# Assignment -1

1. In this problem we will learn about passing values as parameter through assembly language

Ques1.c

```c
#include<stdio.h>
 int fun(int a,int b,int c,int d,int e){
        int f=a+b+c+d+e;
        return f;
 }
void main(){
        int a=10,b=20,c=30,d=40,e=50;
        int p=fun(a,b,c,d,e);
        printf("%d",p);
}
```

For the above program Assembly code is as follows

```
fun:
.LFB0:
        .cfi_startproc
        endbr64
        pushq   %rbp
        .cfi_def_cfa_offset 16
        .cfi_offset 6, -16
        movq    %rsp, %rbp
        .cfi_def_cfa_register 6
        movl    %edi, -20(%rbp)
        movl    %esi, -24(%rbp)
        movl    %edx, -28(%rbp)
        movl    %ecx, -32(%rbp)
        movl    %r8d, -36(%rbp)
        movl    -20(%rbp), %edx
        movl    -24(%rbp), %eax
        addl    %eax, %edx
        movl    -28(%rbp), %eax
        addl    %eax, %edx
        movl    -32(%rbp), %eax
        addl    %eax, %edx
        movl    -36(%rbp), %eax
        addl    %edx, %eax
        movl    %eax, -4(%rbp)
        movl    -4(%rbp), %eax
```

All the values passed are stored the following registers such as edi,esi,edx,ecx,r8d.To check for more detail we will have to look

Ques1.c

```
50          .cfi_def_cfa_register 6
51          subq    $32, %rsp
52          movl    $10, -24(%rbp)
53          movl    $20, -20(%rbp)
54          movl    $30, -16(%rbp)
55          movl    $40, -12(%rbp)
56          movl    $50, -8(%rbp)
57          movl    -8(%rbp), %edi
58          movl    -12(%rbp), %ecx
59          movl    -16(%rbp), %edx
60          movl    -20(%rbp), %esi
61          movl    -24(%rbp), %eax
62          movl    %edi, %r8d
63          movl    %eax, %edi
64          call    fun
65          movl    %eax, -4(%rbp)
66          movl    -4(%rbp), %eax
67          movl    %eax, %esi
68          leaq    .LC0(%rip), %rdi
69          movl    $0, %eax
70          call    printf@PLT
71          nop
72          leave
73          .cfi_def_cfa 7, 8
74          ret
75          .cfi_endproc
```

Statement 62 and 63 shows the movement of actual parameters to formal parameters.

In statement 64 the main function calls the fun which pushes the eip(stack pointer) which contains the return address of the called function.

2.This problem is solved using a pointer in c.Pointer is a special data structure that stores the address of another variable. Here we can see that addresses are passed as reference to calling functions in which pointers are declared that store the address passed.

```c
#include<stdio.h>
 int fun(int *a,int *b,int *c,int *d,int *e){
        int f=*a+*b+*c+*d+*e;
        return f;
 }
void main(){
        int a=10,b=20,c=30,d=40,e=50;
        int p=fun(&a,&b,&c,&d,&e);
        printf("%d",p);
}
```

