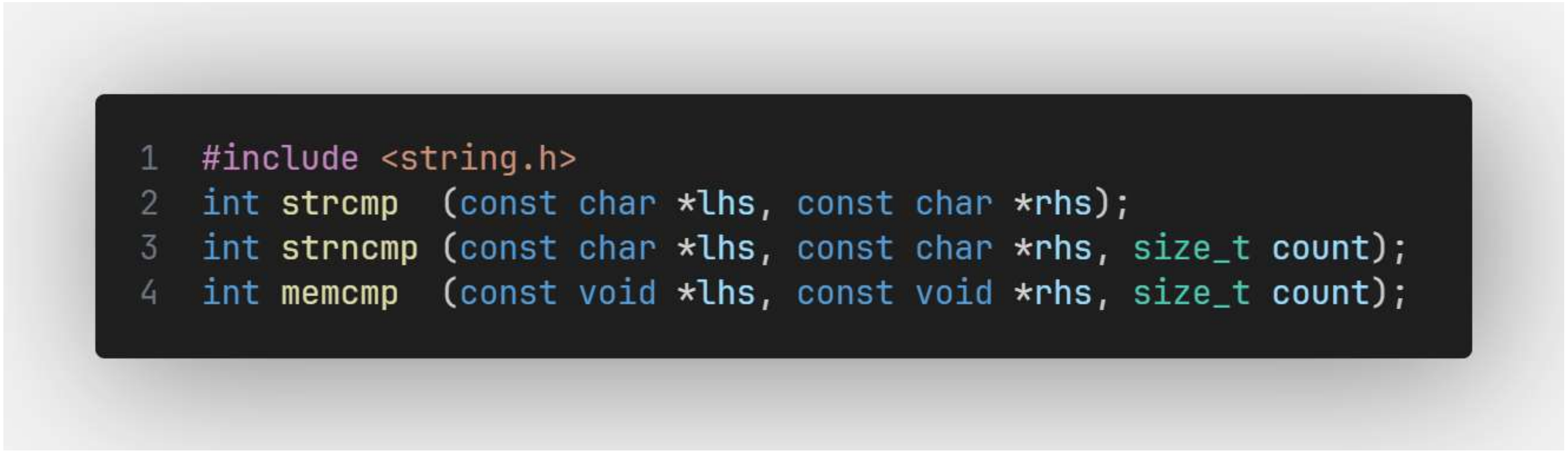
NekoBytes Week7 More about C

Part 1. string.h

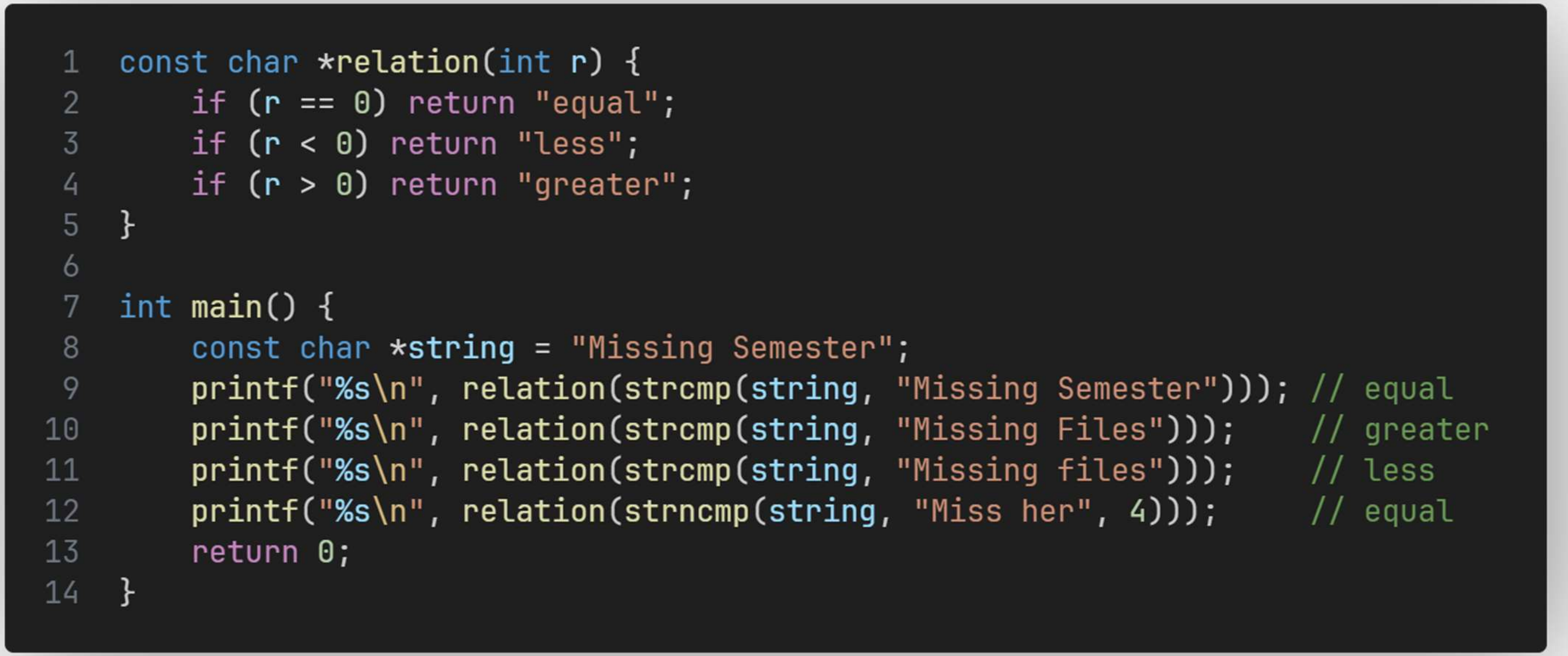
Compare



```c
#include <string.h>
int strcmp  (const char *lhs, const char *rhs);
int strncmp (const char *lhs, const char *rhs, size_t count);
int memcmp  (const void *lhs, const void *rhs, size_t count);
```

