What have we learned?

Branch / conditional statements

May be a combination of conditions through Boolean operation

```
if (condition)
{   ⇦  Use { } to include more than one statements;
    instructions;
}
else
{
    alternative_instructions;
}
```

Optional

Expression, instead of a complete condition!

```
switch (expression)
{
case value1:
    instructions_1;
    break;
case value2:
    instructions_2;
    break;
  ⋮
default:
    default_instructions;
}
```

int or char only!

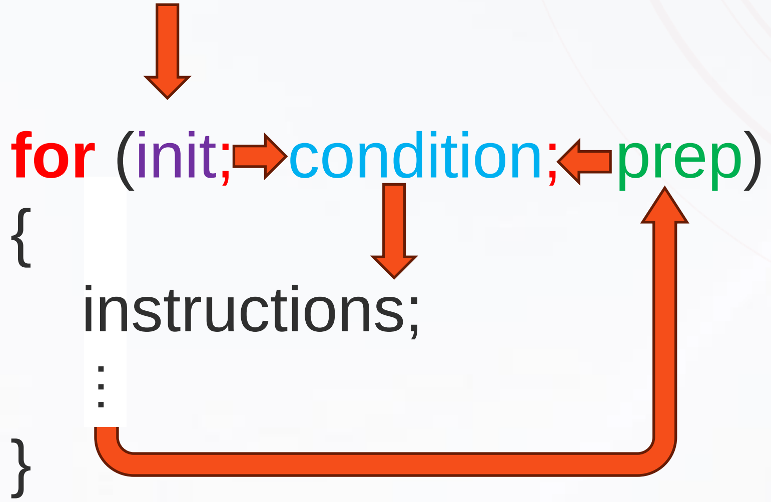
break not mandatory, but highly recommended!

Optional

What have we learned

Loop statements

```
while (condition)
{
    instructions;
    ⋮
}
```

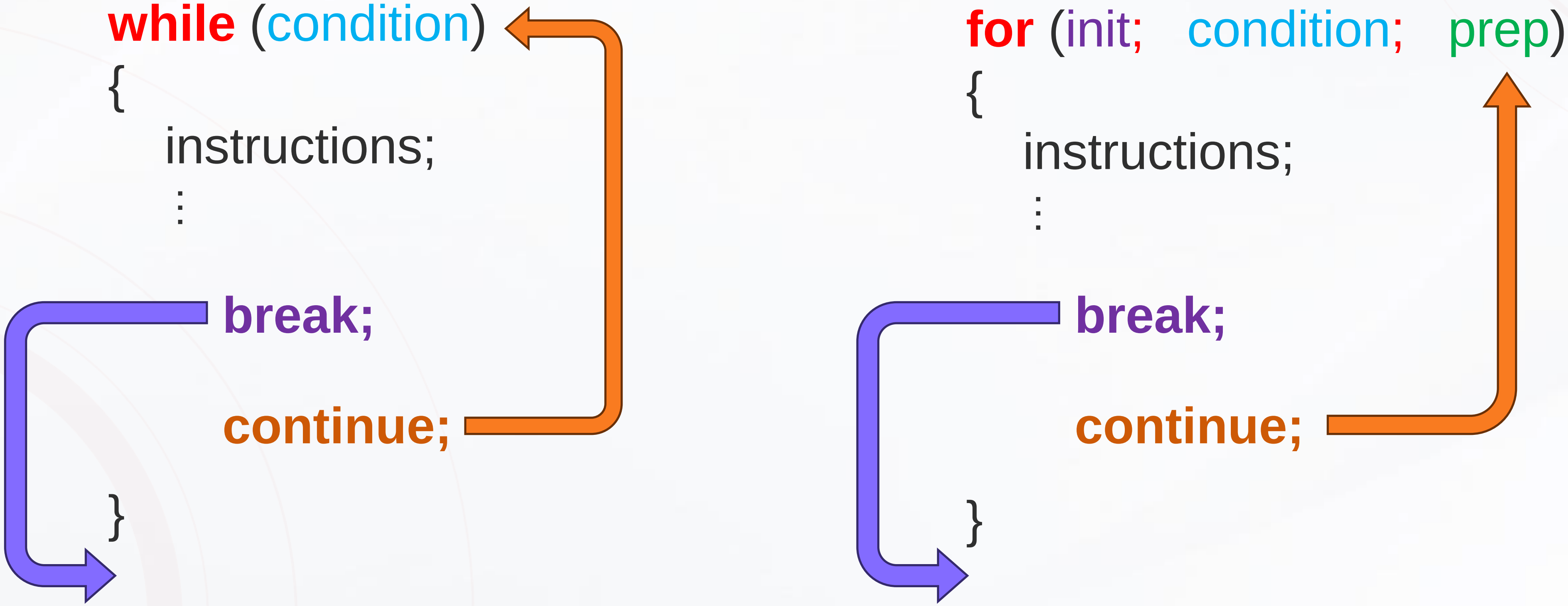


```
for (init; condition; prep)
{
    instructions;
    ⋮
}
```