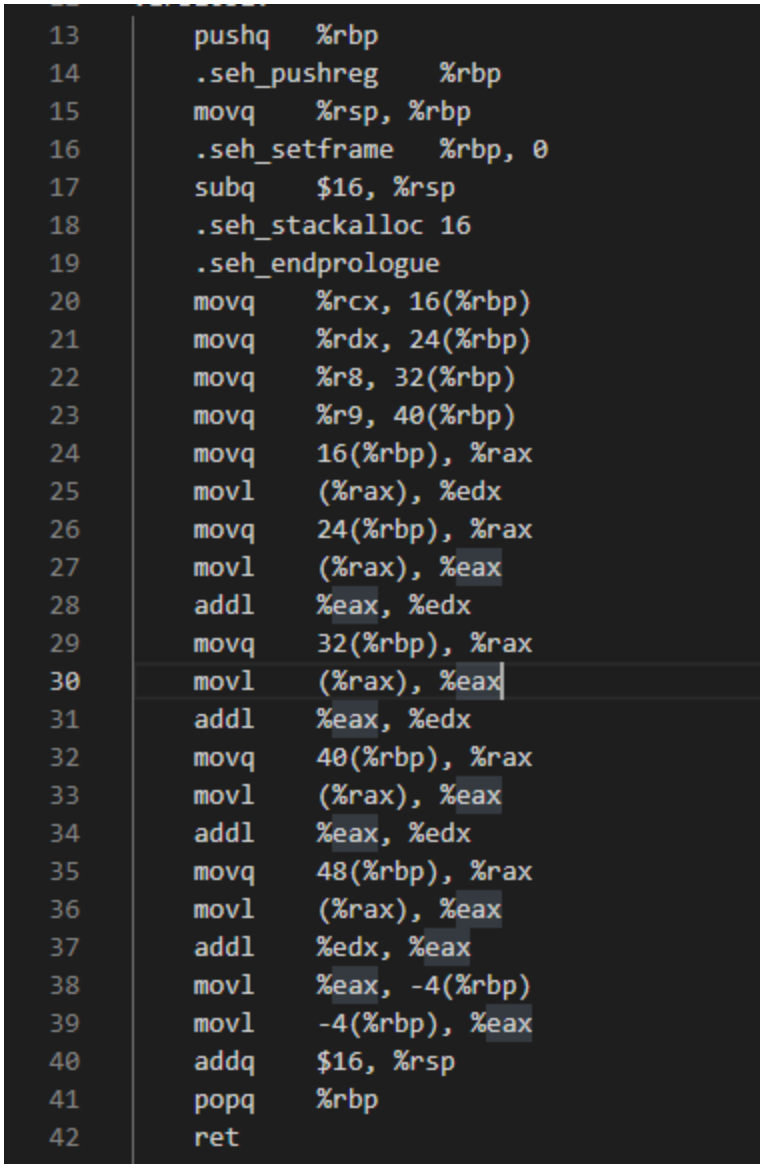
Here all the five variables are integer type and their address are passed in pointer variables.we can see the assembly code of this

```
fun:
    pushq   %rbp
    .seh_pushreg    %rbp
    movq    %rsp, %rbp
    .seh_setframe   %rbp, 0
    subq    $16, %rsp
    .seh_stackalloc 16
    .seh_endprologue
    movq    %rcx, 16(%rbp)
    movq    %rdx, 24(%rbp)
    movq    %r8, 32(%rbp)
    movq    %r9, 40(%rbp)
    movq    16(%rbp), %rax
    movl    (%rax), %edx
    movq    24(%rbp), %rax
    movl    (%rax), %eax
    addl    %eax, %edx
    movq    32(%rbp), %rax
    movl    (%rax), %eax
    addl    %eax, %edx
    movq    40(%rbp), %rax
    movl    (%rax), %eax
    addl    %eax, %edx
    movq    48(%rbp), %rax
    movl    (%rax), %eax
    addl    %edx, %eax
    movl    %eax, -4(%rbp)
    movl    -4(%rbp), %eax
    addq    $16, %rsp
    popq    %rbp
    ret
    .seh_endproc
```

main():