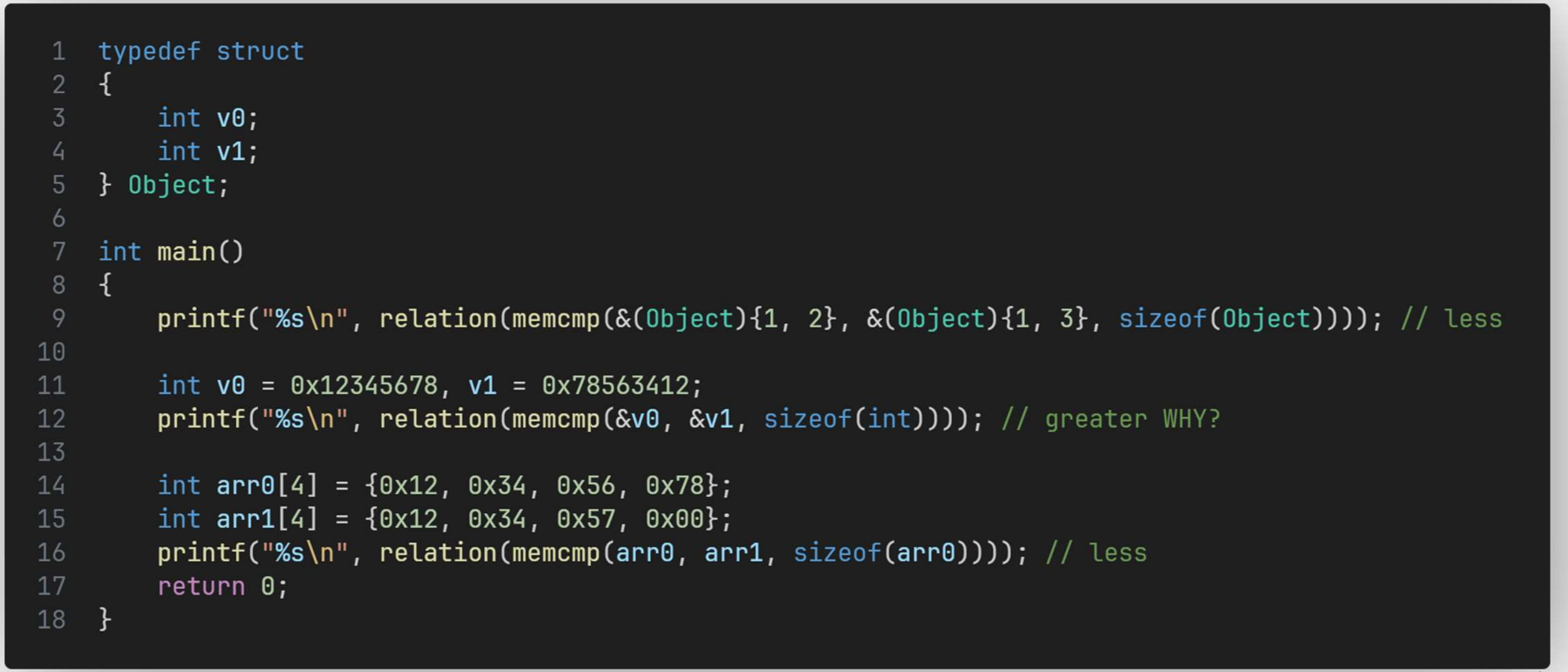
$$\text{Return value}\begin{cases}\texttt{0 lhs = rhs}\\ \texttt{+ lhs > rhs}\\ \texttt{- lhs > rhs}\end{cases}$$

Compare



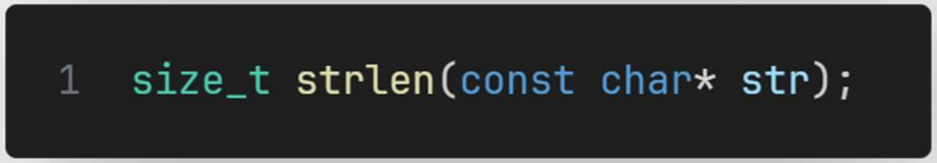
```c
const char *relation(int r) {
    if (r == 0) return "equal";
    if (r < 0) return "less";
    if (r > 0) return "greater";
}

int main() {
    const char *string = "Missing Semester";
    printf("%s\n", relation(strcmp(string, "Missing Semester"))); // equal
    printf("%s\n", relation(strcmp(string, "Missing Files")));    // greater
    printf("%s\n", relation(strcmp(string, "Missing files")));    // less
    printf("%s\n", relation(strncmp(string, "Miss her", 4)));     // equal
    return 0;
}
```

