

PROJECT STUDY ON EMBEDDED - C

A Major-project report submitted in partial fulfillment of the requirement for degree of

BACHELOR OF TECHNOLOGY

In

COMPUTER SCIENCE & ENGINEERING

By

K.Mahendra Reddy(R171122)

Under the guidance of
Mr.K.Vinod Kumar

Asst.Prof. In Department of Computer Science & Engineering



AP IIIT, RGUKT-RK Valley,
Vempalli, Kadapa(Dist), Andhra Pradesh-516330, India
September 2022-February 2023



**RAJIV GANDHI UNIVERSITY OF KNOWLEDGE
TECHNOLOGIES**

(A.P.Government Act 18 of 2008)

RGUKT-RK Valley

Vempalli,Kadapa,Andhrapradesh-516330.

CERTIFICATE OF EXAMINATION

This is to certify that I,K.Mahendra Reddy(R171122) have learned concepts of Embedded-C and hereby accord my approval of it as a study carried out and presented in a manner required for its acceptance in partial fulfillment for the award of Bachelor of Technology degree for which it has been submitted.This approval does not necessarily endorse or accept every statement made ,opinion expressed or conclusions drawn,as recorded in this thesis.It only signifies the acceptance of this thesis for the purpose for which it has been submitted.

EXAMINER



**RAJIV GANDHI UNIVERSITY OF KNOWLEDGE
TECHNOLOGIES**

(A.P.Government Act 18 of 2008)

RGUKT-RK Valley

Vempalli,Kadapa,Andhrapradesh-516330.

CERTIFICATE OF PROJECT COMPLETION

This is to certify that I,K.Mahendra Reddy(R171122) have learned concepts of Embedded-C under our guidance and supervision for the partial fulfillment for the degree of Bachelor of Technology in Computer Science and Engineering during the academic session September 2022 – February 2023 at RGUKT-RK Valley

To the best of my knowledge , the results embodied in this dissertation work have not been submitted to any university or institute for thw award of any degree or diploma.

Mr.K.Vinod Kumar,
Lecturer in Computer Science & Engg.
RGUKT-RK Valley.

Mr.N.Satyanandaram,
Asst.Prof. In Computer Science & Engg,
Head of the Department,
RGUKT-RK Valley.



**RAJIV GANDHI UNIVERSITY OF KNOWLEDGE
TECHNOLOGIES**

(A.P.Government Act 18 of 2008)

RGUKT-RK Valley

Vempalli,Kadapa,Andhrapradesh-516330.

DECLARATION

I,K.Mahendra Reddy(R171122) have learned concepts of Embedded-C under guidance of Mr.K.Vinod Kumar is submitted in partial fulfillment for the degree of Bachelor of Technology in Computer Science and Engineering during the academic session September 2022 – February 2023 at RGUKT-RK Valley.we also declare that this project is a result of our own effort and has not been copied or imitated from any source.Citations from any websites are mentioned in the references.To the best of my knowledge , the results embodied in this dissertation work have not been submitted to any university or institute for thw award of any degree or diploma.

K.Mahendra Reddy (R171122)

ACKNOWLEDGEMENT

We would like to express our deep sense of gratitude & respect to all those people behind the screen who guided, inspired and helped us crown all our efforts with success. We wish to express our gratitude to Mr.K.Vinod Kumar for his valuable guidance at all stages of study, advice, constructive suggestions, supportive attitude and continuous encouragement, without which it would not be possible to complete this project.

We would also like to extend our deepest gratitude & reverence to the Director of RGUKT, RK Valley **Prof.K.Sandhyarani** and HOD of Computer Science and Engineering **Mr.N.Satyanandaram** for their constant support and encouragement.

Last but not least we express our gratitude to our parents for their constant source of encouragement and inspiration for us to keep our morals high.

Table of Contents

1.Bit fields.....	7-12
2.Bit Manipulation.....	13-14
3.Pointers.....	15-19
4.Conclusion.....	20

Abstract

Embedded systems work on a single functionality for example washing machines. So their RAM is very less like 10Kb comparing to laptops so in that case memory should manage efficiently in Embedded systems so we use concept of Bit fields. And also they should respond fast so Bit manipulations are very important.

