Operations on Structures

Using Typedef

- The other way is to use `typedef`.
- E.g.: structure variables of struct type `myType` can be declared as follows.

```
typedef struct {
    int number;
    char name[MAXLEN+1];
    int extra;
} myType;  // myType can be used as a conventional type

myType s1, s2; // prefix struct not required now
```

Structure as Arguments

Example

```
typedef struct { // Defining a structure as a type
    char name[MAXLEN+1];
    char sex;
    int age;
    int code;
} part; // Comma required

part construct(char *name, char sex, int age, int code) {
    part p;

    // Create structure with input parameters
    strcpy(p.name, name);
    p.age = age;
    p.code = code;
    p.sex = sex;
    return p; // Return the structure
}
```

Structure as Arguments

Example

```
int readLine(char str[]) { // Reading a string for name
    int ch, i = 0;
    while ((ch = getchar()) != '\n')
        if (i < MAXLEN)
            str[i++] = ch;
    str[i] = '\0';
    return i;
}
void printStruct(part p) {
    printf("%s    ", p.name);
    printf("%c    ", p.sex);
    printf("%d    ", p.age);
    printf("%d    \n", p.code);
}
```

Structure as Arguments

Example

```
int main() {
    char name[MAXLEN+1], s;
    int a, c;
    part record;

    printf("Enter name: ");
    readLine(name);
    printf("Enter gender: ");
    scanf("%c", &s);
    printf("Enter age: ");
    scanf("%d", &a);
    printf("Enter code: ");
    scanf("%d", &c);
    record = construct(name, s, a, c);
    printStruct(record);
}
```

Dynamic Storage Allocation

Importance of Dynamic Allocation

- Normally, C data structures are of fixed size
- Once a program has been compiled, the length of an array is fixed.
- Though the size of a variable length array is determined at runtime, but remains fixed during the execution.
- In many cases it is not possible to anticipate the size in advance.
- Ideally data structures should grow and shrink according to program requirements during execution.
- C supports **dynamic allocation**.

Dynamic Storage Allocation

Memory Allocation Functions

- Functions are available in `stdlib.h`.
- Three main functions defined therein are:
  - `malloc`: allocates a block of unitialized memory.
  - `calloc`: allocates a block of memory and clears it.
  - `realloc`: resize previously allocated block of memory.
- Since he data types to be stored by user not know in advance, allocation functions return void or generic pointers to allocated blocks.

Allocation for Strings

Using Malloc to Allocate for a String

```c
#include <time.h>
#include <stdio.h>
#include <stdlib.h>
char *allocString(int n) {
    char *buf;
    buf = malloc(n + 1);
    return buf;
}
int main() {
    int i, len;
    char *s;

    /*** Rest of the program ***/
}
```

Allocation for Strings

Using Malloc to Allocate for a String

```
printf("Enter length of string needed: ");
scanf("%d", &len);

s = allocString(len);

srand((unsigned int) time(NULL));

for (i = 0; i < len; i++)
    s[i] = rand() % 26 + 'a';
s[len] = '\0';
printf("%s\n", s);
free(s);
```