Compare

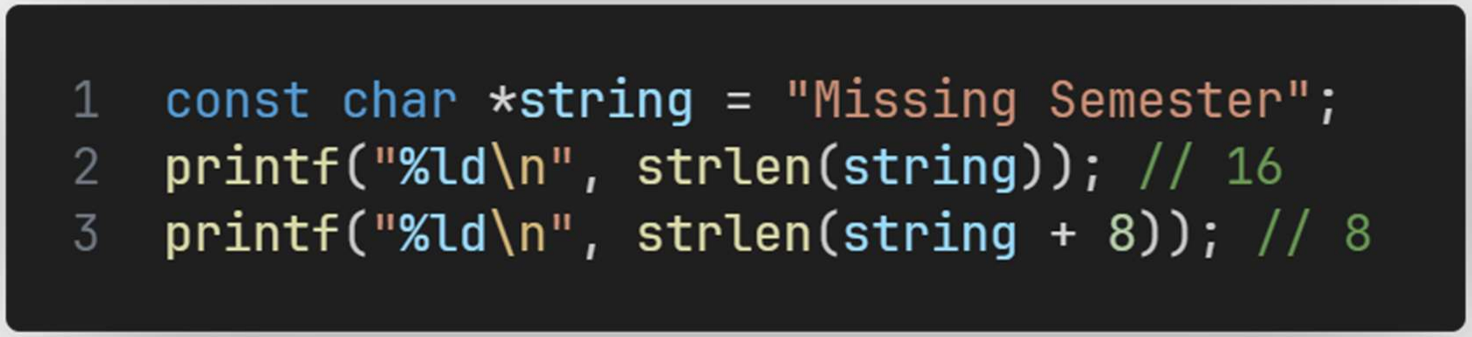


```c
typedef struct
{
    int v0;
    int v1;
} Object;

int main()
{
    printf("%s\n", relation(memcmp(&(Object){1, 2}, &(Object){1, 3}, sizeof(Object)))); // less

    int v0 = 0x12345678, v1 = 0x78563412;
    printf("%s\n", relation(memcmp(&v0, &v1, sizeof(int)))); // greater WHY?

    int arr0[4] = {0x12, 0x34, 0x56, 0x78};
    int arr1[4] = {0x12, 0x34, 0x57, 0x00};
    printf("%s\n", relation(memcmp(arr0, arr1, sizeof(arr0)))); // less
    return 0;
}
```

Length

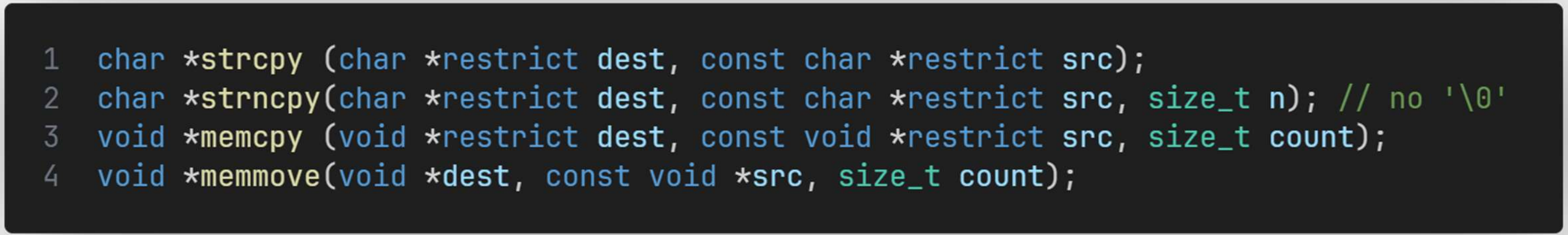


```c
size_t strlen(const char* str);
```



```c
const char *string = "Missing Semester";
printf("%ld\n", strlen(string)); // 16
printf("%ld\n", strlen(string + 8)); // 8
```

Copy



```c
char *strcpy (char *restrict dest, const char *restrict src);
char *strncpy(char *restrict dest, const char *restrict src, size_t n); // no '\0'
void *memcpy (void *restrict dest, const void *restrict src, size_t count);
void *memmove(void *dest, const void *src, size_t count);
```

WARNING:

OVERFLOW and OVERLAP

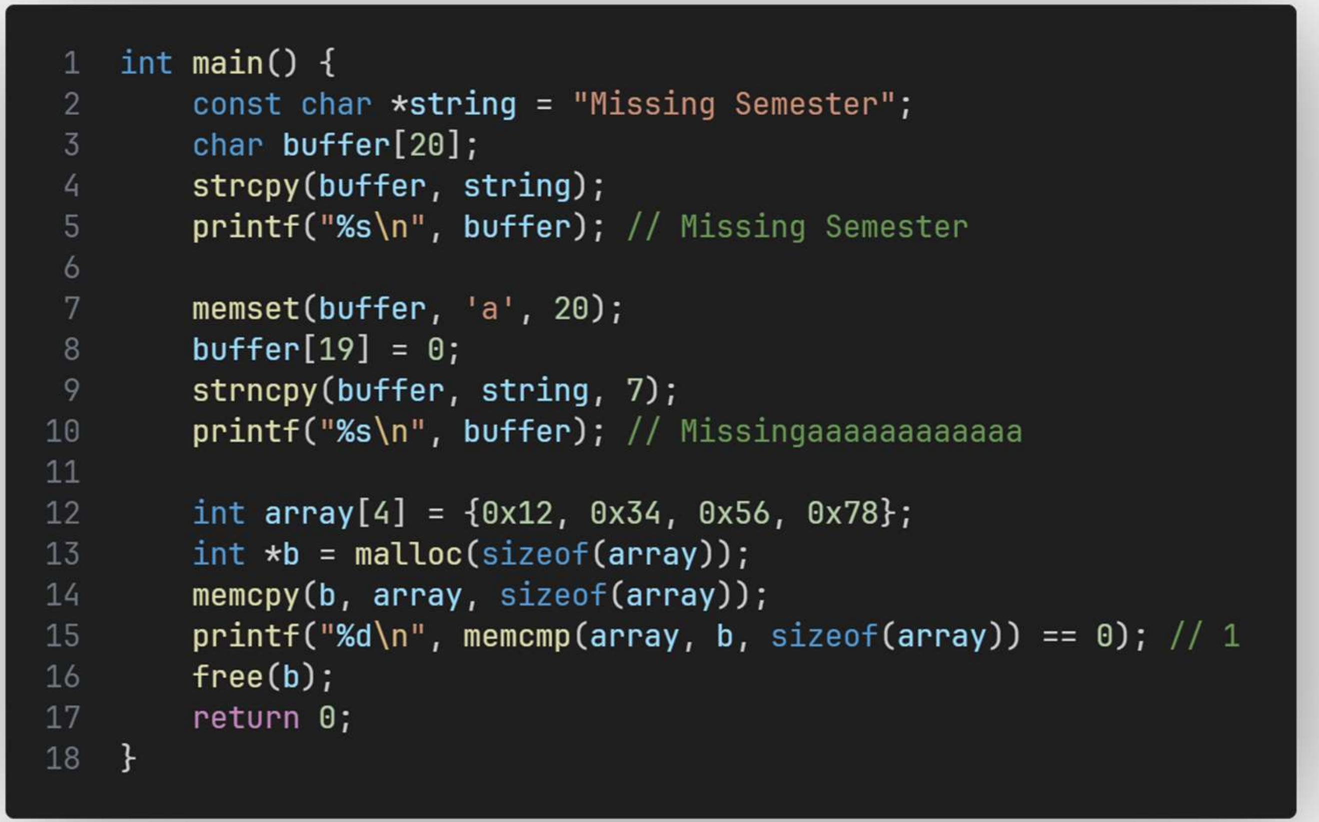
Fill

```c
void *memset(void *dest, int ch, size_t count);
```

