"From C To Assembly" Cheat Sheet

Author: oa2013.github.io

Date: July 25, 2021

Sources used:

- Dennis Yurichev's Reverse Engineering For Beginners
- Compiler Explorer (x86-64 gcc 5.1)
- Beginners Book (C Tutorials)
- Geeks for Geeks (C Tutorials)

printf() with arguments

```c
#include <stdio.h>

int main()
{
    printf("a=%d; b=%d; c=%d", 1, 2, 3);
    return 0;
}
```

```
.LC0:
        .string "a=%d; b=%d; c=%d"
main:
        push    rbp
        mov     rbp, rsp
        mov     ecx, 3
        mov     edx, 2
        mov     esi, 1
        mov     edi, OFFSET FLAT:.LC0
        mov     eax, 0
        call    printf
        mov     eax, 0
        pop     rbp
        ret

```

scanf()

```c
#include <stdio.h>

int main()
{
    int x;
    scanf ("%d", &x);
    return 0;
}
```

```
.LC0:
        .string "%d"
main:
        push    rbp
        mov     rbp, rsp
        sub     rsp, 16
        lea     rax, [rbp-4]
        mov     rsi, rax
        mov     edi, OFFSET FLAT:.LC0
        mov     eax, 0
        call    __isoc99_scanf
        mov     eax, 0
        leave
        ret
```

Passed arguments

```c
#include <stdio.h>

int f(int a, int b, int c)
{
    return a*b+c;
}

int main()
{
    printf ("%d\n", f(1, 2, 3));
    return 0;
}
```

```
f:
        push    rbp
        mov     rbp, rsp
        mov     DWORD PTR [rbp-4], edi
        mov     DWORD PTR [rbp-8], esi
        mov     DWORD PTR [rbp-12], edx
        mov     eax, DWORD PTR [rbp-4]
        imul    eax, DWORD PTR [rbp-8]
        mov     edx, eax
        mov     eax, DWORD PTR [rbp-12]
        add     eax, edx
        pop     rbp
        ret
.LC0:
        .string "%d\n"
main:
        push    rbp
        mov     rbp, rsp
        mov     edx, 3
        mov     esi, 2
        mov     edi, 1
        call    f
        mov     esi, eax
        mov     edi, OFFSET FLAT:.LC0
        mov     eax, 0
        call    printf
        mov     eax, 0
        pop     rbp
        ret
```

Pointers

```c
#include <stdio.h>

int main()
{
    /* Pointer p */
    int *p;

    int var = 5;

    /* Assigning the address of var to p */
    p = &var;

    return 0;
}
```

```asm
main:
        push    rbp
        mov     rbp, rsp
        mov     DWORD PTR [rbp-12], 5
        lea     rax, [rbp-12]
        mov     QWORD PTR [rbp-8], rax
        mov     eax, 0
        pop     rbp
        ret

```

Double pointers

```c
#include <stdio.h>

int main()
{
    int var = 3;

    //pointer for var
    int *ptr2;

    //double pointer for ptr2
    int **ptr1;

    //storing address of var in ptr2
    ptr2 = &var;

    //storing address of ptr2 in ptr1
    ptr1 = &ptr2;

    return 0;
}
```

```asm
main:
        push    rbp
        mov     rbp, rsp
        mov     DWORD PTR [rbp-12], 3
        lea     rax, [rbp-12]
        mov     QWORD PTR [rbp-24], rax
        lea     rax, [rbp-24]
        mov     QWORD PTR [rbp-8], rax
        mov     eax, 0
        pop     rbp
        ret

```

Arrays

```c
#include <stdio.h>

int main()
{
    int arr[5] = {1, 2, 3, 4 ,5};
    return 0;
}
```

```asm
main:
        push    rbp
        mov     rbp, rsp
        mov     DWORD PTR [rbp-32], 1
        mov     DWORD PTR [rbp-28], 2
        mov     DWORD PTR [rbp-24], 3
        mov     DWORD PTR [rbp-20], 4
        mov     DWORD PTR [rbp-16], 5
        mov     eax, 0
        pop     rbp
        ret
```

