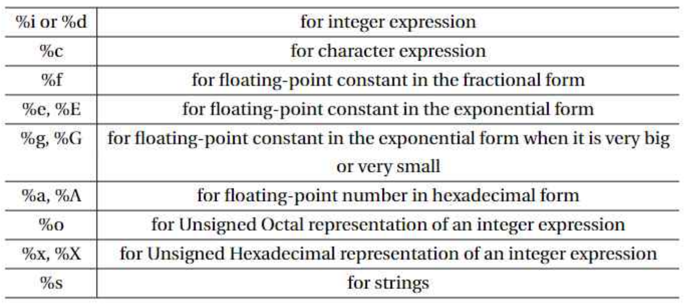
<Format specifier>

%%format specifier for standard output function only in C

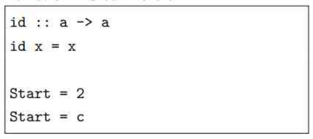


%i or %d	for integer expression
%c	for character expression
%f	for floating-point constant in the fractional form
%e, %E	for floating-point constant in the exponential form
%g, %G	for floating-point constant in the exponential form when it is very big or very small
%a, %A	for floating-point number in hexadecimal form
%o	for Unsigned Octal representation of an integer expression
%x, %X	for Unsigned Hexadecimal representation of an integer expression
%s	for strings

Table 1: Types of format-specifier in C

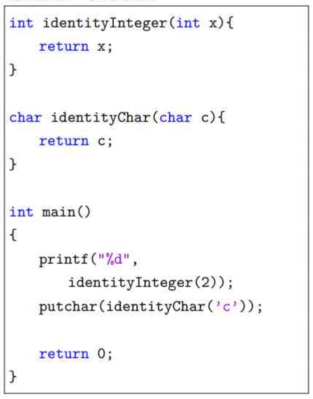
<Problem 1>. identity function (polymorphic type)

Listing 3: The simplest polymorphic function – Clean version



```
id :: a -> a
id x = x

Start = 2
Start = c
```

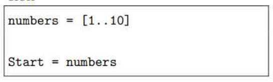
Listing 4: The simplest polymorphic function – C version



```
int identityInteger(int x){
    return x;
}

char identityChar(char c){
    return c;
}

int main()
{
    printf("%d",
        identityInteger(2));
    putchar(identityChar('c'));

    return 0;
}
```

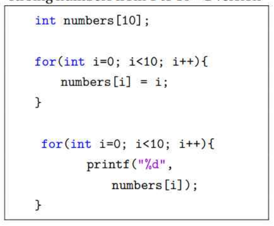
<Problem 2>. simple variable declaration & printing

Listing 5: Variable named 'numbers' for stroing numbers from 1 to 10 - Clean version



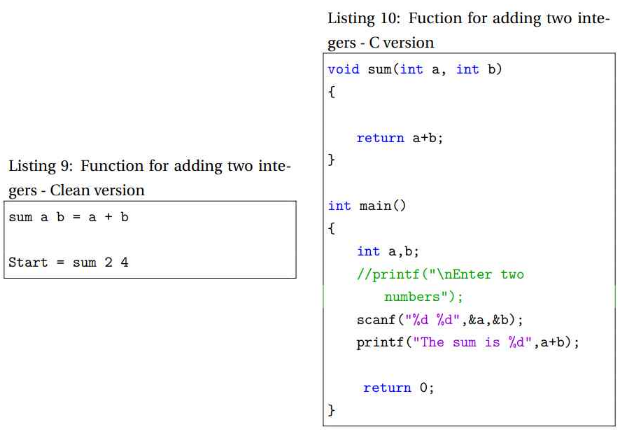
```
numbers = [1..10]

Start = numbers
```

Listing 6: Variable named 'numbers' for stroing numbers from 1 to 10 - C version



```
int numbers[10];

for(int i=0; i<10; i++){
    numbers[i] = i;
}

 for(int i=0; i<10; i++){
        printf("%d",
            numbers[i]);
}
```