Fill with `(unsinged char)ch`

What is the difference between `(unsinged char)ch` and `(signed char)ch`?

Copy and Fill Example



```c
int main() {
    const char *string = "Missing Semester";
    char buffer[20];
    strcpy(buffer, string);
    printf("%s\n", buffer); // Missing Semester

    memset(buffer, 'a', 20);
    buffer[19] = 0;
    strncpy(buffer, string, 7);
    printf("%s\n", buffer); // Missingaaaaaaaaaaaa

    int array[4] = {0x12, 0x34, 0x56, 0x78};
    int *b = malloc(sizeof(array));
    memcpy(b, array, sizeof(array));
    printf("%d\n", memcmp(array, b, sizeof(array)) == 0); // 1
    free(b);
    return 0;
}
```

Search and Match

First Occurrence

```c
void *memchr (const void* ptr, int ch, size_t count);
char *strchr (const char* str, int ch);
char *strstr(const char *str, const char *substr);
size_t strcspn(const char *dest, const char *src);
```

Last Occurrence

```c
char* strrchr(const char* str, int ch);
```

First Not In

```c
size_t strspn(const char* dest, const char* src);
```

Search and Match

Split

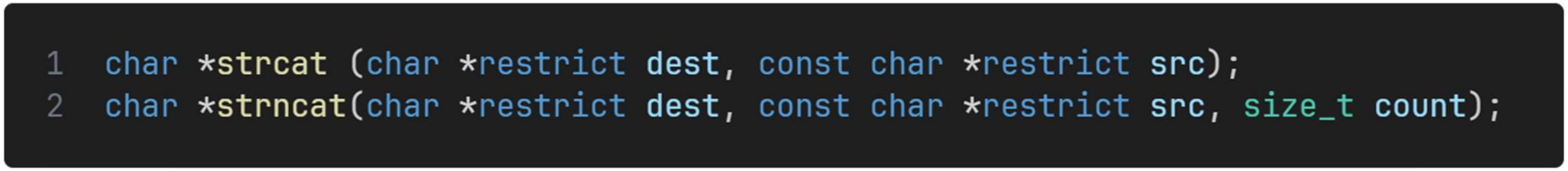
```c
char *strtok(char *str, const char *delim);
```

WARNING:

NOT CONST

```c
int main() {
    char string[48] = "I am studying the Missing Semester";
    char *tok = strtok(string, " ");
    while (tok != NULL) {
        printf("%s\n", tok);
        tok = strtok(NULL, " ");
    }
    /*
    I
    am
    studying
    the
    Missing
    Semester
    */
    return 0;
}
```

Catch

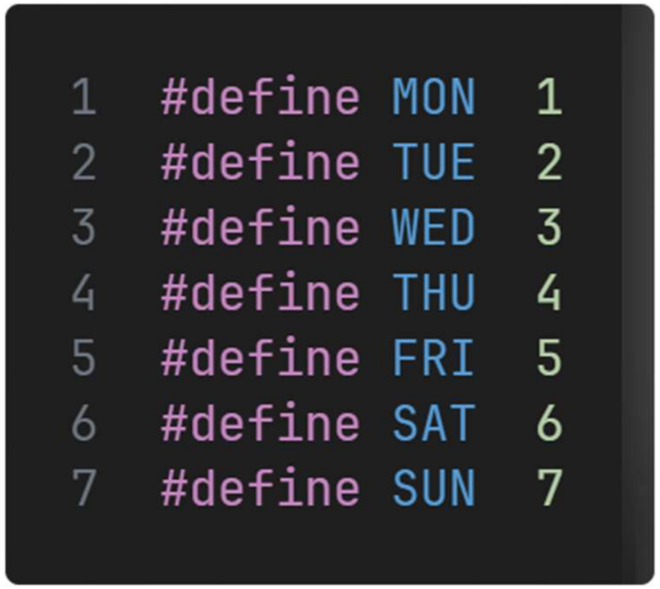


```c
char *strcat (char *restrict dest, const char *restrict src);
char *strncat(char *restrict dest, const char *restrict src, size_t count);
```