Usually
for (i = 0; i < max; i++)

What have we learned

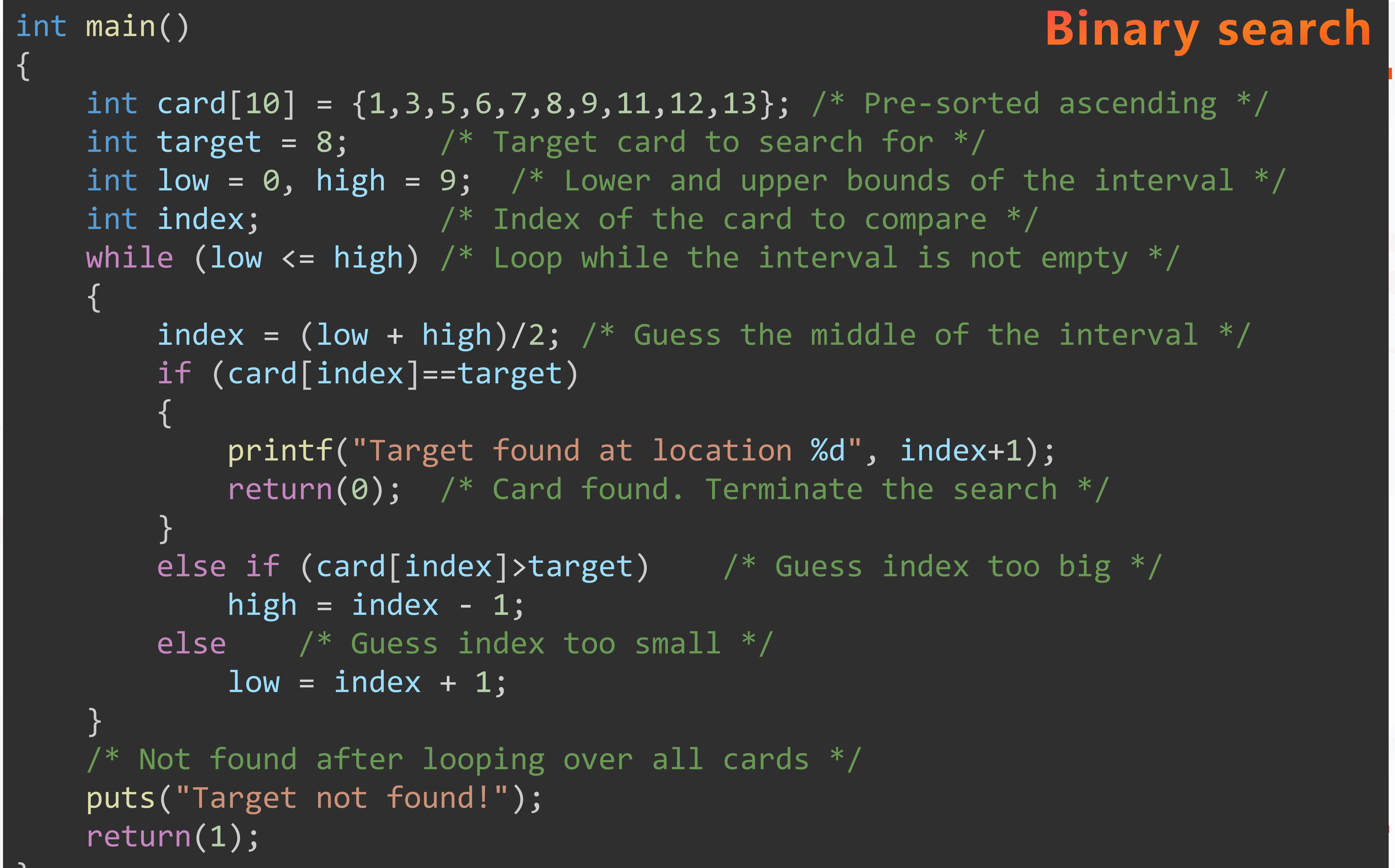
break and continue



```
while (condition)
{
    instructions;
    ⋮

      break;

      continue;

}
```

```
for (init;   condition;   prep)
{
    instructions;
    ⋮

      break;

      continue;

}
```