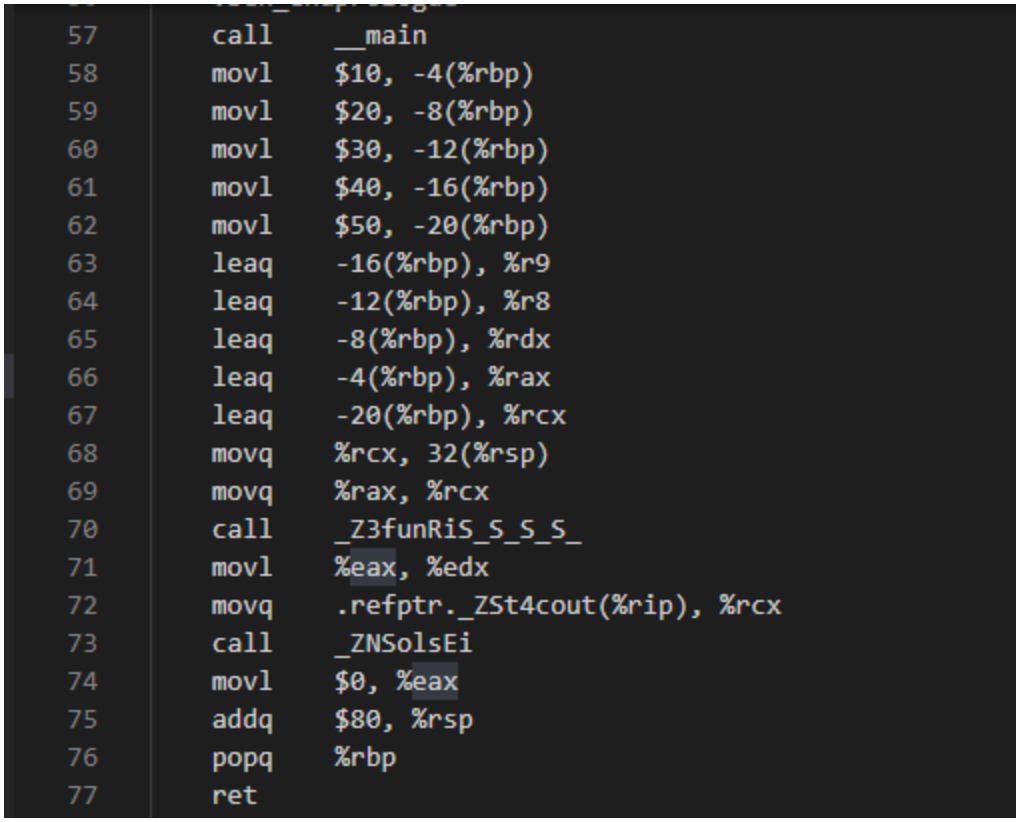
```
main:
    pushq   %rbp
    .seh_pushreg    %rbp
    movq    %rsp, %rbp
    .seh_setframe   %rbp, 0
    subq    $80, %rsp
    .seh_stackalloc 80
    .seh_endprologue
    call    __main
    movl    $10, -4(%rbp)
    movl    $20, -8(%rbp)
    movl    $30, -12(%rbp)
    movl    $40, -16(%rbp)
    movl    $50, -20(%rbp)
    leaq    -16(%rbp), %r9
    leaq    -12(%rbp), %r8
    leaq    -8(%rbp), %rdx
    leaq    -4(%rbp), %rax
    leaq    -20(%rbp), %rcx
    movq    %rcx, 32(%rsp)
    movq    %rax, %rcx
    call    fun
    movl    %eax, %edx
    leaq    .LC0(%rip), %rcx
    call    printf
    movl    $0, %eax
    addq    $80, %rsp
    popq    %rbp
    ret
```

Using reference variable in our assembly code looks like of fun() as:



```
13      pushq   %rbp
14      .seh_pushreg    %rbp
15      movq    %rsp, %rbp
16      .seh_setframe   %rbp, 0
17      subq    $16, %rsp
18      .seh_stackalloc 16
19      .seh_endprologue
20      movq    %rcx, 16(%rbp)
21      movq    %rdx, 24(%rbp)
22      movq    %r8, 32(%rbp)
23      movq    %r9, 40(%rbp)
24      movq    16(%rbp), %rax
25      movl    (%rax), %edx
26      movq    24(%rbp), %rax
27      movl    (%rax), %eax
28      addl    %eax, %edx
29      movq    32(%rbp), %rax
30      movl    (%rax), %eax
31      addl    %eax, %edx
32      movq    40(%rbp), %rax
33      movl    (%rax), %eax
34      addl    %eax, %edx
35      movq    48(%rbp), %rax
36      movl    (%rax), %eax
37      addl    %edx, %eax
38      movl    %eax, -4(%rbp)
39      movl    -4(%rbp), %eax
40      addq    $16, %rsp
41      popq    %rbp
42      ret
```

Of main() as:



```
57      call    __main
58      movl    $10, -4(%rbp)
59      movl    $20, -8(%rbp)
60      movl    $30, -12(%rbp)
61      movl    $40, -16(%rbp)
62      movl    $50, -20(%rbp)
63      leaq    -16(%rbp), %r9
64      leaq    -12(%rbp), %r8
65      leaq    -8(%rbp), %rdx
66      leaq    -4(%rbp), %rax
67      leaq    -20(%rbp), %rcx
68      movq    %rcx, 32(%rsp)
69      movq    %rax, %rcx
70      call    _Z3funRiS_S_S_
71      movl    %eax, %edx
72      movq    .refptr._ZSt4cout(%rip), %rcx
73      call    _ZNSolsEi
74      movl    $0, %eax
75      addq    $80, %rsp
76      popq    %rbp
77      ret
```

