

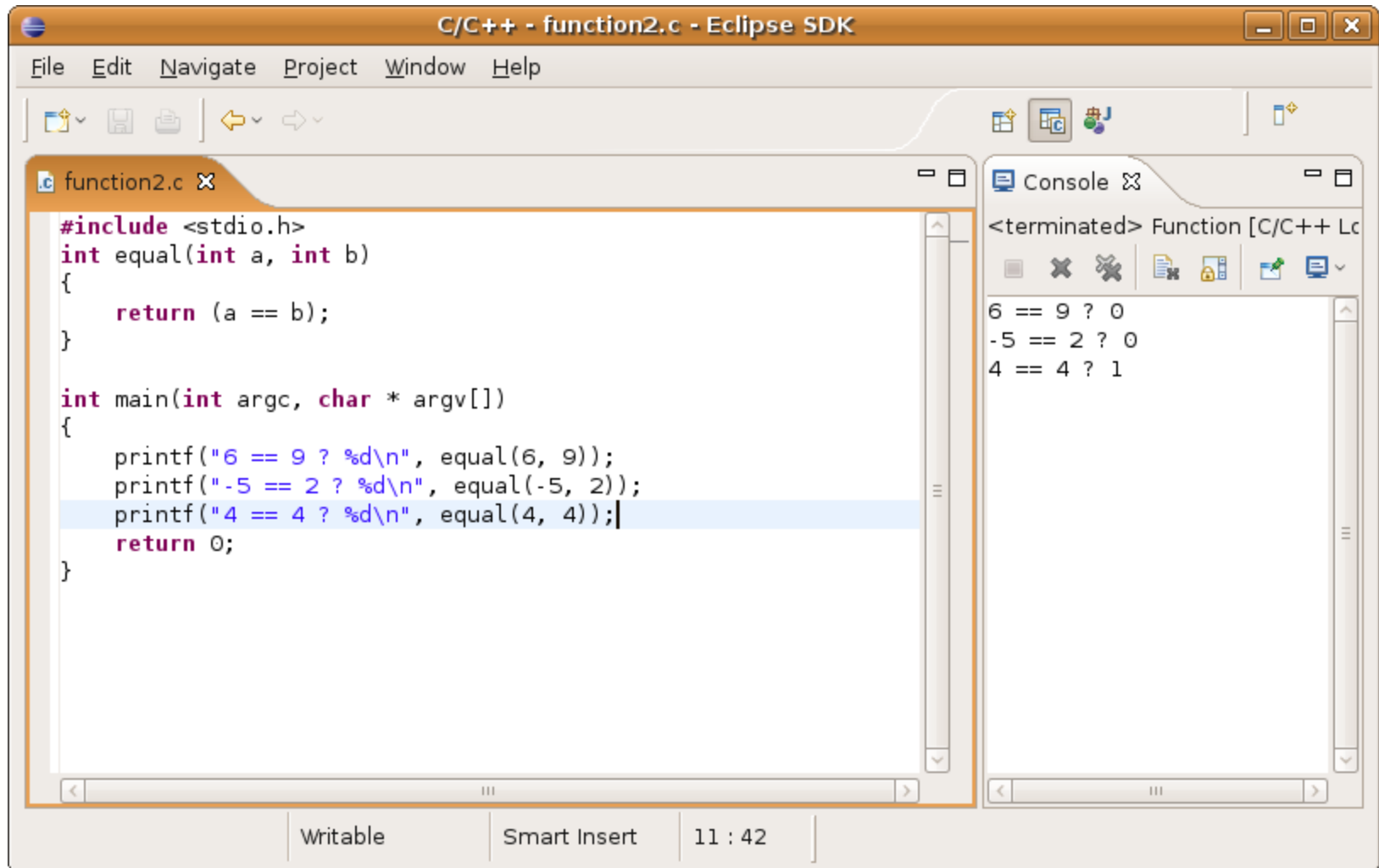
Functions

Yung-Hsiang Lu

Functions in C Programs

- Functions are fundamental building blocks.
- A C program starts at the “main” function.
- Functions are essential for **code reuse**.
- Code reuse: perform similar tasks, differences controlled by the input arguments. Do **not** copy-paste code in multiple places in a program.
- We have used some functions
 - printf
 - scanf
 - strtol

Compare Two Arguments



The screenshot displays the Eclipse IDE with a C program named `function2.c`. The program defines an `equal` function that returns `1` if two integers are equal and `0` otherwise. The `main` function tests this function with three pairs of arguments: `(6, 9)`, `(-5, 2)`, and `(4, 4)`. The console output shows the results of these comparisons: `6 == 9 ? 0`, `-5 == 2 ? 0`, and `4 == 4 ? 1`.