Binary search



```c
int main()
{
    int card[10] = {1,3,5,6,7,8,9,11,12,13}; /* Pre-sorted ascending */
    int target = 8;     /* Target card to search for */
    int low = 0, high = 9;  /* Lower and upper bounds of the interval */
    int index;          /* Index of the card to compare */
    while (low <= high) /* Loop while the interval is not empty */
    {
        index = (low + high)/2; /* Guess the middle of the interval */
        if (card[index]==target)
        {
            printf("Target found at location %d", index+1);
            return(0);  /* Card found. Terminate the search */
        }
        else if (card[index]>target)    /* Guess index too big */
            high = index - 1;
        else    /* Guess index too small */
            low = index + 1;
    }
    /* Not found after looping over all cards */
    puts("Target not found!");
    return(1);
}
```

Equivalence between while and for loops

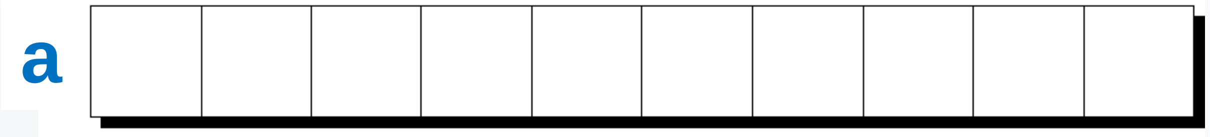
```
int main()
{
    int card[10] = {1,3,5,6,7,8,9,11,12,13}; /* Pre-sorted ascending */
    int target = 8;     /* Target card to search for */
    int low = 0, high = 9;  /* Lower and upper bounds of the interval */
    for (int index = 4; low <= high; index = (low + high)/2)
    {
        if (card[index]==target)
        {
            printf("Target found at location %d", index+1);
            return(0);  /* Card found. Terminate the search */
        }
        else if (card[index]>target)    /* Guess index too big */
            high = index - 1;
        else    /* Guess index too big */
            low = index + 1;
    }
    /* Not found after looping over all cards */
    puts("Target not found!");
    return(1);
}
```