1.Bit Fields

In C,we can specify the size (in bits) of the structure and union members. The idea of bit-field is to use memory efficiently when we know that the value of a field or group of fields will never exceed a limit or is within a small range.Bit fields are used when the storage of our program is limited. Need of bit fields in C programming language:

- Reduces memory consumption.
- To make our program more efficient and flexible.
- Easy to Implement.

Applications of Bit-fields:

- If storage is limited, we can go for bit-field.
- When devices transmit status or information encoded into multiple bits for this type of situation bit-field is most efficient.
- Encryption routines need to access the bits within a byte in that situation bit-field is quite useful.

PROGRAM – 1(Without bit fields)

```
#include <stdio.h>

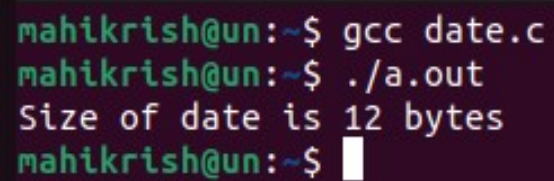
// A simple representation of the date
struct date {
    int d;
    int m;
    int y;
};

int main()
{
    printf("Size of date is %lu bytes\n", sizeof(struct date));
    struct date dt = { 31, 12, 2014 };
```



```
    printf("Date is %d/%d/%d", dt.d, dt.m, dt.y);  
}
```

OUTPUT:

A terminal window with a dark purple background. The prompt is 'mahikrish@un:~\$'. The first command is 'gcc date.c', the second is './a.out', and the output is 'Size of date is 12 bytes'.

```
mahikrish@un:~$ gcc date.c  
mahikrish@un:~$ ./a.out  
Size of date is 12 bytes  
mahikrish@un:~$
```

The above representation of 'date' takes 12 bytes on a compiler whereas an unsigned int takes 4 bytes. Since we know that the value of d is always from 1 to 31, and the value of m is from 1 to 12, we can optimize the space using bit fields.

Declaration of bit-fields in C

Bit-fields are variables that are defined using a predefined width or size. Format and the declaration of the bit-fields in C are shown below:

Syntax:

```
struct
{
    data_type member_name: width_of_bit-field;
};
```

Example:

```
struct date
{
    // month has value between 0 and 15,
    // so 4 bits are sufficient for month variable.
    int month : 4;
};
```

However, if the same code is written using signed int and the value of the fields goes beyond the bits allocated to the variable and something interesting can happen. For example, consider the same code but with signed integers:

PROGRAM – 2(Using bit fields)

```
// C program to demonstrate use of Bit-fields
#include <stdio.h>

// Space optimized representation of the date
struct date {
    // d has value between 0 and 31, so 5 bits
```

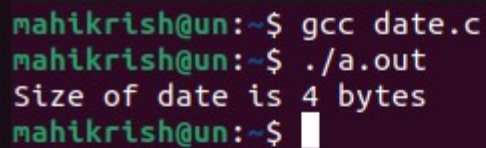
```
// are sufficient
unsigned int d : 5;

// m has value between 0 and 15, so 4 bits
// are sufficient
unsigned int m : 4;

//year cannot be more than 4096 so 11 bits enough
unsigned int y : 11;
};

int main()
{
    printf("Size of date is %lu bytes\n",
           sizeof(struct date));
    struct date dt = { 31, 12, 2014 };
    printf("Date is %u/%u/%u", dt.d, dt.m, dt.y);
    return 0;
}
```

OUTPUT :



```
mahikrish@un:~$ gcc date.c
mahikrish@un:~$ ./a.out
Size of date is 4 bytes
mahikrish@un:~$
```

Conclusion:

So,by using bit fields we can reduce size of memory here we reduced size from 12bytes to 4bytes which is very useful for embedded systems.

2.Bit Manipulation:

Bit Manipulation is a technique used in a variety of problems to get the solution in an optimized way. This technique is very effective from a Competing programming point of view. It is all about Bitwise Operators which directly works upon binary numbers or bits of numbers that help the implementation fast. Below are the **Bitwise Operators** that are used:

- Bitwise AND (&)
- Bitwise OR (|)
- Bitwise XOR (^)
- Bitwise NOT (!)

All data in computer programs are internally stored as bits, i.e., as numbers 0 and 1.