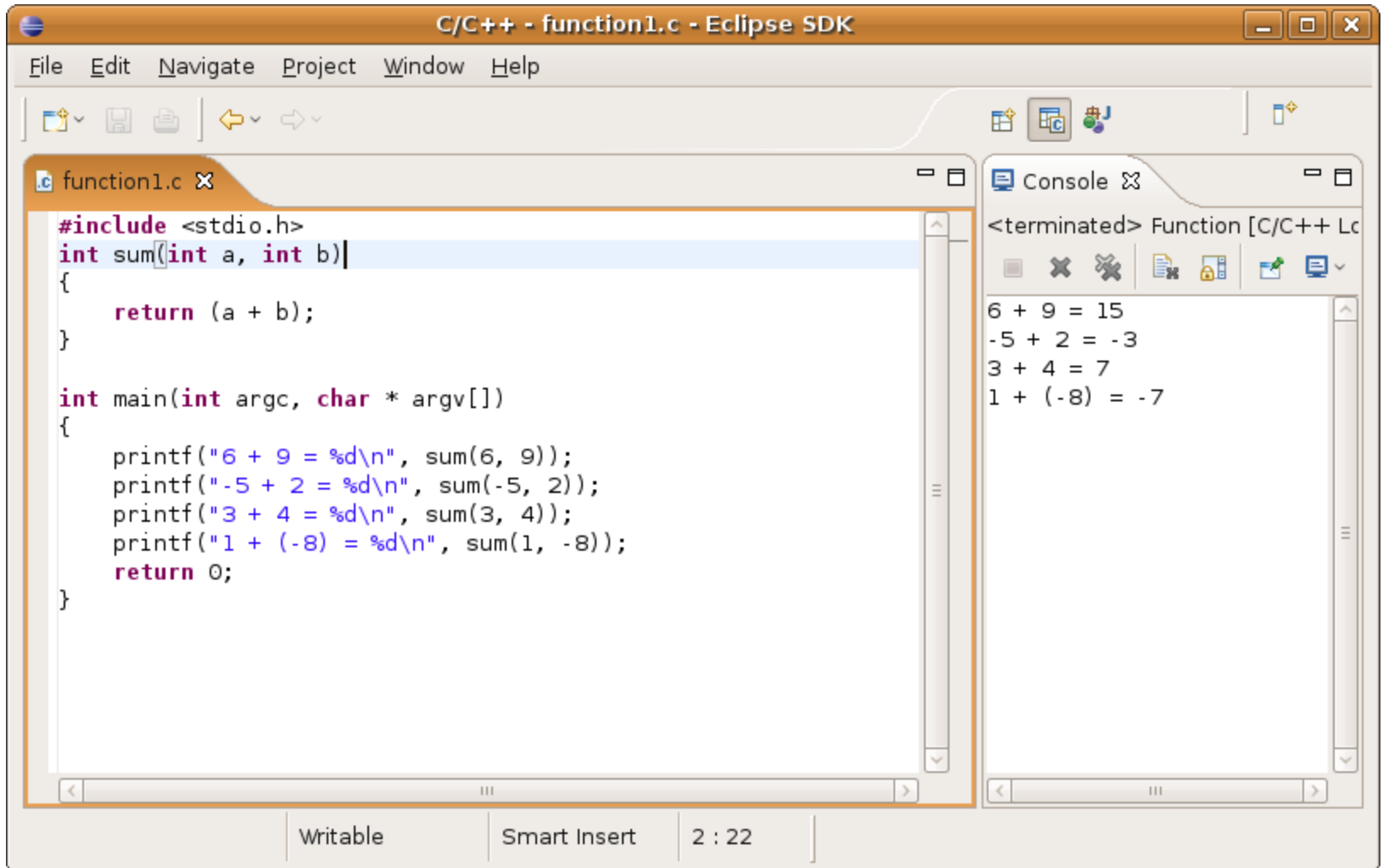
```
#include <stdio.h>
int equal(int a, int b)
{
    return (a == b);
}

int main(int argc, char * argv[])
{
    printf("6 == 9 ? %d\n", equal(6, 9));
    printf("-5 == 2 ? %d\n", equal(-5, 2));
    printf("4 == 4 ? %d\n", equal(4, 4));
    return 0;
}
```

