In The Name Of God

Principles Of Programming

(Session 4)

Chapter 2, 3

K&R

Presenter : Graders' Team – Spring 2019

Constants

In C program we can define constants in two ways:

1. Using `#define` preprocessor directive, example:

```
#define MAX 100
```

2. Using a `const` keyword, example:

```
const int MAX = 100;
```

Example 1:

```c
#include <stdio.h>

#define MAXLINE 1000


int main() {

    char line [MAXLINE+1];
    return 0;

}
```

Example 2:

```c
#include <stdio.h>

#define NEWLINE printf("\n");


int main() {

    printf("Hello");
    NEWLINE;
    printf("World");

    return 0;
}

// output: Hello\nWorld
```

Example 3:

```c
#include <stdio.h>

#define FOR(x) for (int i=0; i<x; i++)


int main() {
    FOR(10) {
        printf("%d ", i);
    }
}

// output: 0 1 2 3 4 5 6 7 8 9
```

How to declare constants

```
const int number;
```

✗

```
const int number;
number = 10;
```

✗

```
const int number = 10;
```

✓

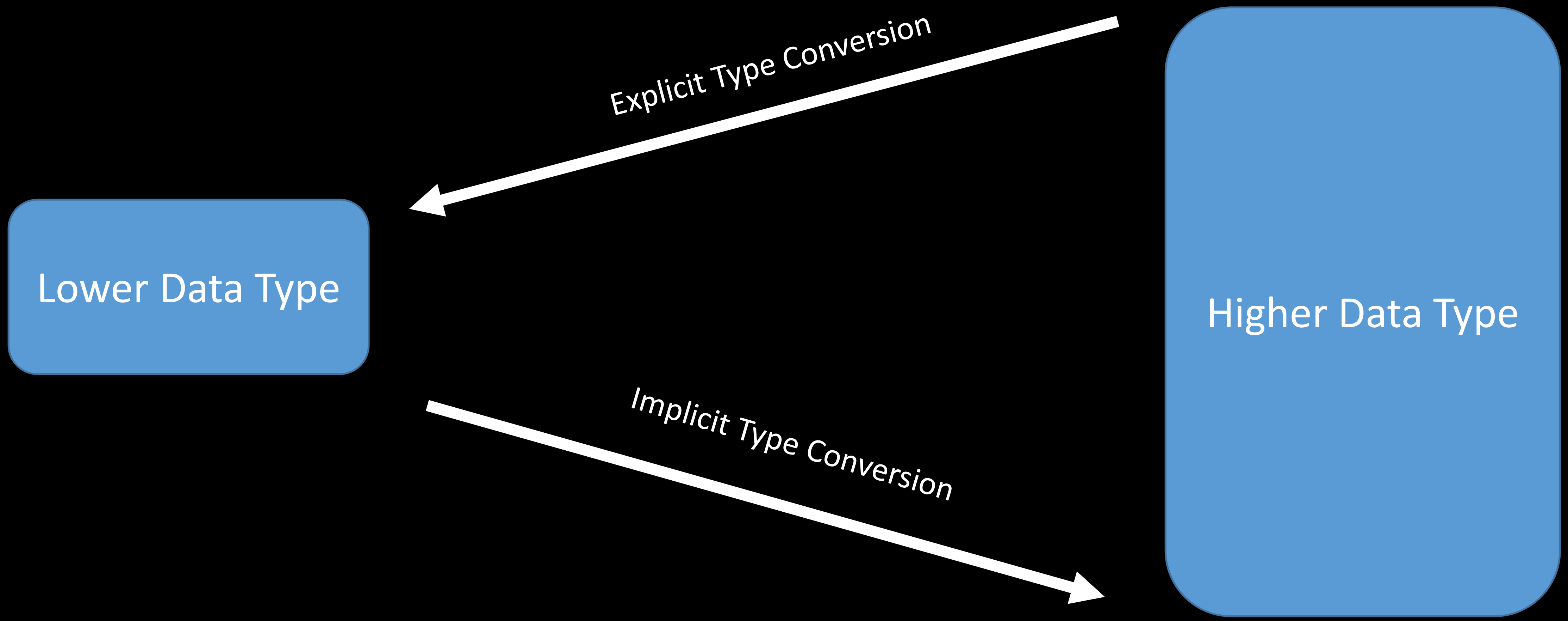
The const declaration can also be used with array arguments, to indicate that the function does not change that array: `int strlen(const char[]);`