Choose by convenience & readability

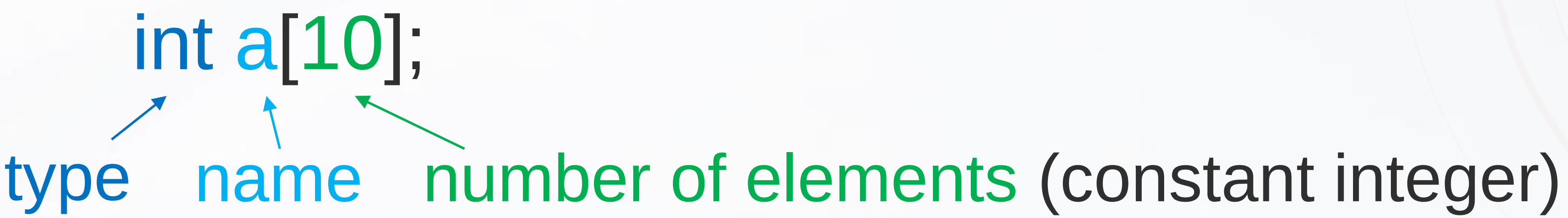
Lecture 4 Arrays

Array

- ***array*** - a data structure containing a number of data items, all of which have the same type.
- These items, known as ***elements,*** can be individually selected by their position within the array.
- The simplest kind of array has just one dimension.
- The elements of a one-dimensional array **a** are conceptually arranged one after another in a single row (or column):



Declaration of an array



```
int a[10];
```

type name number of elements (constant integer)

- Use a pre-defined *macro* to define the length

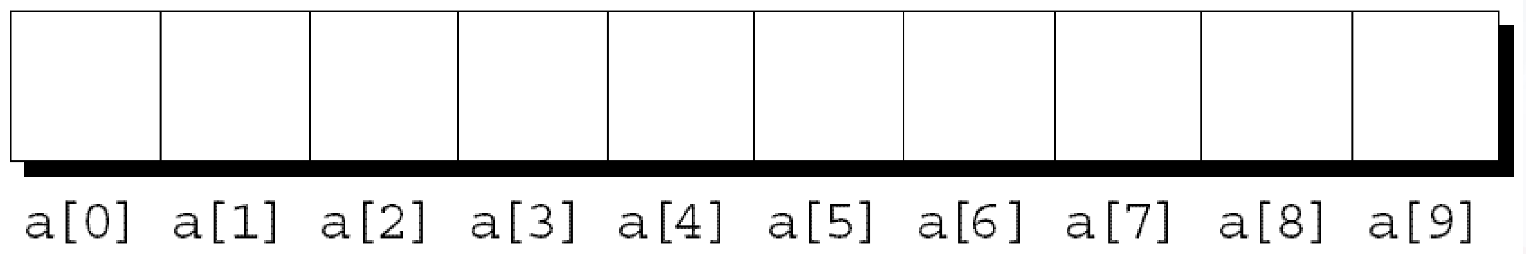
```
#define NumElem 10

 …

int a[NumElem];
```

- Initialization (at the time of declaration)

```
int a[10] = {1, 2, 3, 4, 5, 6, 7, 8, 9, 10};
int a[ ] = {1, 2, 3, 4, 5, 6, 7, 8, 9, 10};
```

/ The default length of an array is that of the initializer */*

Array indexing



a[i]

- Accessing an element of an array, just like an ordinary variable of the same type.

```
a[0] = 1;
printf("%d\n", a[5]);
++a[i];
```

- Array index starts from 0 in C!
- Array of *n* elements are indexed from 0 to *n* – 1.

Array indexing

- Use a for loop to (*naturally*) access the elements of an array sequentially

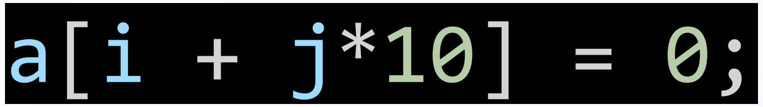
```c
for (i = 0; i < N; i++)
    scanf("%d", &a[i]);         /* reads data into a */
sum = 0;
for (i = 0; i < N; i++)
    sum += a[i];                /* sums all elements */
```

- C compiler does *not* check the **bounds** for you!

```c
int a[10], i;
for (i = 1; i <= 10; i++)
    a[i] = 0;
```