Console Output:

```
<terminated> Function [C/C++ Lc
6 == 9 ? 0
-5 == 2 ? 0
4 == 4 ? 1
```

Function Return Value



The screenshot shows the Eclipse IDE with a C program named `function1.c` open. The program defines a `sum` function that takes two integers and returns their sum. The `main` function calls `sum` with four different pairs of numbers and prints the results. The console output shows the results of these calculations.

```
#include <stdio.h>
int sum(int a, int b)
{
    return (a + b);
}


int main(int argc, char * argv[])
{
    printf("6 + 9 = %d\n", sum(6, 9));
    printf("-5 + 2 = %d\n", sum(-5, 2));
    printf("3 + 4 = %d\n", sum(3, 4));
    printf("1 + (-8) = %d\n", sum(1, -8));
    return 0;
}
```

Console Output:

```
<terminated> Function [C/C++ Lc
6 + 9 = 15
-5 + 2 = -3
3 + 4 = 7
1 + (-8) = -7
```

Modify the Argument

void means return nothing



```
#include <stdio.h>
void incr(int a)
{
    printf("input= %d\n", a);
    a ++;
    printf("before function ends = %d\n", a);
}

int main(int argc, char * argv[])
{
    int x = 8;
    printf("before calling incr, x = %d\n", x);
    incr(x);
    printf("after calling incr, x = %d\n", x);
    return 0;
}
```

Console

<terminated> Function [C/C++ Local Ap]

before calling incr, x = 8
input= 8
before function ends = 9
after calling incr, x = 8

**x is still 8 after
calling the function**

error: inv...expression

Writable

Smart Insert

15 : 47

Call by Value

A function **copies** the **value** to the function's argument.

```
void incr(int a) ...
```

```
int x = 8;
```

```
incr(x);      /* copy x's value to a */
```

```
a ++;        /* a becomes 9, x is still 8 */
```

similar to

```
int x = 8;
```

```
int y = x;
```

```
y ++;        /* y is 9, x is still 8 */
```

Address	Data
somewhere (&x)	8
somewhere (&a)	8 → 9

Address as Argument

The screenshot shows the Eclipse IDE with a C++ project. The main editor displays the source code for `function4.c`. The code defines a function `incr` that takes a pointer to an integer and increments the value it points to. The `main` function initializes a variable `x` to 8, prints its value, calls `incr` with the address of `x`, prints the value again, and returns 0. Annotations include a red arrow pointing to the `&x` argument in the `incr` call, labeled "a pointer", and another red arrow pointing to the `x` variable in the `main` function, labeled "address". The console window on the right shows the output of the program, confirming that the value of `x` is 9 after the function call.

```
#include <stdio.h>
void incr(int * a)
{
    printf("input= %d\n", * a);
    (*a) ++;
    printf("before function ends = %d\n", * a);
}

int main(int argc, char * argv[])
{
    int x = 8;
    printf("before calling incr, x = %d\n", x);
    incr(& x);
    printf("after calling incr, x = %d\n", x);
    return 0;
}
```