<Problem 3>. integer addition function



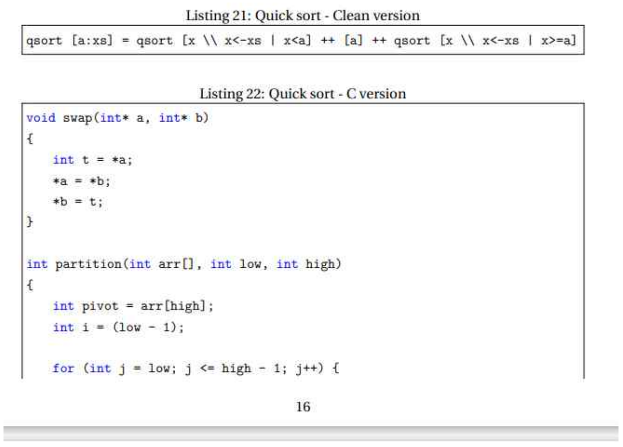
Listing 9: Function for adding two integers - Clean version

```
sum a b = a + b

Start = sum 2 4
```

Listing 10: Fuction for adding two integers - C version

```
void sum(int a, int b)
{

    return a+b;
}

int main()
{
    int a,b;
    //printf("\nEnter two
        numbers");
    scanf("%d %d",&a,&b);
    printf("The sum is %d",a+b);

     return 0;
}
```

<Problem 4>. Quick Sort (recursion)



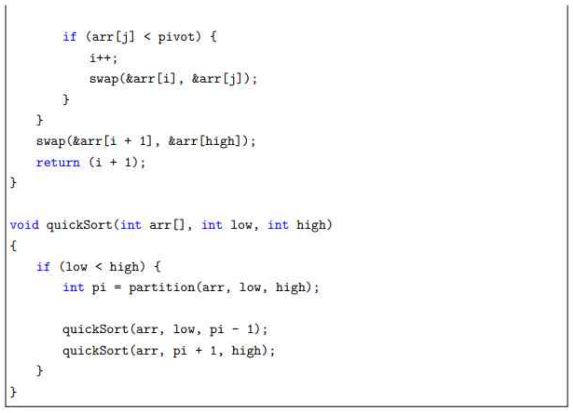
Listing 21: Quick sort - Clean version

```
qsort [a:xs] = qsort [x \\ x<-xs | x<a] ++ [a] ++ qsort [x \\ x<-xs | x>=a]
```

Listing 22: Quick sort - C version

```
void swap(int* a, int* b)
{
    int t = *a;
    *a = *b;
    *b = t;
}

int partition(int arr[], int low, int high)
{
    int pivot = arr[high];
    int i = (low - 1);

    for (int j = low; j <= high - 1; j++) {
```

16



```
        if (arr[j] < pivot) {
            i++;
            swap(&arr[i], &arr[j]);
        }
    }
    swap(&arr[i + 1], &arr[high]);
    return (i + 1);
}

void quickSort(int arr[], int low, int high)
{
    if (low < high) {
        int pi = partition(arr, low, high);

        quickSort(arr, low, pi - 1);
        quickSort(arr, pi + 1, high);
    }
}
```

<Problem 5>. printing array elements

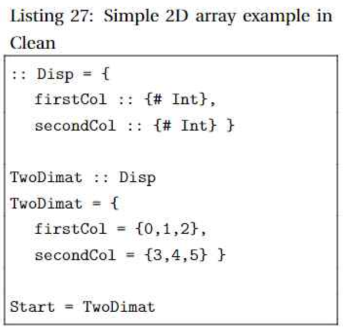
Listing 25: Printing all elements in array - Clean version

```
arr = {1,2,3,4,5}
Start = arr
```

Listing 26: Printing all elements in array - C version

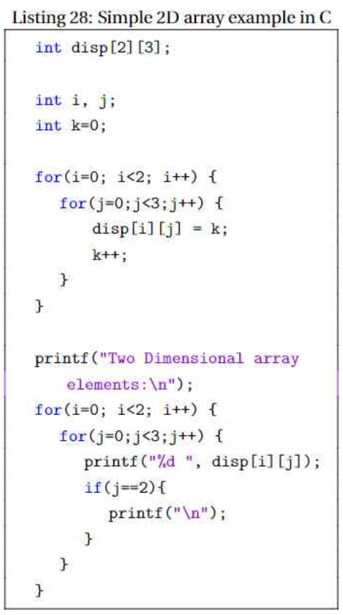
```
void printArray(int arr[], int
    size)
{
    int i;
    printf("Array elements are: ");
    for(i = 0; i < size; i++)
    {
        printf("%d ", arr[i]);
    }
}
int main()
{
    int arr[5] = {1,2,3,4,5};
    printArray(arr, 5);  // Pass
        array directly to function
        printArray
    return 0;
}
```