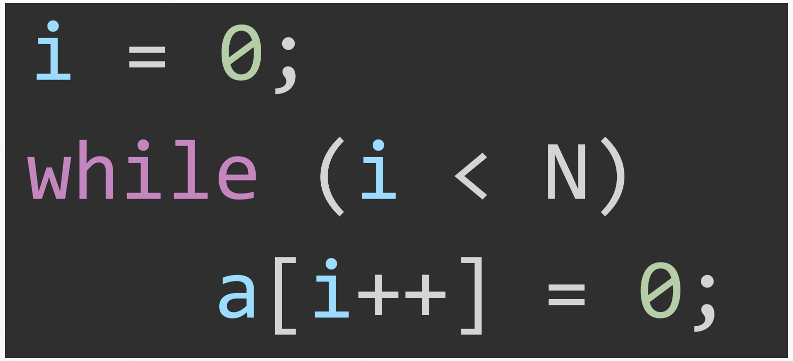
Side effects of indices

- An array subscript may be any integer expression:



```
a[i + j*10] = 0;
```

- Warning: the expression inside the bracket may have side effects:



```
i = 0;
while (i < N)
    a[i++] = 0;
```

Do we need the parentheses/brackets?

C Operator Precedence (incomplete)

Precedence	Operator	Description
1	++ -- () []	Suffix/postfix increment and decrement Function call Array subscripting
2	!	Logical NOT
3	* / %	Multiplication, division, and remainder
4	+ -	Addition and subtraction
5	< <= > >=	Relational operators < and ≤ respectively Relational operators > and ≥ respectively
6	== !=	Relational = and ≠ respectively
7	&&	Logical AND
8	||	Logical OR
9	= += -=	Simple assignment Assignment by sum and difference

If you are not 100% sure what would happen, then avoid using it!

```
i = 0;
while (i < N)
    a[i] = b[i++];
```

```
for (i = 0; i < N; i++)
    a[i] = b[i];
```

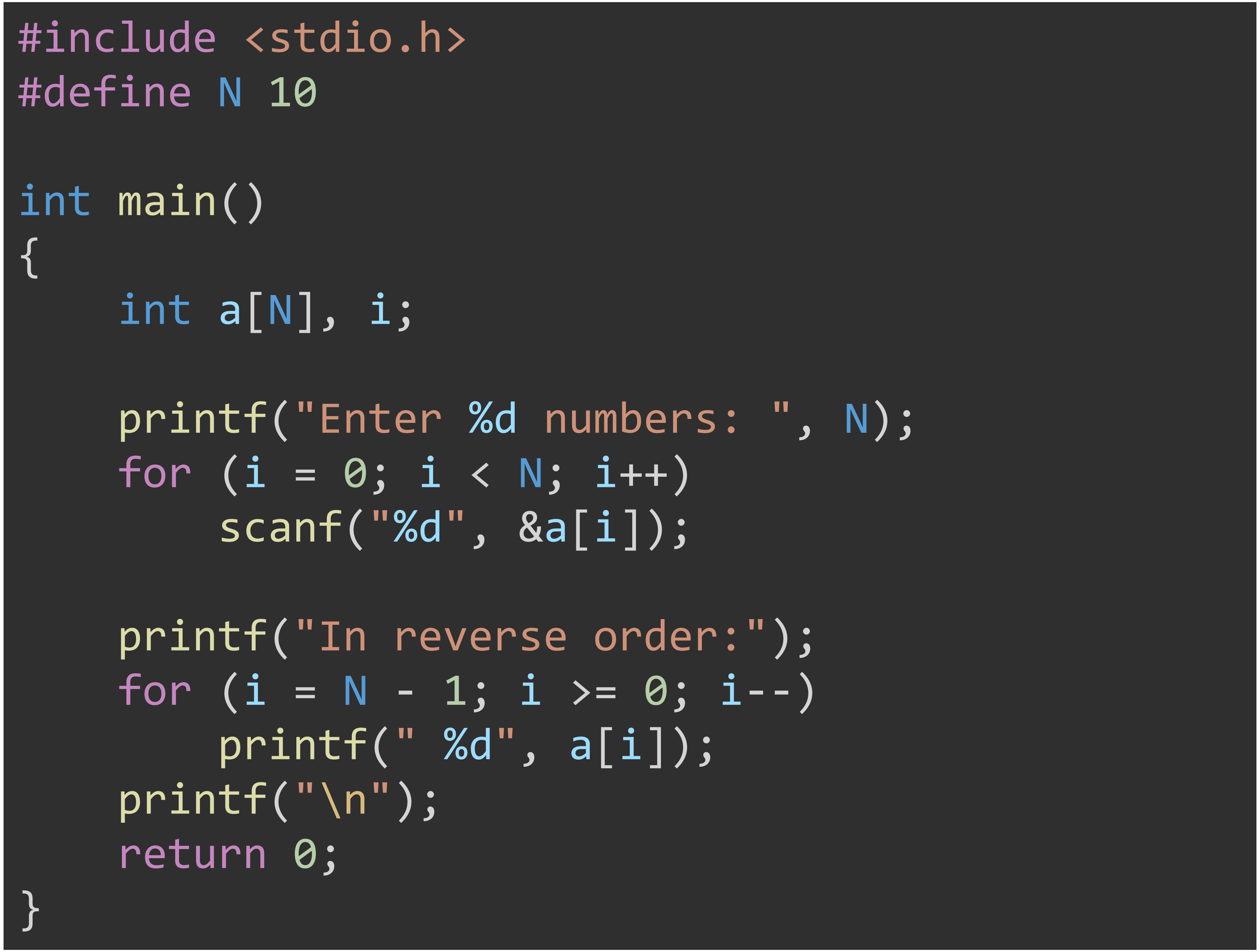
What will happen?

