variable

- This program explains how to use structure within structure in C using normal variable. “`student_college_detail`’ structure is declared inside “`student_detail`” structure in this program. Both structure variables are normal structure variables.

- Please note that members of “`student_college_detail`” structure are accessed by 2 dot(.) operator and members of “`student_detail`” structure are accessed by single dot(.) operator.

```c
#include <stdio.h>
#include <string.h>

struct student_college_detail
{
    int college_id;
    char college_name[50];
};

struct student_detail
{
    int id;
    char name[20];
    float percentage;
    // structure within structure
    struct student_college_detail clg_data;
}stu_data;
```

```c
int main()
{
    struct student_detail stu_data = {1, "Raju", 90.5,
71145,"Anna University"};
    printf(" Id is: %d \n", stu_data.id);
    printf(" Name is: %s \n", stu_data.name);
    printf(" Percentage is: %f \n\n", stu_data.percentage);
    printf(" College Id is: %d \n",
                    stu_data.clg_data.college_id);
    printf(" College Name is: %s \n",
                    stu_data.clg_data.college_name);
    return 0;
}
```

```
Id is: 1
Name is: Raju
Percentage is: 90.500000
College Id is: 71145
College Name is: Anna University
```

Structure within structure in C using pointer variable

- This program explains how to use structure within structure in C using pointer variable. “`student_college_detail`’ structure is declared inside “`student_detail`” structure in this program. one normal structure variable and one pointer structure variable is used in this program.

- Please note that combination of .(dot) and ->(arrow) operators are used to access the structure member which is declared inside the structure.

```c
#include <stdio.h>
#include <string.h>

struct student_college_detail
{
    int college_id;
    char college_name[50];
};

struct student_detail
{
    int id;
    char name[20];
    float percentage;
    // structure within structure
    struct student_college_detail clg_data;
}stu_data, *stu_data_ptr;
```

```c
int main()
{
  struct student_detail stu_data = {1, "Raju", 90.5, 71145,
                                    "Anna University"};
    stu_data_ptr = &stu_data;

    printf(" Id is: %d \n", stu_data_ptr->id);
    printf(" Name is: %s \n", stu_data_ptr->name);
    printf(" Percentage is: %f \n\n",
                          stu_data_ptr->percentage);

    printf(" College Id is: %d \n",
                          stu_data_ptr->clg_data.college_id);
    printf(" College Name is: %s \n",
                        stu_data_ptr->clg_data.college_name);

    return 0;
}
```

```
Id is: 1
Name is: Raju
Percentage is: 90.500000
College Id is: 71145
College Name is: Anna University
```

C – Struct memory allocation

- Do you know how memory is allocated for structure members in C?.

- You can learn below concepts of C in this topic.

    1. how structure members are stored in memory?
    2. What is structure padding?
    3. How to avoid structure padding?

- **How structure members are stored in memory?**
    - Always, contiguous(adjacent) memory locations are used to store structure members in memory.

    - Consider below example to understand how memory is allocated for structures.

```c
#include <stdio.h>
#include <string.h>

struct student
{
        int id1;
        int id2;
        char a;
        char b;
        float percentage;
};
```

```c
int main()
{
    int i;
    struct student record1 = {1, 2, 'A', 'B', 90.5};

    printf("size of structure in bytes : %d\n", sizeof(record1));

    printf("\nAddress of id1        = %u", &record1.id1 );
    printf("\nAddress of id2        = %u", &record1.id2 );
    printf("\nAddress of a          = %u", &record1.a );
    printf("\nAddress of b          = %u", &record1.b );
    printf("\nAddress of percentage = %u",&record1.percentage);

    return 0;
}
```

```
size of structure in bytes : 16
Address of id1 = 675376768
Address of id2 = 675376772
Address of a = 675376776
Address of b = 675376777
Address of percentage = 675376780
```

C – Struct memory allocation

- There are 5 members declared for structure in above program.

- In 32 bit compiler,
  - **`4 bytes`** of memory is occupied by **`int`** datatype.
  - **`1 byte`** of memory is occupied by **`char`** datatype and
  - **`4 bytes`** of memory is occupied by **`float`** datatype.