Console Output:

```
<terminated> Function [C/C++ Local Ap]
before calling incr, x = 8
input= 8
before function ends = 9
after calling incr, x = 9
```

x is 9 after calling the function

Call by Address

A function **copies** the **address** to the argument.

```
void incr(int * a) ...
```

```
int x = 8;
```

```
incr(& x);    /* copy x's value to a */
```

```
(*a) ++;      /* a becomes 9, so is x */
```


similar to

```
int x = 8;
```

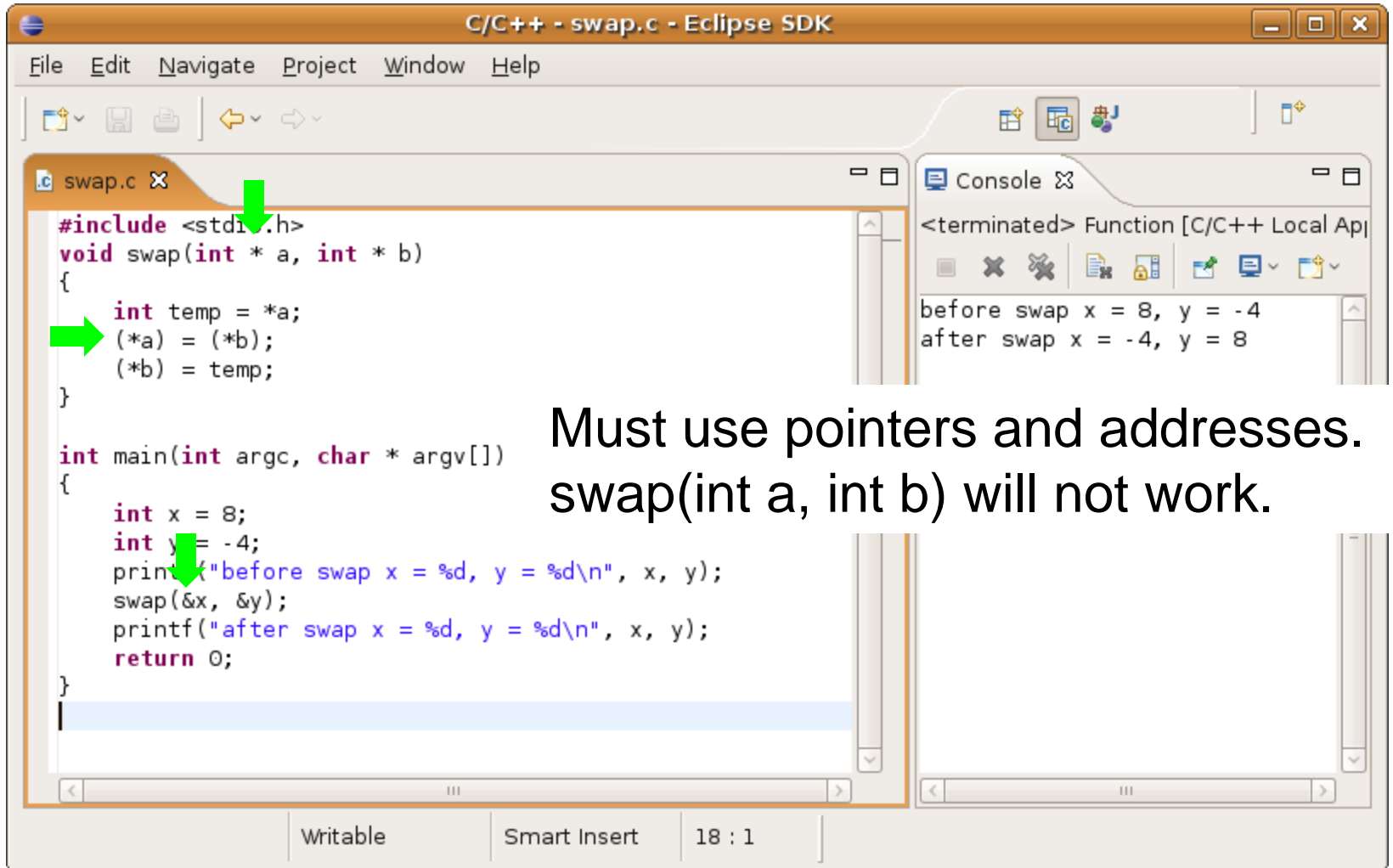
```
int * y = & x;
```

```
(*y) ++;      /* x is 9 */
```

Address	Data
somewhere (&x)	8
somewhere (&a)	& x



Swap Function



```
#include <stdio.h>
void swap(int *a, int *b)
{
    int temp = *a;
    (*a) = (*b);
    (*b) = temp;
}

int main(int argc, char * argv[])
{
    int x = 8;
    int y = -4;
    printf("before swap x = %d, y = %d\n", x, y);
    swap(&x, &y);
    printf("after swap x = %d, y = %d\n", x, y);
    return 0;
}
```

Console Output:

```
<terminated> Function [C/C++ Local Appl]
before swap x = 8, y = -4
after swap x = -4, y = 8
```

Must use pointers and addresses.
swap(int a, int b) will not work.