Avoid confusion/undefined operation by moving the increment out of the index!

Example: Reserve the order!

Write a program that prompts the user to enter a series of numbers, then writes the numbers in reverse order

```
Enter 10 numbers: 34 82 49 102 7 94 23 11 50 31
In reverse order: 31 50 11 23 94 7 102 49 82 34
```



```c
#include <stdio.h>
#define N 10

int main()
{
    int a[N], i;

    printf("Enter %d numbers: ", N);
    for (i = 0; i < N; i++)
        scanf("%d", &a[i]);

    printf("In reverse order:");
    for (i = N - 1; i >= 0; i--)
        printf(" %d", a[i]);
    printf("\n");
    return 0;
}
```

More about initialization

```
int a[10] = {1, 2, 3, 4, 5, 6};
    /* initial value of a is {1, 2, 3, 4, 5, 6, 0, 0, 0, 0} */
```

- The initializer may be shorter – the remaining ones will be filled with 0s.

```
int a[10] = {0};
/* initial value of a is {0, 0, 0, 0, 0, 0, 0, 0, 0, 0} */
```

- Can we omit the interstitial 0s?

```
int a[15] = {0, 0, 29, 0, 0, 0, 0, 0, 0, 7, 0, 0, 0, 0, 48};
```

```
int a[15] = {[2] = 29, [9] = 7, [14] = 48};
```

Designated initializer */* All others are filled with 0s */*

Designated initializer (C99 and after)

- Does not have to be in order

```
int a[15] = {[14] = 48, [9] = 7, [2] = 29};
```

- May be mixed with the conventional way (not recommended)

```
int c[10] = {5, 1, 9, [4] = 3, 7, 2, [8] = 6};
```