3.
Fixed Stack dynamic arrays:In fixed stack dynamic arrays the size We know the size of array at compile time but its space allocation is done during run time when required
Eg.

```
Void fun(){
    int arr[7];
    }
```

Array is allocated when fun() is called.
Stack Dynamic Arrays: In stack dynamic arrays we don't know the size of the array until run time.
Eg.

```
int n;
cin>>n;
Int arr[n];
```

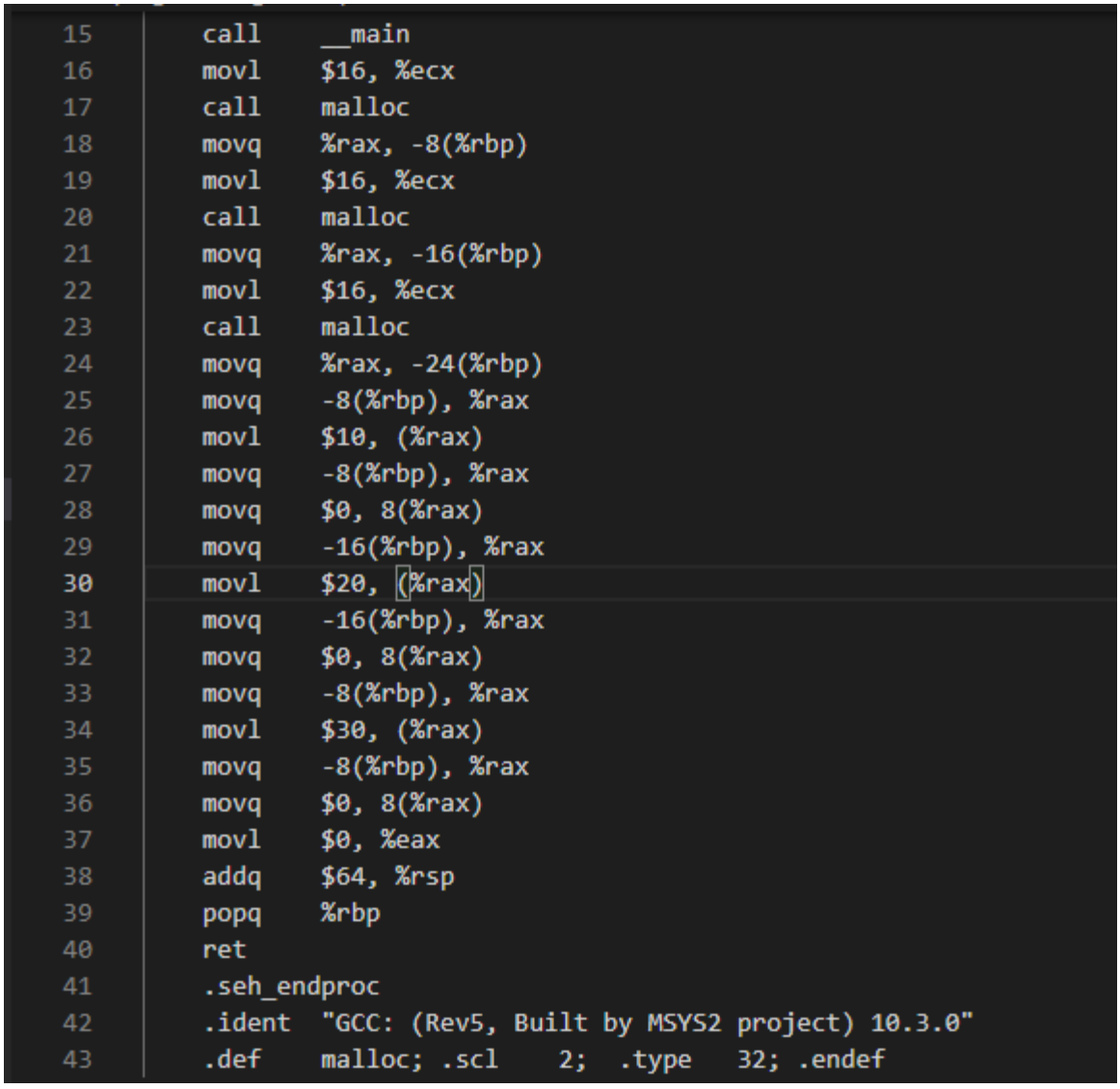
4.Assembly code of heap dynamic variables. Unlike stack here

