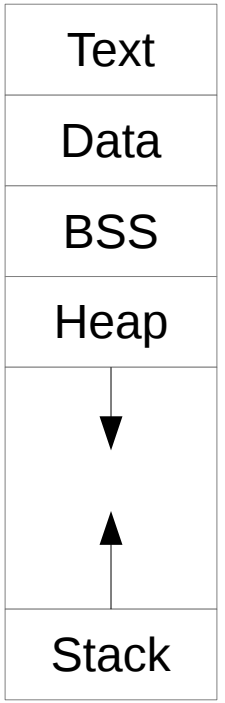
Memory Allocation

Memory

- What is memory?
  - Storage for variables, data, code etc.
- How is memory organized?
  - Text (Code)
  - Data (Constants)
  - BSS (Global and static variables)
  - Stack (Local variables)
  - Heap (Dynamic Memory)



Memory Allocation

```
int iSize;

char *f(){
    char *p;
    iSize = 8;
    p = malloc(isize);
    return p;
}
```

BSS

Stack

Data

Heap

Memory Allocation

- How memory is allocated ?
  - Global and static variables = program start up
  - Local variables = function call
  - Dynamic memory = malloc() / calloc()

Memory Allocation

```
int iSize;

char *f(){
    char *p;
    iSize = 8;
    p = malloc(isize);
    return p;
}
```

Allocated in BSS, set to 0 at startup

Allocated on Stack at start of function f

Allocated in Heap by malloc

Memory Deallocation

- How memory is deallocated ?
  - Global and static variables = program finish
  - Local variables = function return
  - Dynamic memory = free()
- All memory is deallocated at program termination
  - Good practice to use free()

Memory Initialization

- Local variables do not have any initialization
- Memory allocated by malloc has no initialized value
- Memory allocated by calloc is initialized to 0 by default
- Global and static variables are initialized to 0 by default

Two new data types

- Structure
- Union

Structure

- A variable in C can hold only one data of one type (eg. int a, float b, char c etc.)
- An array can hold group of data of same data types (eg. int a[5])
- What is structure ?
  - Collection of different data types

Structure

- Syntax

```
struct point{
    int x;
    int y;
};
```

- Declaring structure variable
  - struct point p;
- Declaring structure using pointer variable
  - struct point *p;

Structure

- Use typedef to rename a data type
  - struct point p1;
  - struct point p2;

    or
  - typedef struct point point;
  - point p1;
  - point p2;

Structure Example

```
#include <stdio.h>
#include <string.h>
typedef struct student{
        int id;
        char name[20];
        float percentage;
}student;
int main() {
        student record = {0}; //Initializing to null
        record.id=1;
        strcpy(record.name, "XYZ");
        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;
}
```

Structure Example

```
#include <stdio.h>
#include <string.h>
struct student {
        int id;
        char name[20];
        float percentage;
};
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);
}
int main() {
        struct student record;
        record.id=1;
        strcpy(record.name, "XXX");
        record.percentage = 86.5;
        func(&record);
        return 0;
}
```

Structure memory allocation

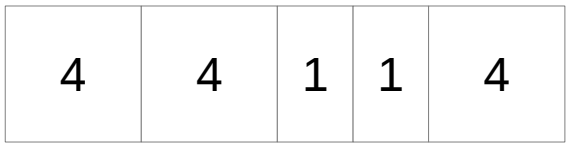
```
#include <stdio.h>
#include <string.h>

/*  Below structure1 and structure2 are same.     They differ only in member's allignment */
struct structure1 {
        int age;
        int id;
        char nameFirstLetter;
        char surnameFirstLetter;
        float percentage;
};

struct structure2 {
        int age;
        char nameFirstLetter;
        int id;
        char surnameFirstLetter;
        float percentage;
};
```

Structure memory allocation

```
struct structure1 {
        int age;
        int id;
        char nameFirstLetter;
        char surnameFirstLetter;
        float percentage;
};
```

```
struct structure2 {
        int age;
        char nameFirstLetter;
        int id;
        char surnameFirstLetter;
        float percentage;
};
```

What is expected ?