For loop

```c
#include <stdio.h>

int main()
{
    int i;

    for (i=1; i<=4; i++)
    {
        printf("%d\n", i);
    }

    return 0;
}
```

```
.LC0:
        .string "%d\n"
main:
        push    rbp
        mov     rbp, rsp
        sub     rsp, 16
        mov     DWORD PTR [rbp-4], 1
        jmp     .L2
.L3:
        mov     eax, DWORD PTR [rbp-4]
        mov     esi, eax
        mov     edi, OFFSET FLAT:.LC0
        mov     eax, 0
        call    printf
        add     DWORD PTR [rbp-4], 1
.L2:
        cmp     DWORD PTR [rbp-4], 4
        jle     .L3
        mov     eax, 0
        leave
        ret
```

If...else

```c
#include <stdio.h>

int main()
{
   int salary;

   salary = 17;

   if(salary >=15)
   {
       printf("Above min wage");
   }
   else
   {
            printf("Below min wage");
   }

   return 0;
}
```

```
.LC0:
        .string "Above min wage"
.LC1:
        .string "Below min wage"
main:
        push    rbp
        mov     rbp, rsp
        sub     rsp, 16
        mov     DWORD PTR [rbp-4], 17
        cmp     DWORD PTR [rbp-4], 14
        jle     .L2
        mov     edi, OFFSET FLAT:.LC0
        mov     eax, 0
        call    printf
        jmp     .L3
.L2:
        mov     edi, OFFSET FLAT:.LC1
        mov     eax, 0
        call    printf
.L3:
        mov     eax, 0
        leave
        ret
```

While loop

```c
#include <stdio.h>

int main()
{
   int count=1;

   while (count <= 3)
   {
        printf("%d ", count);
        count++;
   }

   return 0;
}
```

```
.LC0:
        .string "%d "
main:
        push    rbp
        mov     rbp, rsp
        sub     rsp, 16
        mov     DWORD PTR [rbp-4], 1
        jmp     .L2
.L3:
        mov     eax, DWORD PTR [rbp-4]
        mov     esi, eax
        mov     edi, OFFSET FLAT:.LC0
        mov     eax, 0
        call    printf
        add     DWORD PTR [rbp-4], 1
.L2:
        cmp     DWORD PTR [rbp-4], 3
        jle     .L3
        mov     eax, 0
        leave
        ret
```

Structures

```c
#include <stdio.h>

struct Food{
    char *name;
    int cost;
    int foodId;
};

int main()
{
   struct Food orange;

   orange.name = "Blood Orange";
   orange.cost = 3;
   orange.foodId = 1;

   printf("Food name is: %s \n", orange.name);
   printf("Food cost is: %d \n", orange.cost);
   printf("Food id is: %d \n", orange.foodId);

   return 0;
}
```

```
.LC0:
        .string "Blood Orange"
.LC1:
        .string "Food name is: %s \n"
.LC2:
        .string "Food cost is: %d \n"
.LC3:
        .string "Food id is: %d \n"
main:
        push    rbp
        mov     rbp, rsp
        sub     rsp, 16
        mov     QWORD PTR [rbp-16], OFFSET FLAT:.LC0
        mov     DWORD PTR [rbp-8], 3
        mov     DWORD PTR [rbp-4], 1
        mov     rax, QWORD PTR [rbp-16]
        mov     rsi, rax
        mov     edi, OFFSET FLAT:.LC1
        mov     eax, 0
        call    printf
        mov     eax, DWORD PTR [rbp-8]
        mov     esi, eax
        mov     edi, OFFSET FLAT:.LC2
        mov     eax, 0
        call    printf
        mov     eax, DWORD PTR [rbp-4]
        mov     esi, eax
        mov     edi, OFFSET FLAT:.LC3
        mov     eax, 0
        call    printf
        mov     eax, 0
        leave
        ret
```