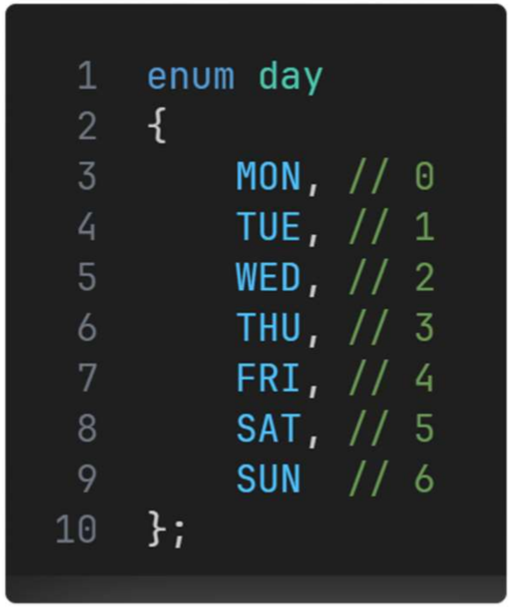
NekoBytes Week7 More about C

Part 2. Enumeration and Union

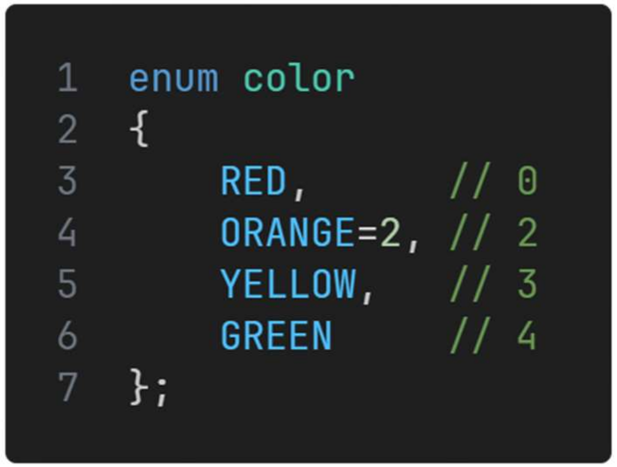
Way to Tag



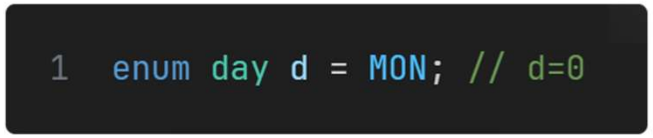
```c
#define MON   1
#define TUE   2
#define WED   3
#define THU   4
#define FRI   5
#define SAT   6
#define SUN   7
```



```c
enum day
{
    MON, // 0
    TUE, // 1
    WED, // 2
    THU, // 3
    FRI, // 4
    SAT, // 5
    SUN  // 6
};
```

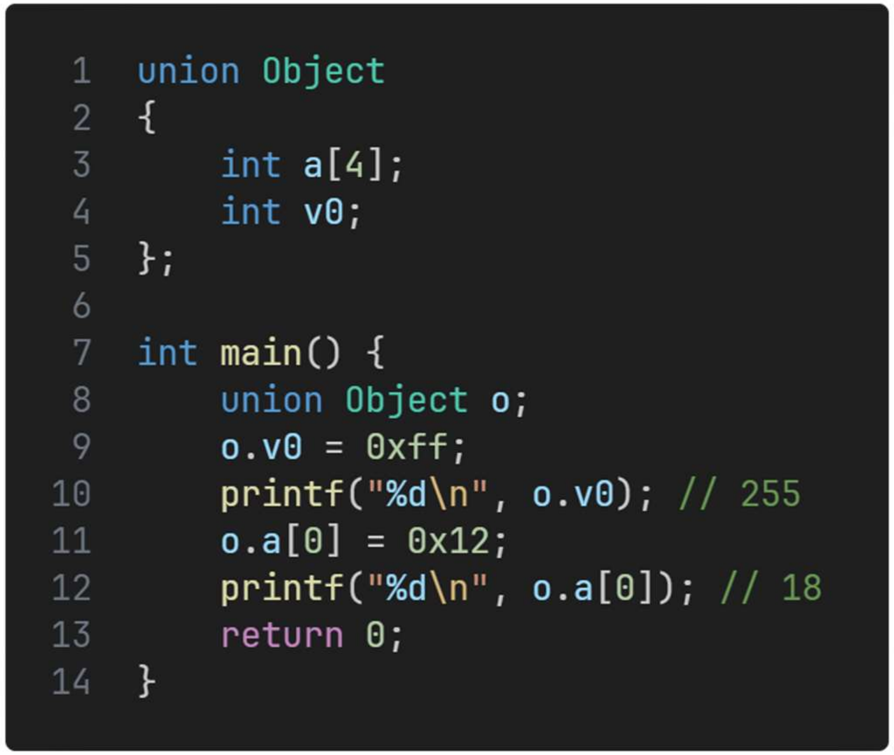


```c
enum color
{
    RED,       // 0
    ORANGE=2, // 2
    YELLOW,    // 3
    GREEN      // 4
};
```