Memory is created at random location for different variables
Which we can see using gdb;

c++ programming > C problem4.c > main()

```c
#include<stdio.h>
#include <stdlib.h>
struct node{
    int data;
    struct node *next;
};
int main(){
    struct node *temp1= (struct node *)malloc(sizeof(struct node));
    struct node *temp2= (struct node *)malloc(sizeof(struct node));
    struct node *temp3= (struct node *)malloc(sizeof(struct node));
    temp1->data = 10;
    temp1->next = NULL;
    temp2->data = 20;
    temp2->next = NULL;
    temp1->data = 30;
    temp1->next = NULL;
    return 0;
}
```

Here three dynamic variables are declared memory from heap
Section of memory after use they are not free so after the program
Complex execution still also contains values unlike stack based
variables.



```
15      call    __main
16      movl    $16, %ecx
17      call    malloc
18      movq    %rax, -8(%rbp)
19      movl    $16, %ecx
20      call    malloc
21      movq    %rax, -16(%rbp)
22      movl    $16, %ecx
23      call    malloc
24      movq    %rax, -24(%rbp)
25      movq    -8(%rbp), %rax
26      movl    $10, (%rax)
27      movq    -8(%rbp), %rax
28      movq    $0, 8(%rax)
29      movq    -16(%rbp), %rax
30      movl    $20, (%rax)
31      movq    -16(%rbp), %rax
32      movq    $0, 8(%rax)
33      movq    -8(%rbp), %rax
34      movl    $30, (%rax)
35      movq    -8(%rbp), %rax
36      movq    $0, 8(%rax)
37      movl    $0, %eax
38      addq    $64, %rsp
39      popq    %rbp
40      ret
41      .seh_endproc
42      .ident  "GCC: (Rev5, Built by MSYS2 project) 10.3.0"
43      .def    malloc; .scl    2;  .type   32; .endef
```

6.Let us take a factorial program to understand recursion in c/c++.

In assembly code of main we can see that  that actual value of parameter is passed to function via ecx and in assembly code of _Z4facti we can see the function calling itself

c c++ programming > problem6.cpp > fact(int

```cpp
#include<iostream>
using namespace std;
int fact(int n){
    if(n==0 || n==1) return n;
    return n*fact(n-1);
}
int main(){
    int a=5;
    cout<<fact(a);
    return 0;
}
```

```
main:
.LFB1652:
    pushq   %rbp
    .seh_pushreg    %rbp
    movq    %rsp, %rbp
    .seh_setframe   %rbp, 0
    subq    $48, %rsp
    .seh_stackalloc 48
    .seh_endprologue
    call    __main
    movl    $5, -4(%rbp)
    movl    -4(%rbp), %eax
    movl    %eax, %ecx
    call    _Z4facti
    movl    %eax, %edx
    movq    .refptr._ZSt4cout(%rip), %rcx
    call    _ZNSolsEi
    movl    $0, %eax
    addq    $48, %rsp
    popq    %rbp
    ret
```

```
11  _Z4facti:
12  .LFB1651:
13      pushq   %rbp
14      .seh_pushreg    %rbp
15      movq    %rsp, %rbp
16      .seh_setframe   %rbp, 0
17      subq    $32, %rsp
18      .seh_stackalloc 32
19      .seh_endprologue
20      movl    %ecx, 16(%rbp)
21      cmpl    $0, 16(%rbp)
22      je  .L2
23      cmpl    $1, 16(%rbp)
24      jne .L3
25  .L2:
26      movl    16(%rbp), %eax
27      jmp .L4
28  .L3:
29      movl    16(%rbp), %eax
30      subl    $1, %eax
31      movl    %eax, %ecx
32      call    _Z4facti
33      imull   16(%rbp), %eax
34  .L4:
35      addq    $32, %rsp
36      popq    %rbp
37      ret
38      .seh_endproc
39      .def    __main; .scl    2;  .type   32; .endef
40      .globl  main
41      .def    main;   .scl    2;  .type   32; .endef
42      .seh_proc   main
```