Recursion (Function calling Itself)

The screenshot displays the Eclipse IDE with a C program named `recursion.c`. The code defines a recursive function `f` to calculate the factorial of a number `a`. The function calls itself with `a-1` until it reaches the base case `a == 1`. The `main` function calls `f(6)` and prints the result.

factorial:
 $f(1) = 1$
 $f(n) = n \times f(n-1)$

Calling itself

```
#include <stdio.h>
int f(int a)
{
    printf("calling f(%d)\n", a);
    if (a == 1) { return 1; }
    else { return a * f(a - 1); }
}

int main(int argc, char * argv[])
{
    printf("f(6) = %d\n", f(6));
    return 0;
}
```

Console Output:

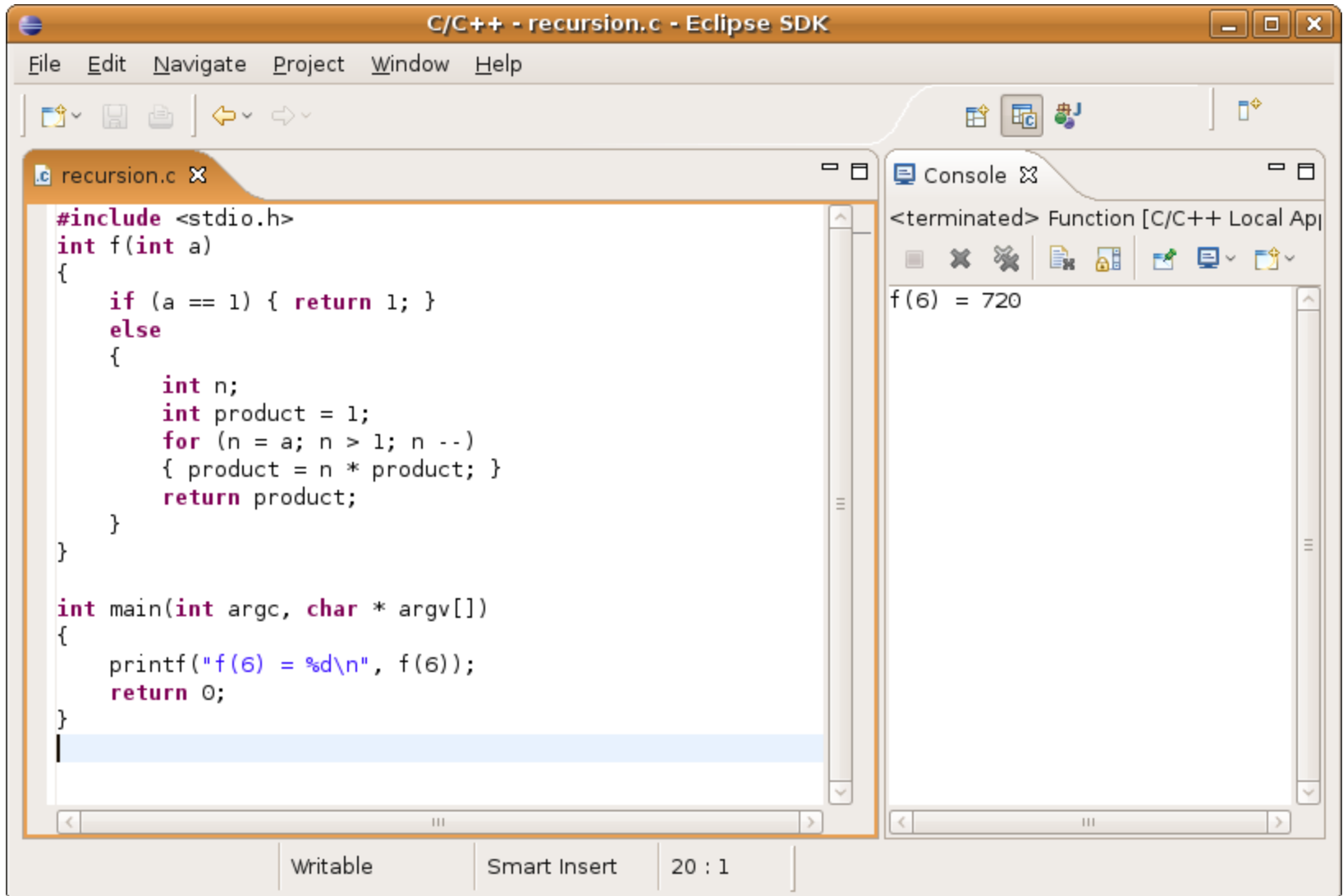
```
<terminated> Function [C/C++ Local Ap]
calling f(6)
calling f(5)
calling f(4)
calling f(3)
calling f(2)
calling f(1)
f(6) = 720
```