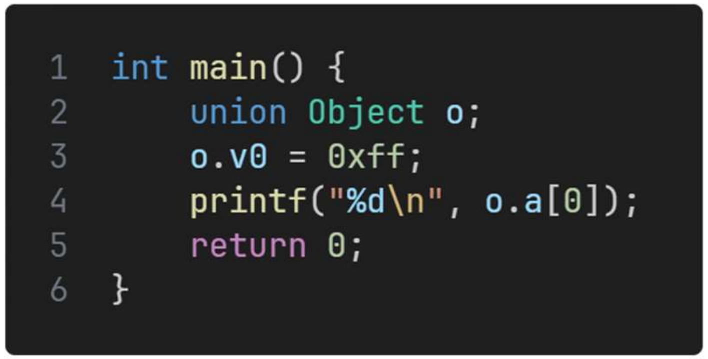
```c
enum day d = MON; // d=0
```

Union



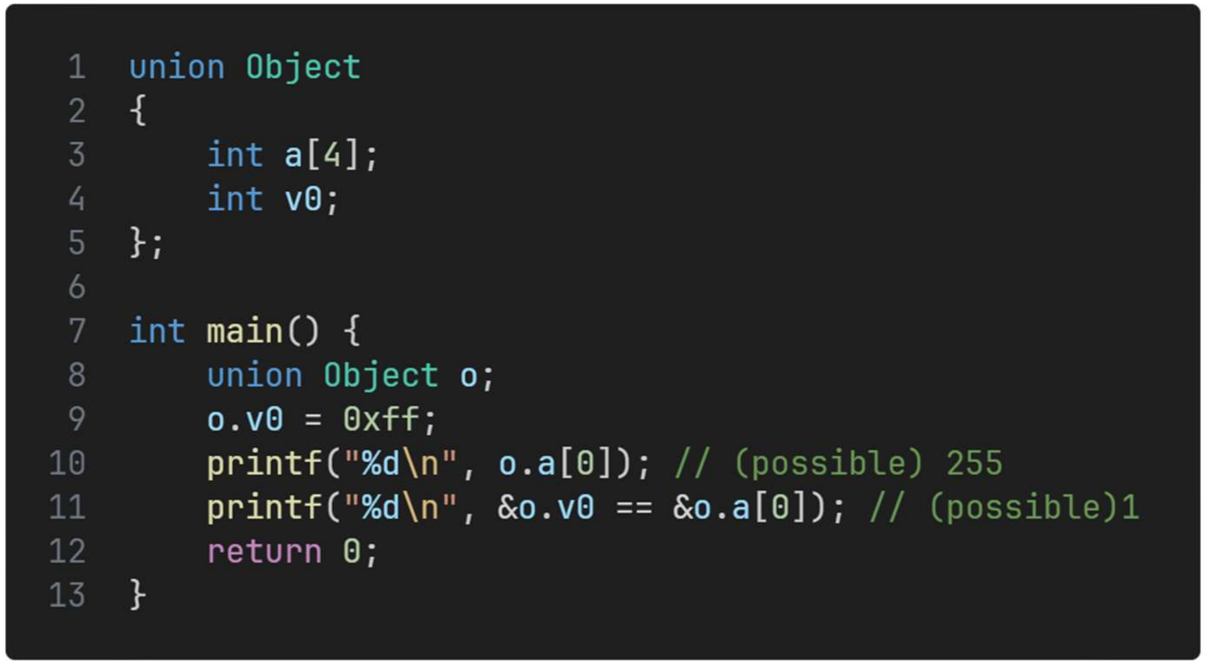
```c
union Object
{
    int a[4];
    int v0;
};

int main() {
    union Object o;
    o.v0 = 0xff;
    printf("%d\n", o.v0); // 255
    o.a[0] = 0x12;
    printf("%d\n", o.a[0]); // 18
    return 0;
}
```

WARNING:
UNDEFINED BEHAVIOR

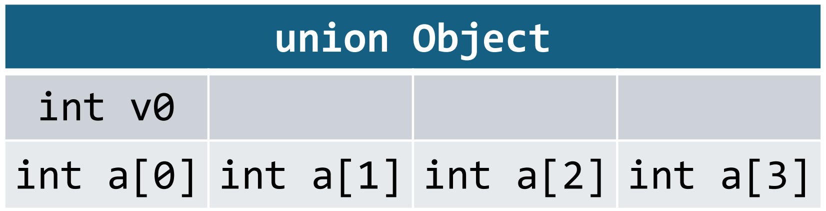


```c
int main() {
    union Object o;
    o.v0 = 0xff;
    printf("%d\n", o.a[0]);
    return 0;
}
```

One Possible Memory Use

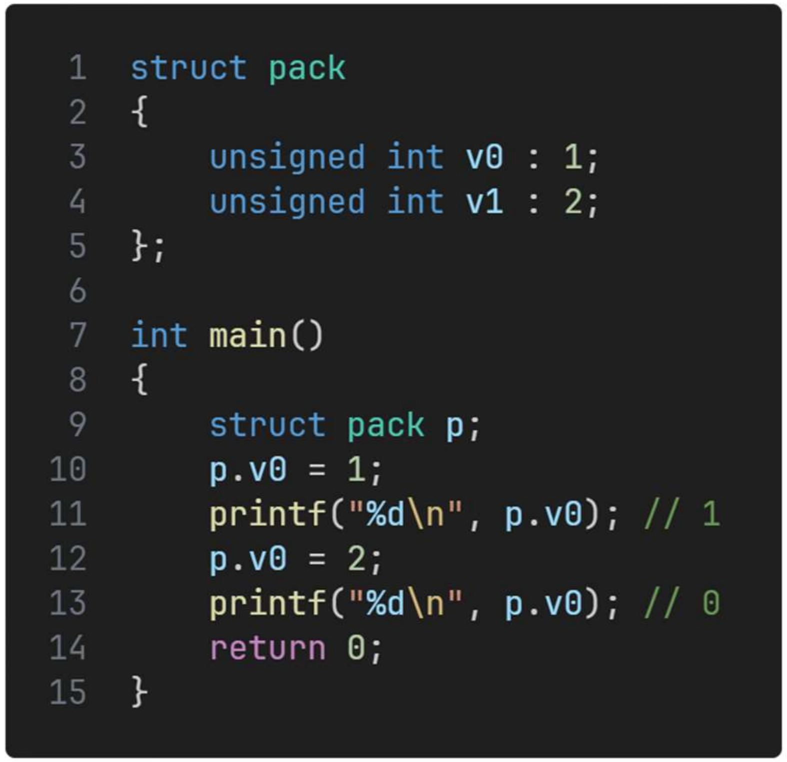


```c
union Object
{
    int a[4];
    int v0;
};

int main() {
    union Object o;
    o.v0 = 0xff;
    printf("%d\n", o.a[0]); // (possible) 255
    printf("%d\n", &o.v0 == &o.a[0]); // (possible)1
    return 0;
}
```