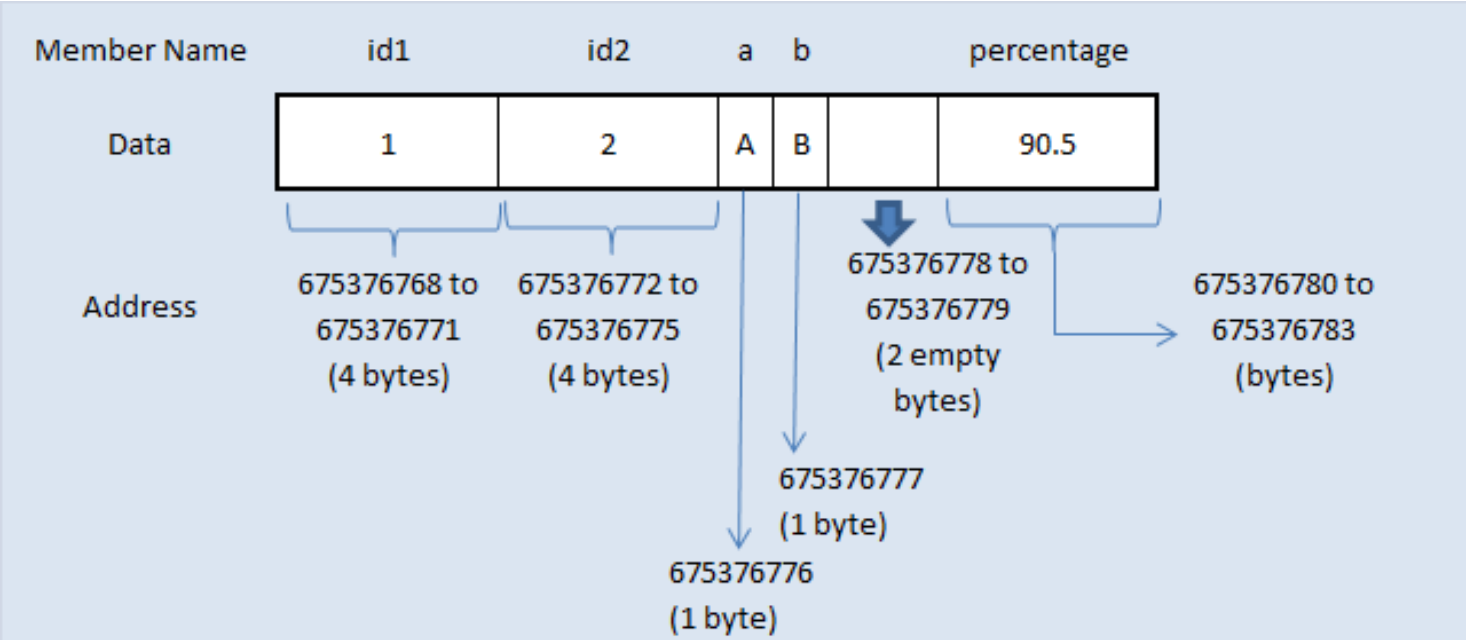
- Please refer below table to know from where to where memory is allocated for each datatype in contiguous (adjacent) location in memory.

C – Struct memory allocation

Datatype	Memory allocation in C (32 bit compiler)		
	From Address	To Address	Total bytes
`int id1`	675376768	675376771	4
`int id2`	**675376772**	**675376775**	**4**
`char a`	675376776		1
`char b`	**675376777**		**1**
Addresses 675376778 and 675376779 are left empty (Do you know why? Please see Structure padding topic later in class)			**2**
`float percentage`	675376780	675376783	4

C – Struct memory allocation

- The representation of above structure memory allocation is given below.
- This diagram will help you to understand the memory allocation concept in C very easily.



C – Structure Padding

- In order to align the data in memory, one or more empty bytes (addresses) are inserted (or left empty) between memory addresses which are allocated for other structure members while memory allocation. This concept is called **structure padding**.

- Architecture of a computer processor is such a way that it can read 1 word (4 byte in 32 bit processor) from memory at a time.

- To make use of this advantage of processor, data are always aligned as 4 bytes package which leads to insert empty addresses between other member's address.

- Because of this structure padding concept in C, size of the structure is always not same as what we think.

C – Structure Padding

- For example, please consider below structure that has 5 members.

```
struct student
{
        int id1;
        int id2;
        char a;
        char b;
        float percentage;
};
```

- As per C concepts, int and float datatypes occupy 4 bytes each and char datatype occupies 1 byte for 32 bit processor. So, only 14 bytes (4+4+1+1+4) should be allocated for above structure.