Structure memory allocation

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

    printf("size of structure1 in bytes : %d\n",  sizeof(a));
    printf ( "\n   Address of age        = %u", &a.age );
    printf ( "\n   Address of id        = %u", &a.id );
    printf ( "\n   Address of nameFirstLetter       = %u", &a.nameFirstLetter );
    printf ( "\n   Address of surnameFirstLetter          = %u", &a.surnameFirstLetter );
    printf ( "\n   Address of percentage = %u",   &a.percentage );

    printf("   \n\nsize of structure2 in bytes : %d\n",   sizeof(b));
    printf ( "\n   Address of age        = %u", &b.age );
    printf ( "\n   Address of nameFirstLetter       = %u", &b.nameFirstLetter );
    printf ( "\n   Address of id        = %u", &b.id );
    printf ( "\n   Address of surnameFirstLetter          = %u", &b.surnameFirstLetter );
    printf ( "\n   Address of percentage = %u",  &b.percentage );
    getchar();
    return 0;
}
```

Structure memory allocation

- Output:

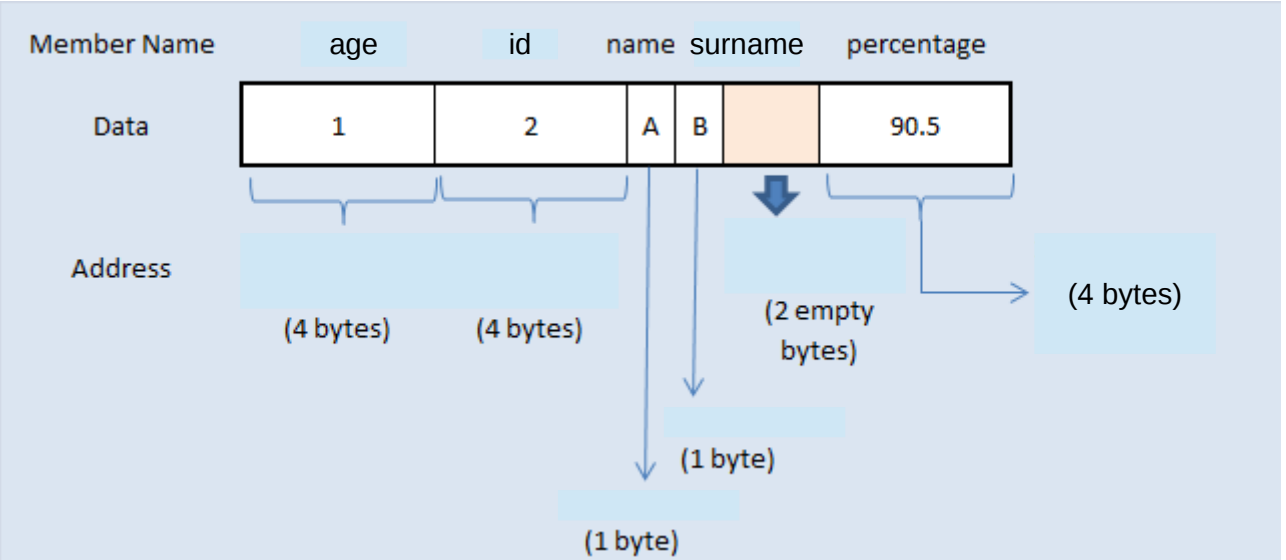
size of structure1 in bytes : 16

Address of age = 3057402288
Address of id = 3057402292
Address of nameFirstLetter = 3057402296
Address of surnameFirstLetter = 3057402297
Address of percentage = 3057402300

size of structure2 in bytes : 20

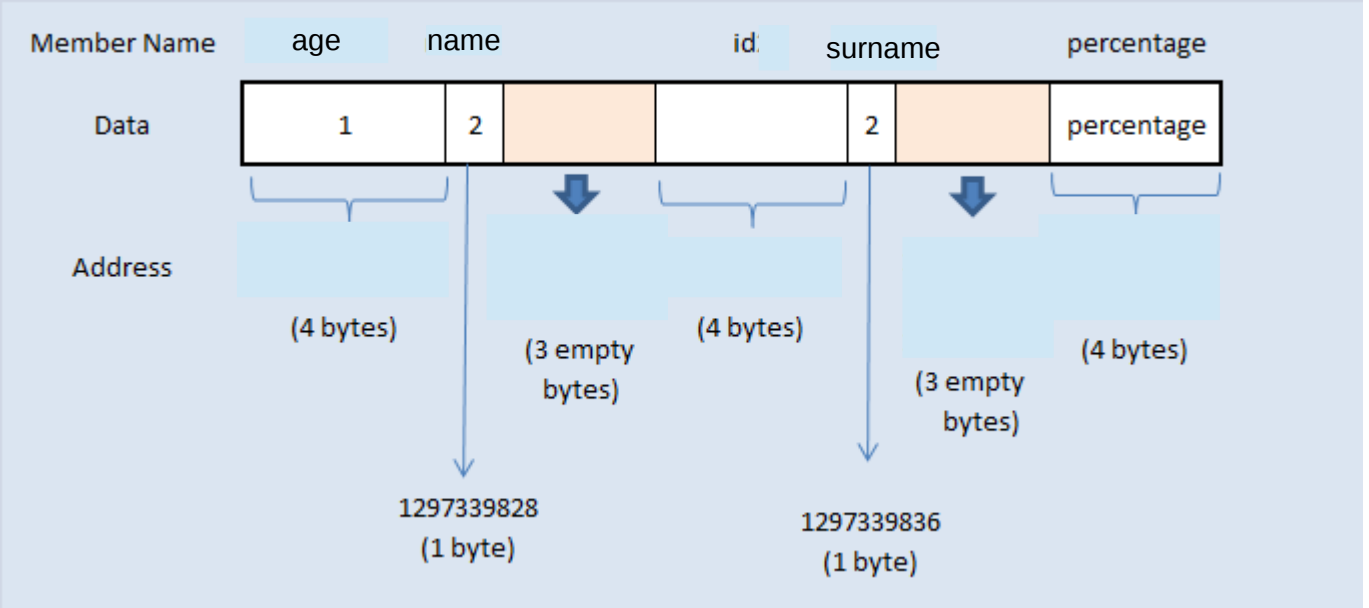
Address of age = 3057402256
Address of nameFirstLetter = 3057402260
Address of id = 3057402264
Address of surnameFirstLetter = 3057402268
Address of percentage = 3057402272

Structure memory allocation