Type conversion (Type cast)

Type cast is basically a conversion from one type to another. There are two types of type conversion:

1. `Implicit Type Conversion`

2. `Explicit Type Conversion`


Explicit Type Conversion
Lower Data Type
Higher Data Type
Implicit Type Conversion

Implicit type conversion (automatic type conversion)

```
bool
char
short int
int
unsigned int
long
unsigned
long long
float
double
long double
```

Explicit type conversion

Syntax: (type) expression, example: `(int)n`

```c
#include <stdio.h>

int main(){
    double x = 1.2;
    int sum = (int)x + 1; // Explicit conversion from double to int
    printf("sum = %d", sum); // sum = 2
    return 0;
}
```

Assignment Operators

Operator	Description	Example
=	Simple assignment operator. Assigns values from right side operands to left side operand	C = A + B will assign the value of A + B to C
+=	Add AND assignment operator. It adds the right operand to the left operand and assign the result to the left operand.	C += A is equivalent to C = C + A
-=	Subtract AND assignment operator. It subtracts the right operand from the left operand and assigns the result to the left operand.	C -= A is equivalent to C = C - A
*=	Multiply AND assignment operator. It multiplies the right operand with the left operand and assigns the result to the left operand.	C *= A is equivalent to C = C * A
/=	Divide AND assignment operator. It divides the left operand with the right operand and assigns the result to the left operand.	C /= A is equivalent to C = C / A
%=	Modulus AND assignment operator. It takes modulus using two operands and assigns the result to the left operand.	C %= A is equivalent to C = C % A
<<=	Left shift AND assignment operator.	C <<= 2 is same as C = C << 2
>>=	Right shift AND assignment operator.	C >>= 2 is same as C = C >> 2
&=	Bitwise AND assignment operator.	C &= 2 is same as C = C & 2
^=	Bitwise exclusive OR and assignment operator.	C ^= 2 is same as C = C ^ 2
|=	Bitwise inclusive OR and assignment operator.	C |= 2 is same as C = C | 2

If, else if, else

```
if(boolean_expression 1) {
/* Executes when the boolean expression 1 is true */
} else if( boolean_expression 2) {
/* Executes when the boolean expression 2 is true */
} else if( boolean_expression 3) {
/* Executes when the boolean expression 3 is true */
} else {
/* executes when the none of the above condition is true */
}
```

for loop

```
for ( init; condition; increment ) {
statement(s);
}


#include <stdio.h>

int main () {

    int a;

    /* for loop execution */
    for( a = 10; a < 20; a = a + 1 ){
        printf("value of a: %d\n", a);
    }

    return 0;
}
```

```
Output:


value of a: 10

value of a: 11

value of a: 12

value of a: 13

value of a: 14

value of a: 15

value of a: 16

value of a: 17

value of a: 18

value of a: 19
```

Do While vs While

```
while( condition ){
statements;
}
```

```
do {
statements;
} while( condition );
```

```
#include <stdio.h>

int main() {

    int n = 5;
    while (n>5) {
        printf("%d ", n);
        n -= 1;
    }
    return 0;

} // output:
```

```
#include <stdio.h>

int main() {

    int n = 5;
    do {
        printf("%d ", n);
        n -= 1;
    } while (n>5);
    return 0;

} // output: 5
```

Break

```c
#include <stdio.h>

int main () {

    /* local variable definition */
    int a = 10;

    /* while loop execution */
    while( a < 20 ) {

        printf("value of a: %d\n", a);
        a++;

        if( a > 15) {
            /* terminate the loop using break statement */
            break;
        }
    }
    return 0;
}
```

Output:

```
value of a: 10

value of a: 11

value of a: 12

value of a: 13

value of a: 14

value of a: 15
```

Continue

```c
#include <stdio.h>

int main () {

    /* local variable definition */
    int a = 10;

    /* do loop execution */
    do {
        if( a == 15) {
            /* skip the iteration */
            a = a + 1;
            continue;
        }
        printf("value of a: %d\n", a);
        a++;

    } while( a < 20 );
    return 0;
}
```

```
Output:


value of a: 10
value of a: 11
value of a: 12
value of a: 13
value of a: 14
value of a: 16
value of a: 17
value of a: 18
value of a: 19
```