sizeof(union Object) = 16

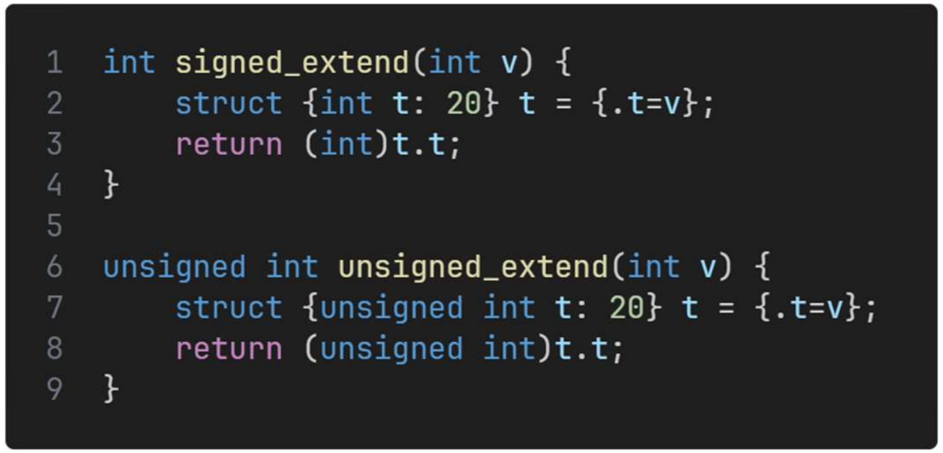
Bit-field – use less or more bytes



```c
struct pack
{
    unsigned int v0 : 1;
    unsigned int v1 : 2;
};

int main()
{
    struct pack p;
    p.v0 = 1;
    printf("%d\n", p.v0); // 1
    p.v0 = 2;
    printf("%d\n", p.v0); // 0
    return 0;
}
```

What is the value of `sizeof(p)`?

Bit Extender by Bit-field



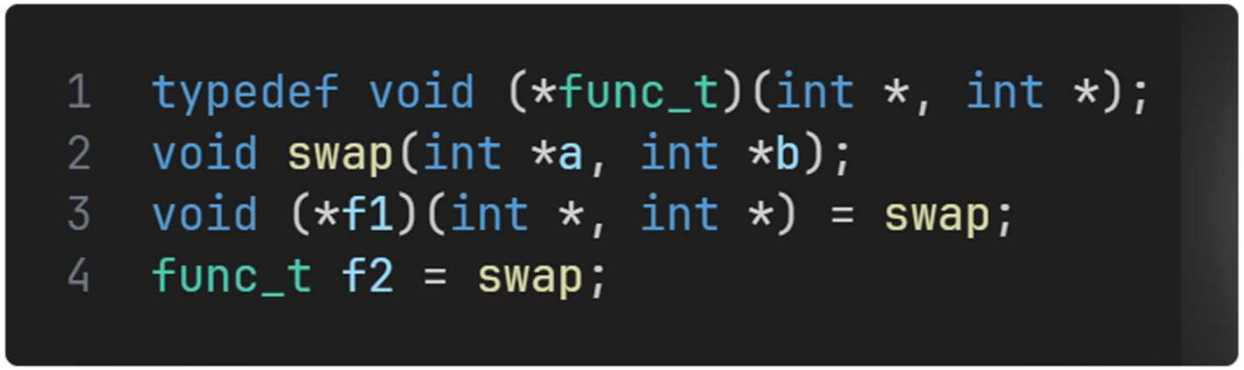
```c
int signed_extend(int v) {
    struct {int t: 20} t = {.t=v};
    return (int)t.t;
}

unsigned int unsigned_extend(int v) {
    struct {unsigned int t: 20} t = {.t=v};
    return (unsigned int)t.t;
}
```

NekoBytes Week7 More about C

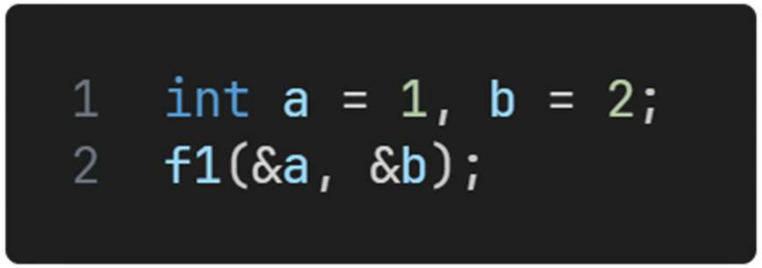
Part 3. The Pointer of the Function

Function Pointer

```
return_type (*pointer_of_function)(arguments_type_list)
typedef return_type (*function_t)(arguments_type_list)
```



```c
typedef void (*func_t)(int *, int *);
void swap(int *a, int *b);
void (*f1)(int *, int *) = swap;
func_t f2 = swap;
```



```c
int a = 1, b = 2;
f1(&a, &b);
```