Structure of Recursive Calls

- function with arguments
 1. If the terminal condition is satisfied, solve the problem and return the result
 2. Otherwise, divide the problem into smaller parts and solve individual parts by calling the function with new arguments

```
        if (a == 1) { return 1; }      /* step 1 */  
        else { return a * f(a-1); }    /* step 2 */
```
- This is an example of a problem-solving strategy called **divide and conquer**.

Recursion \Rightarrow Iteration



The screenshot shows the Eclipse IDE with a C/C++ project. The editor displays the file `recursion.c` with the following code:

```
#include <stdio.h>
int f(int a)
{
    if (a == 1) { return 1; }
    else
    {
        int n;
        int product = 1;
        for (n = a; n > 1; n --)
        { product = n * product; }
        return product;
    }
}

int main(int argc, char * argv[])
{
    printf("f(6) = %d\n", f(6));
    return 0;
}
```

The right-hand pane shows the Console output, indicating that the program has terminated successfully and printed the result:

```
<terminated> Function [C/C++ Local Appl
f(6) = 720
```

The status bar at the bottom indicates the editor is in 'Writable' mode, 'Smart Insert' is active, and the cursor is at line 20, column 1.

Fibonacci Number

```
#include <stdio.h>
int fib(int a)
{
    if (a == 0) { return 0; }
    if (a == 1) { return 1; }
    return (fib(a-1) + fib(a-2));
}

int main(int argc, char * argv[])
{
    printf("fib(9) = %d\n", fib(9));
    return 0;
}
```

Fibonacci Number:
 $f(0) = 0$
 $f(1) = 1$
 $f(n) = f(n-1) + f(n-2)$

Fibonacci Number by Iteration

When a is large, this is much faster than the previous implementation.

```
#include <stdio.h>
int fib(int a)
{
    if (a == 0) { return 0; }
    if (a == 1) { return 1; }
    int f[a + 1];
    f[0] = 0;
    f[1] = 1;
    int n;
    for (n = 2; n <= a; n++)
    {
        f[n] = f[n-1] + f[n-2];
    }
    return f[a];
}

int main(int argc, char * argv[])
{
    printf("fib(9) = %d\n", fib(9));
    return 0;
}
```

Console Output: fib(9) = 34

Writable Smart Insert 22 : 1

Which statement is correct? You may choose multiple answers.

- ☒ A) A function may have zero, one, two, or more arguments.
- ☒ B) A function may return a value.
- ☒ C) A function may call itself.
- ☒ D) A function may call another function.

Correct - Click anywhere to continue

Incorrect - Click anywhere to continue

Your answer:

You did not answer this question

You must answer the question before continuing

Submit

Clear

What is the value of x after calling the function?

```
void f(int a)
{
    a -= 2;
}
int x = 7;
f(x);
```

x's value
is

Correct - Click anywhere to
continue

Incorrect - Click anywhere to
continue

Your answer:

You did not answer this question

You must answer the question
before continuing

Submit

Clear

Function

Your Score	{score}
Max Score	{max-score}
Number of Quiz Attempts	{total-attempts}

Question Feedback/Review Information Will Appear Here

Continue

Review Quiz