- When length omitted, compiler will deduce from the *largest designator*

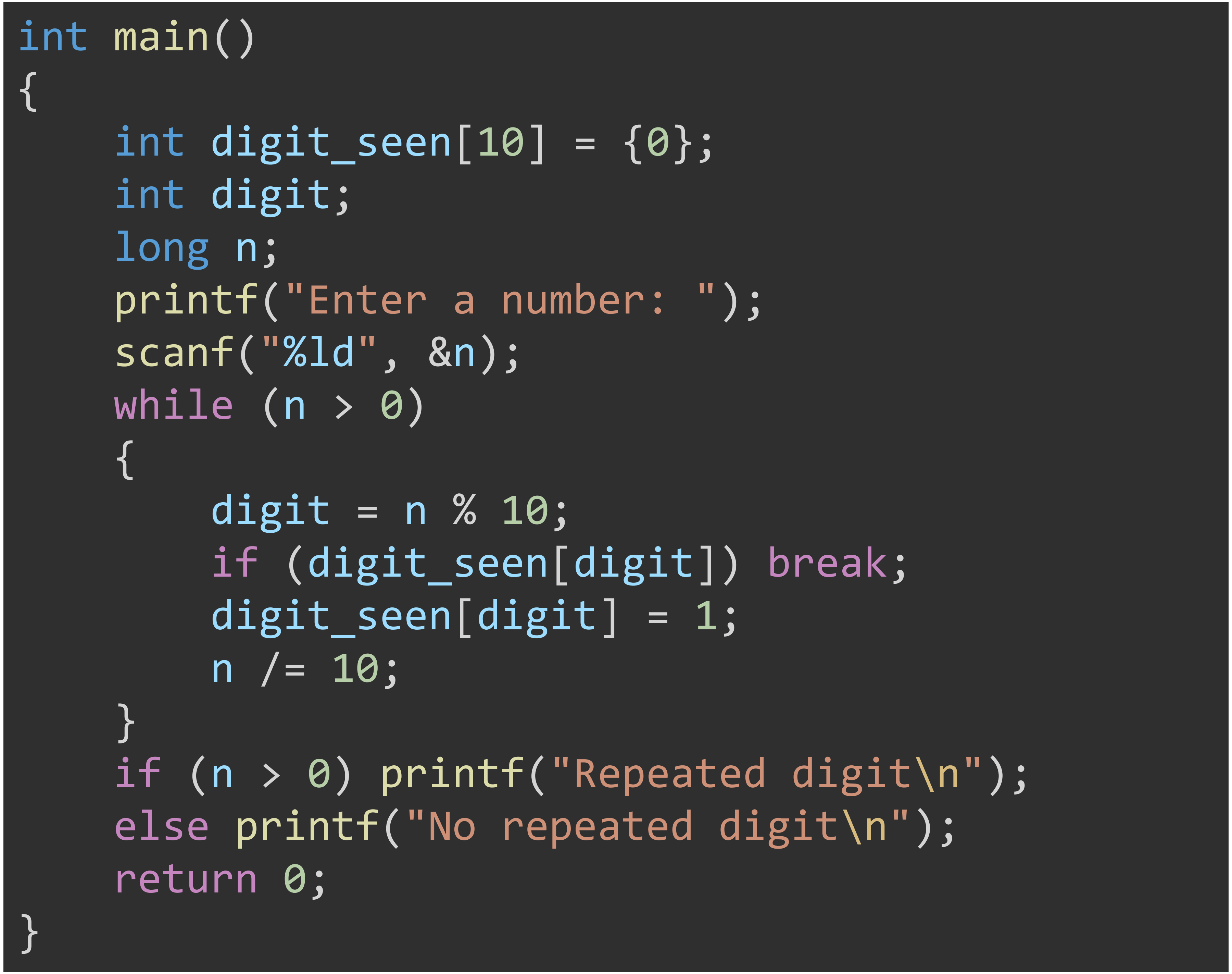
```
int b[] = {[5] = 10, [23] = 13, [11] = 36, [15] = 29};
```

/ b has 24 elements */*

Example: Checking repeated digits

- Write a program to check whether any of the digits in a number appears more than once.

```
Enter a number: 28212
    Repeated digit

Enter a number: 935742
    No repeated digit
```



```c
int main()
{
    int digit_seen[10] = {0};
    int digit;
    long n;
    printf("Enter a number: ");
    scanf("%ld", &n);
    while (n > 0)
    {
        digit = n % 10;
        if (digit_seen[digit]) break;
        digit_seen[digit] = 1;
        n /= 10;
    }
    if (n > 0) printf("Repeated digit\n");
    else printf("No repeated digit\n");
    return 0;
}
```