```
struct structure1 {
        int age;
        int id;
        char nameFirstLetter;
        char surnameFirstLetter;
        float percentage;
};
```

Structure memory allocation




```
struct structure2 {
        int age;
        char nameFirstLetter;
        int id;
        char surnameFirstLetter;
        float percentage;
};
```

Union

- C Union is also like structure, collection of different data types
- Each element in a union is called member.
- Union and structure in C are same in concepts, except allocating memory for their members.
  - Structure allocates storage space for all its members separately
  - Union allocates one common storage space for all its members
- We can access only one member of union at a time
  - This is because, Union allocates one common storage space for all its members.
- Many union variables can be created in a program and memory will be allocated for each union variable separately

Union

```
union student{
    int mark;
    char name[6];
    double average;
};
```

- For above union, only 8 bytes of memory will be allocated since double data type will occupy maximum space of memory over other data types.

  Total memory allocation = 8 Bytes

Command Line Argument

Command Line Argument

int main(int argc, char **argv){}

- argc : count of argument passing through command line
- argv : arguments
- By default

  argc = 1;

  argv[0] = the name by which the program was called

Command Line Argument

```
#include <stdio.h>
#include <string.h>
int main(int argc, char **argv){
    int i;
    printf ("%d", argc);
    for (i = 0; i < argc; i++)
    printf ("\nargv[%d]%s", i, argv[i]);
    printf("\n");
}
```

Run : ./a.out Indian Statistical Institute

Example of Union

```
void testEndian(){
    union xx{
        int myInt;
        char val;
    } xunion;
    xunion.myInt = 1;
    if (xunion.val)
        printf (“Little Endian”);
    else
        printf (“Big endian”);
}
```