PROGRAM – 1

// C program to check even or odd number using bitwise operator

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

```
int main()
```

```
{
```

```
    int num;
```

```
    /* Input number from user */
```

```
    printf("Enter any number: ");
```

```
    scanf("%d", &num);
```

```
    if(num & 1)
```

```
    {
```

```
        printf("%d is odd.", num);
```

```
    }
```

```
    else
```

```
    {
```

```
        printf("%d is even.", num);
```

```
    }
```

```
    return 0;
```

```
}
```

```
/*C program to swap two nibbles of a given byte*/
```

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

```
void dec_bin(unsigned int num)
```

```
{
```

```
    for(int i=7;i>=0;i--)
```

```
        (num&(1<<i)) ? printf("1") : printf("0");
```

```
    printf("\n");
```

```
}
```

```
int main()
```

```
{
```

```
    unsigned int key;
```

```
    printf("Enter number:");
```

```
    scanf("%i",&key);
```

```
    printf("Before swapping nibbles are :");
```

```
    dec_bin(key);
```

```
    key=((key>>4)|(key<<4));
```

```
    printf("\nAfter swapping nibbles are :");
```

```
    dec_bin(key);
```

```
    return 0;
```

```
}
```

OUTPUT:

```
Enter number:120
Before swapping nibbles are :01111000

After swapping nibbles are :10000111

...Program finished with exit code 0
```

3.Pointers

Pointers in C are used to store the address of variables or a memory location. This variable can be of any data type i.e, int, char, function, array, or any other pointer. Pointers are one of the core concepts of C programming language that provides low-level memory access and facilitates dynamic memory allocation.

What is a Pointer in C?

A pointer is a derived data type in C that can store the address of other variables or a memory. We can access and manipulate the data stored in that memory location using pointers.

Syntax of C Pointers

```
datatype * pointer_name;
```

The above syntax is the generic syntax of C pointers. The actual syntax depends on the type of data the pointer is pointing to.

How to Use Pointers?

To use pointers in C, we must understand below two operators

1. Addressof Operator

The addressof operator (&) is a unary operator that returns the address of its operand. Its operand can be a variable, function, array, structure, etc.

Syntax of Address of Operator

```
&variable_name;
```

2. Dereferencing Operator

The dereference operator (*), also known as the indirection operator is a unary operator. It is used in pointer declaration and dereferencing.

C Pointer Declaration

In pointer declaration, we only declare the pointer but do not initialize it. To declare a pointer, we use the * dereference operator before its name.

```
data_type * pointer_name;
```

The pointer declared here will point to some random memory address as it is not initialized. Such pointers are also called wild pointers that we will study later in this article.

C Pointer Initialization

When we assign some value to the pointer, it is called Pointer Initialization in C. There are two ways in which we can initialize a pointer in C of which the first one is:

Method 1: C Pointer Definition

```
datatype * pointer_name = address;
```

The above method is called Pointer Definition as the pointer is declared and initialized at the same time.

Method 2: Initialization After Declaration

The second method of pointer initialization in C is assigning some address after the declaration.

```
datatype * pointer_name;  
pointer_name = addresss;
```

Dereferencing a C Pointer

Dereferencing is the process of accessing the value stored in the memory address specified in the pointer. We use dereferencing operator for that purpose.