<Problem 6>. simple 2D array



Listing 27: Simple 2D array example in Clean

```
:: Disp = {
   firstCol :: {# Int},
   secondCol :: {# Int} }

TwoDimat :: Disp
TwoDimat = {
   firstCol = {0,1,2},
   secondCol = {3,4,5} }

Start = TwoDimat
```



Listing 28: Simple 2D array example in C

```
int disp[2][3];

int i, j;
int k=0;

for(i=0; i<2; i++) {
   for(j=0;j<3;j++) {
       disp[i][j] = k;
       k++;
   }
}

printf("Two Dimensional array
    elements:\n");
for(i=0; i<2; i++) {
   for(j=0;j<3;j++) {
      printf("%d ", disp[i][j]);
      if(j==2){
         printf("\n");
      }
   }
}
```

<Problem 7>. simple class & object implementation

Listing 29: Simple class and object implmentation in C

```c
// Define the class (struct)
struct Person {
  char *name;
  int age;
  float height;
};

// Define a function to create a new object (instance) of the class
struct Person createPerson(char *name, int age, float height) {
  struct Person p;
  p.name = name;
  p.age = age;
  p.height = height;
  return p;
}

// Define a function to print the properties of an object
void printPerson(struct Person p) {
  printf("Name: %s\n", p.name);
  printf("Age: %d\n", p.age);
  printf("Height: %.2f\n", p.height);
}

int main() {
  // Create a new object (instance) of the class
  struct Person john = createPerson("John", 25, 1.8);

  // Print the properties of the object
  printPerson(john);

  return 0;
}
```

Listing 30: Simple class and object implmentation in Clean

```
//Define the class (struct)
:: Person = { name :: String, age :: Int, height :: Real }

//Define a function to create a new object(instance) of the class
createPerson :: String Int Real -> Person
createPerson name age height
   = {name = name, age = age, height = height}

//Define a function to print the properties of an object
printPerson :: Person -> String
printPerson {name, age, height}
   = "Name: " +++ name +++ "\n"
   +++ "Age: " +++ toString(age) +++ "\n"
   +++ "Height: " +++ toString(height) +++ "\n"

Start = printPerson john
where
 john = createPerson "John" 25 1.8
```

8. simple file I/O implementation

Listing 31: Copy contents from two files in C

```c
#include <stdio.h>
#include <stdlib.h>

int main()
{
// Open two files to be merged
FILE *fp1 = fopen("file1.txt", "r");
FILE *fp2 = fopen("file2.txt", "r");

// Open file to store the result
FILE *fp3 = fopen("file3.txt", "w");
char c;

if (fp1 == NULL || fp2 == NULL || fp3 == NULL)
{
     puts("Could not open files");
     exit(0);
}

// Copy contents of first file to file3.txt
while ((c = fgetc(fp1)) != EOF)
   fputc(c, fp3);

// Copy contents of second file to file3.txt
while ((c = fgetc(fp2)) != EOF)
   fputc(c, fp3);

printf("Merged file1.txt and file2.txt into file3.txt");

fclose(fp1);
fclose(fp2);
fclose(fp3);
return 0;
}
```

Listing 32: Copy contents from two files in Clean

```
CopyFile :: String String *Files -> *Files
CopyFile inputfname outputfname files
 # (readok,infile,files) = sfopen inputfname FReadText files
 | not readok = abort (inputfname)
 # (writeok,outfile,files) = fopen outputfname FWriteText files
 | not writeok = abort (outputfname)
 # copiedfile = CharFileCopy infile outfile
(closeok,files) = fclose copiedfile files
 | not closeok = abort (outputfname)
  = files
```