- But, this is wrong. Do you know why?

C – Structure Padding

- Architecture of a computer processor is such a way that it can read 1 word from memory at a time.
- 1 word is equal to 4 bytes for 32 bit processor and 8 bytes for 64 bit processor.
- So, 32 bit processor always reads 4 bytes at a time and 64 bit processor always reads 8 bytes at a time.
- This concept is very useful to increase the processor speed.
- To make use of this advantage, memory is arranged as a group of 4 bytes in 32 bit processor and 8 bytes in 64 bit processor.
- Below C program is compiled and executed in 32 bit compiler. Please check memory allocated for structure1 and structure2 in below program.

```c
#include <stdio.h>
#include <string.h>

/*  Below structure1 and structure2
are same.
They differ only in member's
allignment */

struct structure1
{
       int id1;
       int id2;
       char name;
       char c;
       float percentage;
};

struct structure2
{
       int id1;
       char name;
       int id2;
       char c;
       float percentage;
};
int main()
{
    struct structure1 a;
    struct structure2 b;
```

```c
printf("size of structure1 in bytes : %d\n", sizeof(a));
    printf ( "\n   Address of id1        = %u", &a.id1 );
    printf ( "\n   Address of id2        = %u", &a.id2 );
    printf ( "\n   Address of name       = %u", &a.name );
    printf ( "\n   Address of c          = %u", &a.c );
    printf ( "\n   Address of percentage = %u",&a.percentage );

    printf("   \n\nsize of structure2 in bytes : %d\n",sizeof(b));
    printf ( "\n   Address of id1        = %u", &b.id1 );
    printf ( "\n   Address of name       = %u", &b.name );
    printf ( "\n   Address of id2        = %u", &b.id2 );
    printf ( "\n   Address of c          = %u", &b.c );
    printf ( "\n   Address of percentage = %u",&b.percentage );
    getchar();
    return 0;
}
```

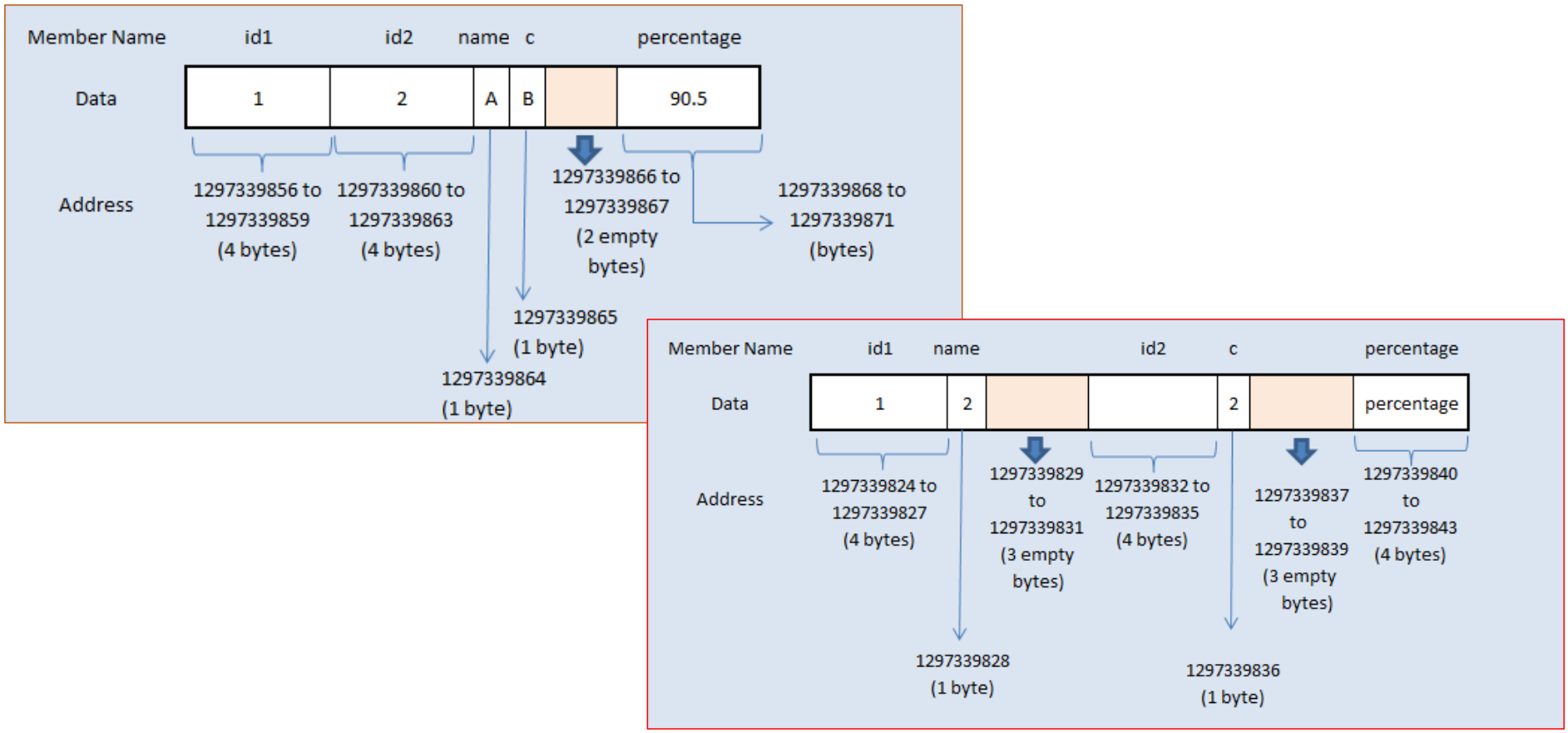
```
size of structure1 in bytes : 16

Address of id1 = 1297339856
Address of id2 = 1297339860
Address of name = 1297339864
Address of c = 1297339865
Address of percentage = 1297339868

size of structure2 in bytes : 20

Address of id1 = 1297339824
Address of name = 1297339828
Address of id2 = 1297339832
Address of c = 1297339836
Address of percentage = 1297339840
```