Dereferencing a Pointer in C

PROGRAM - 1

```
// C program to illustrate Pointers
```

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

```
void display()
```

```
{
```

```
    int var = 20;
```

```
    // declare pointer variable
```



```
int* ptr;

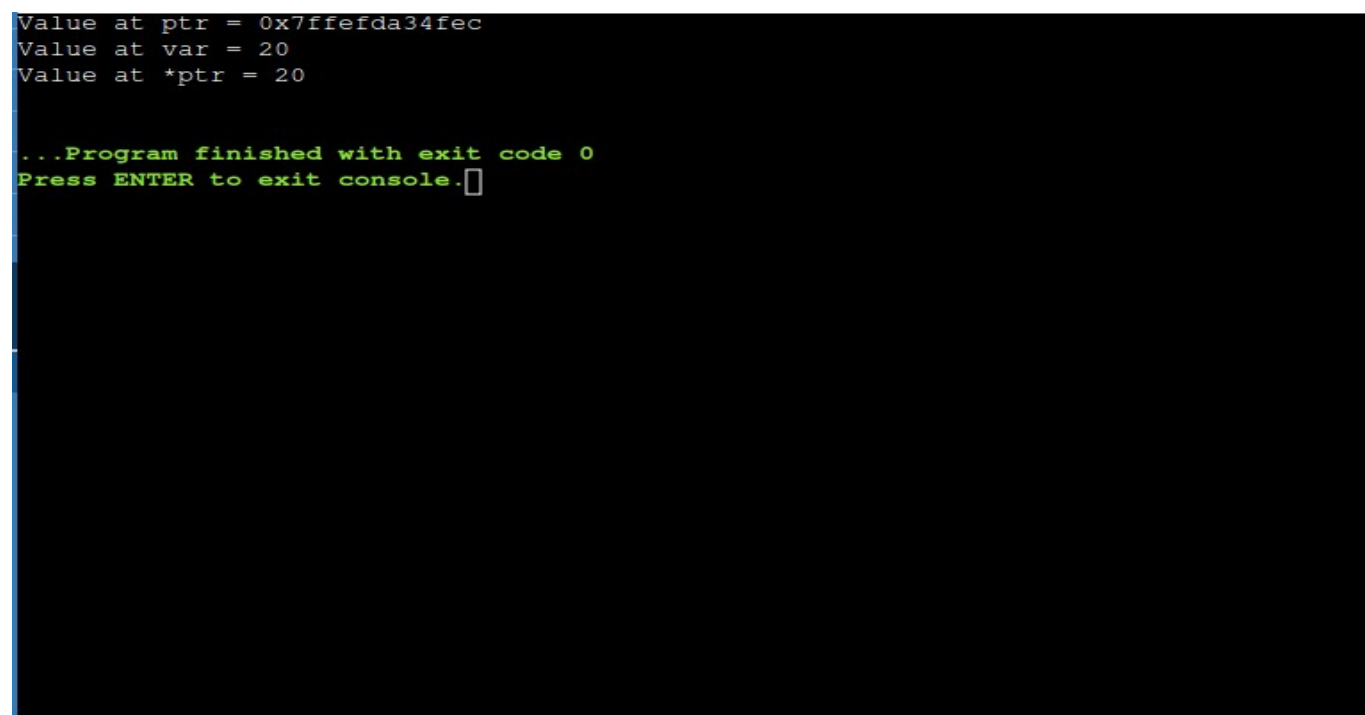
// note that data type of ptr and var must be same

ptr = &var;

// assign the address of a variable to a pointer
printf("Value at ptr = %p \n", ptr);
printf("Value at var = %d \n", var);
printf("Value at *ptr = %d \n", *ptr);
}

// Driver program
int main()
{
    display();
    return 0;
}
```

OUTPUT:



```
Value at ptr = 0x7ffefda34fec
Value at var = 20
Value at *ptr = 20

...Program finished with exit code 0
Press ENTER to exit console.□
```

Pointer Arithmetic

Only a limited set of operations can be performed on pointers. The Pointer Arithmetic refers to the legal or valid operations that can be performed on a pointer. It is slightly different from the ones that we generally use for mathematical calculations.

The operations are:

- Increment in a Pointer
- Decrement in a Pointer
- Addition of integer to a pointer
- Subtraction of integer to a pointer
- Subtracting two pointers of the same type
- Comparison of pointers of the same type.
- Assignment of pointers of the same type.

PROGRAM 2

// C program to illustrate Pointer Arithmetic

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

```
int main()
{
```

```
    // Declare an array
```

```
    int v[3] = { 10, 100, 200 };
```

```
    // Declare pointer variable
```

```
    int* ptr;
```

```
    // Assign the address of v[0] to ptr
```

```
    ptr = v;
```

```
    for (int i = 0; i < 3; i++) {
```

```
        // print value at address which is stored in ptr
```

```
        printf("Value of *ptr = %d\n", *ptr);
```

```
        // print value of ptr
```

```
        printf("Value of ptr = %p\n\n", ptr);
```

```
        // Increment pointer ptr by 1
        ptr++;
    }
    return 0;
}
```

OUTPUT

```
Value of *ptr = 10
Value of ptr = 0x7fffe9861d9c

Value of *ptr = 100
Value of ptr = 0x7fffe9861da0

Value of *ptr = 200
Value of ptr = 0x7fffe9861da4

...Program finished with exit code 0
Press ENTER to exit console.█
```

Conclusion:

So,we can conclude that concepts like Bit Manipulation important for faster execution of program as they work directly on low level,Bit fields to use memory efficiently and Pointers work on addresses of data.