Override in Struct

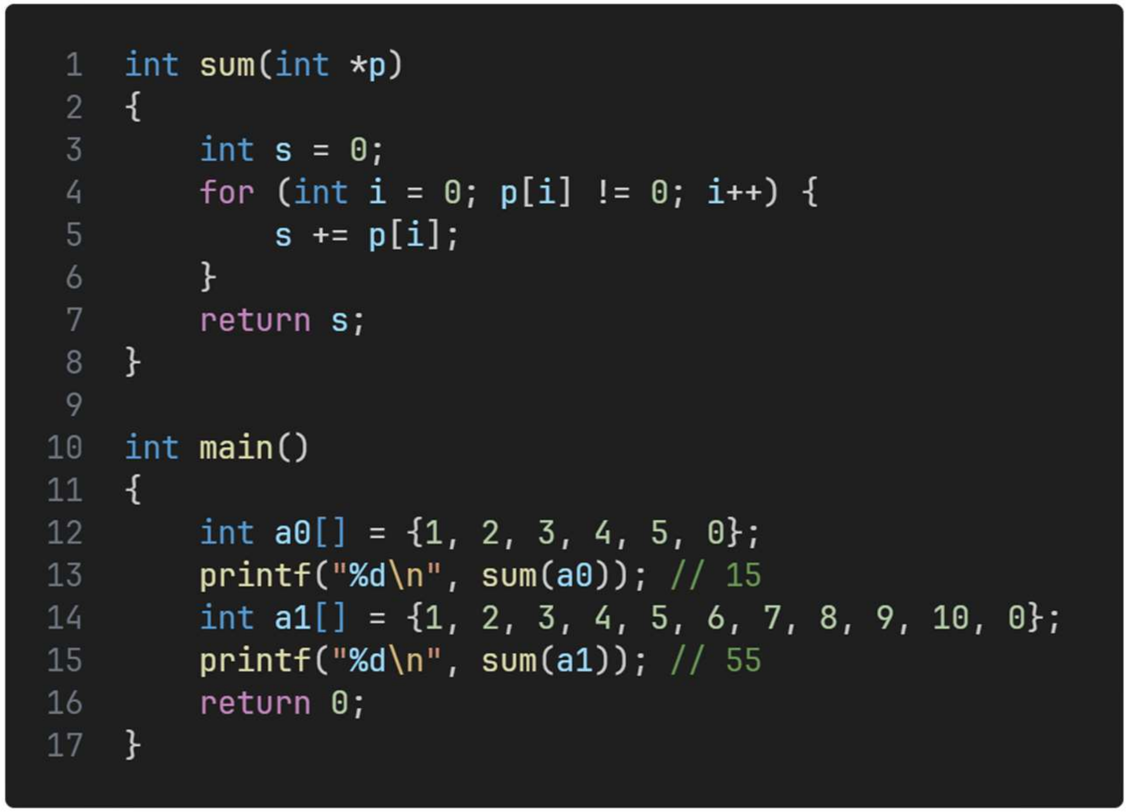
```c
#define ANIMAL_HEADER char name[20]; \
        void (*make_sound)(struct animal *this);

typedef struct animal
{
    ANIMAL_HEADER;
} Animal;

typedef struct dog
{
    ANIMAL_HEADER;
    char favorite[20];
} Dog;

typedef struct cat
{
    ANIMAL_HEADER;
    char goodAt[20];
} Cat;
```

```c
static void dog_make_sound(Animal *this_) {
    Dog *this = (Dog *)this_;
    printf("I am dog %s, I like %s best.\n", this->name, this->favorite);
}

void dog_init(Dog *dog, char *name, char *favorite) {
    dog->make_sound = dog_make_sound;
    strncpy(dog->name, name, 19);
    strncpy(dog->favorite, favorite, 19);
}

static void cat_make_sound(Animal *this_) {
    Cat *this = (Cat *)this_;
    printf("I am cat %s, I am good at %s.\n", this->name, this->goodAt);
}

void cat_init(Cat *cat, char *name, char *goodAt) {
    cat->make_sound = cat_make_sound;
    strncpy(cat->name, name, 19);
    strncpy(cat->goodAt, goodAt, 19);
}
```

NekoBytes Week7 More about C

Part 3.
Variable Parameter Function

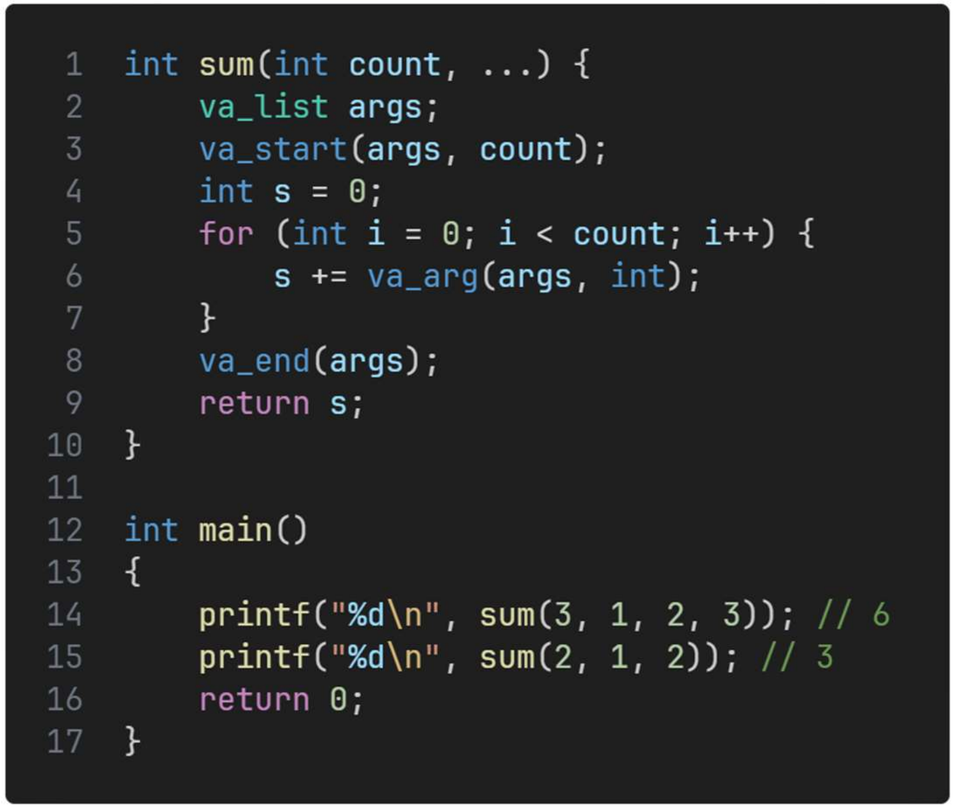
Use pointer



```c
int sum(int *p)
{
    int s = 0;
    for (int i = 0; p[i] != 0; i++) {
        s += p[i];
    }
    return s;
}

int main()
{
    int a0[] = {1, 2, 3, 4, 5, 0};
    printf("%d\n", sum(a0)); // 15
    int a1[] = {1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 0};
    printf("%d\n", sum(a1)); // 55
    return 0;
}
```

Use your own rule to phrase memory

stdarg.h



```c
int sum(int count, ...) {
    va_list args;
    va_start(args, count);
    int s = 0;
    for (int i = 0; i < count; i++) {
        s += va_arg(args, int);
    }
    va_end(args);
    return s;
}

int main()
{
    printf("%d\n", sum(3, 1, 2, 3)); // 6
    printf("%d\n", sum(2, 1, 2)); // 3
    return 0;
}
```

1. Init

```
va_list args;
va_start(args, last_parameter)
```

2. Use

```
va_arg(args, type_of_parameter)
```

3. Deinit

```
va_end(args)
```