C – Structure Padding



How to avoid structure padding in C?

- `#pragma pack ( 1 )` directive can be used for arranging memory for structure members very next to the end of other structure members.

- Please check the below program where there will be no addresses (bytes) left empty because of structure padding.

- A pragma (from the Greek word meaning action) is used to direct the actions of the compiler in particular ways, but has no effect on the semantics of a program (in general).

- `#pragma pack (1) , one byte`
- `#pragma pack (2) , two bytes`
- `#pragma pack (4) , four bytes`
- `#pragma pack (8) , eight bytes`

```c
#include <stdio.h>
#include <string.h>

#pragma pack(1)
struct structure1
{
        int id1;
        int id2;
        char name;
        char c;
        float percentage;
};

struct structure2
{
        int id1;
        char name;
        int id2;
        char c;
        float percentage;
};
```

```c
int main()
{
    struct structure1 a;
    struct structure2 b;

    printf("size of structure1 in bytes : %d\n",
                    sizeof(a));
    printf ( "\n   Address of id1        = %u", &a.id1 );
    printf ( "\n   Address of id2        = %u", &a.id2 );
    printf ( "\n   Address of name       = %u", &a.name );
    printf ( "\n   Address of c          = %u", &a.c );
    printf ( "\n   Address of percentage = %u", &a.percentage );

    printf("   \n\nsize of structure2 in bytes : %d\n",
                    sizeof(b));
    printf ( "\n   Address of id1        = %u", &b.id1 );
    printf ( "\n   Address of name       = %u", &b.name );
    printf ( "\n   Address of id2        = %u", &b.id2 );
    printf ( "\n   Address of c          = %u", &b.c );
    printf ( "\n   Address of percentage = %u", &b.percentage );
    getchar();
    return 0;
}
```

```
size of structure1 in bytes : 14
Address of id1 = 3438103088
Address of id2 = 3438103092
Address of name = 3438103096
Address of c = 3438103097
Address of percentage = 3438103098

size of structure2 in bytes : 14
Address of id1 = 3438103072
Address of name = 3438103076
Address of id2 = 3438103077
Address of c = 3438103081
Address of percentage = 3438103082
```

- **Try other pragma values and see what is the output?**

Exercise

- **Try other pragma values and see what is the output?**
- **#pragma pack (1) , one byte**
- **#pragma pack (2) , two bytes**
- **#pragma pack (4) , four bytes**
- **#pragma pack (8) , eight bytes**