31. Typedefs

Objectives

- Create typedef
- Understand that typedef does not create a new type

I will show you how to create an alias for a type. This does not really create a new type but simply gives a type another name.

typedef

Just like we prefer to create constants instead of hardcoding constants:

```
const int MAX_AGE = 100;
```

so we can also give type another (hopefully more readable) name. For instance instead of

```
const int CEO = 0;
const int MANAGER = 1;
const int FULLTIME = 2;
const int PARTTIME = 3;

int employeeCode = FULLTIME;
std::cin >> employeeCode;
```

we can create a name for employee code type:

```
typedef int EmployeeCode;

const EmployeeCode CEO = 0;
const EmployeeCode MANAGER = 1;
const EmployeeCode FULLTIME = 2;
const EmployeeCode PARTTIME = 3;

EmployeeCode employeeCode = FULLTIME;
std::cin >> employeeCode;
```

The statement

```
typedef int EmployeeCode;
```

basically tells C/C++ to replace `EmployeeCode` with `int` before compiling the program. The format for a typedef looks like this:

typedef *[type] [typedef name]*;

Exercise. Create a typedef for `double` with the name `GPA`. Declare a variable `johnDoeGPA` of type `GPA` and initialize it with the value of `3.25`.

It's important to remember that `typedefs` are type aliases. Why is this important? This means that you **cannot** have the following overloaded functions like the following:

```
#include <iostream>

typedef int EmployeeCode;

const EmployeeCode CEO = 0;
const EmployeeCode MANAGER = 1;
const EmployeeCode FULLTIME = 2;
const EmployeeCode PARTTIME = 3;
```

```
void print(int i)
{
    std::cout << i;
}

void print(EmployeeCode code)
{
    std::cout << "Employee code: " << code;
}

int main()
{
    print(42);
    print(CEO);
    return 0;
}
```

Why? Because `EmployeeCode` is just an alias for `int`. In other words as far as the compiler is concerned the code is the same as:

```
...

void print(int i)
{
    std::cout << i;
}

void print(int code)
{
    std::cout << "Employee code: " << code;
}

...
```

And of course you see the problem: There are two functions with the same signatures and that's not allowed.

The following is an example on how to `typedef` a reference type, a constant type, and a pointer type (no surprises):

```
typedef int & intr;      // typedef for int &
typedef const int cint;  // typedef for const int
typedef int * pint;      // typedef for int *

int i = 42;

intr x = i;
cint j = 0;
pint p = &i;
```

Exercise. Make sure you create corresponding typedefs for doubles. Test the typedefs.

Typedef for Array Types

You can also do typedef for array types. It's very common to use an array of two integers to model a point in 2-dimensional space.

```
int p[2] = {3, 6}; // variable p models (3,6)
                   // in 2-dimensional space
```

In the above code, we want to think of p as a point where `p[0]` is the x-coordinate of p while `p[1]` is the y-coordinate of p.

You can do this:

```
typedef int Point [2];
```

This is how you can use this `typedef`:

```
typedef int Point [2];
Point p = {3, 6};
```

The format for creating typedefs for array types is this:

typedef *[type] [array typedef]* [*[size]*];

A very **common typo / error / ignorance / atrocity / etc**. is this:

```
typedef int[2] Point; // BADDDD!!!
```

Remember, the size of the array comes last.

Exercise. Rewrite the following program using the above `typedef` for `Point` and replace the type `int [2]` by `Point.`

```
#include <iostream>

void print(int p[2])
{
    std::cout << "(" << p[0] << ", " << p[1] <<  ")";
}

void add(int sum[2], int p[2], int q[2])
{
    sum[0] = p[0] + q[0];
    sum[1] = p[1] + q[1];
}

int main()
{
    int q[2] = {3, 5};
    print(q);
    std::cout << std::endl;
    return 0;
}
```

Typedef For Pointer Types

Try this:

```
#include <iostream>

typedef int * IntPtr;

int main()
{
    IntPtr p = new int;
    *p = 42;
    std::cout << (*p) + 3 << std::endl;
    delete p;
}
```

Get it?

Exercise. Create a `typedef` for a pointer to doubles and use this `typedef` in this code whenever possible. Fill in the missing code and correct errors (yes there are errors).

```
#include <iostream>

//-----------------------------------------------------
// Allocate memory for p
//-----------------------------------------------------
void constructArray(double * p, int size)
{
    p = new double[size];
}

//-----------------------------------------------------
// Deallocate memory for p
//-----------------------------------------------------
void destructArray(double * p)
{
    delete [] p;
}

//-----------------------------------------------------
// Randomize the array p is pointing to.
//-----------------------------------------------------
void randArray(double * p, int size)
{
    for (int i = 0; i < size; i++)
    {
        p[i] = rand() / RANDMAX;
    }
}

//-----------------------------------------------------
// Print all the element p is pointing to
```

Review the difference between
`delete p;`
and
`delete [] p;`

```
//---------------------------------------------------
void printArray(double * p, int size)
{





}

int main()
{
    srand();

    std::cout << "How many doubles do you want?";
    std::cin >> size;

    while (size > 0)
    {
        double * arr;
        constructArray(arr, size);
        randArray(p, size);
        printArray(p, size);
        destructArray(arr);
        std::cout << "How many doubles do you want?";
        std::cin >> size;
    }
}
```

Exercise. Create a typedef for a ***2***-dimensional array in the following program and use it whenever possible.

```
#include <iostream>

const int SIZE = 5;

// put your typedef here

void print(char x[SIZE][SIZE])
{
    for (int row = 0; row < SIZE; row++)
    {
        for (int col = 0; col < SIZE; col++)
        {
            std::cout << x[row][col];
        }
        std::cout << std::endl;
    }
}

int main()
{
    char a[SIZE][SIZE];
    int row = 0, col = 0;
    for (int i = 0; i < SIZE * SIZE; i++)
    {
        a[row][col] = char(i + 'a');
        col++;
        if (col == SIZE)
        {
            row++;
            col = 0;
        }
    }
    print(a);
    return 0;
}
```

Exercise. Now rewrite the above code so that, without changing the behavior of the program, the code has the following form:

```
#include <iostream>

const int SIZE = 5;

// put your typedef here

void print(char x[SIZE][SIZE])
{
    for (int row = 0; row < SIZE; row++)
    {
        for (int col = 0; col < SIZE; col++)
        {
            std::cout << x[row][col];
        }
        std::cout << std::endl;
    }
}

int main()
{
    char a[SIZE][SIZE];

    for (int row = 0; row < SIZE; row++)
    {
        for (int col = 0; col < SIZE; col++)
        {
            a[row][col] = ____________;
        }
    }
    print(a);
    return 0;
}
```

typedef and Multi-file Compilation

typedefs can be placed in a header file.

When do you want to do that? When a typedef is used by several cpp files.

Not only that. If several functions are closely associated with the typedef, then the prototypes of these functions and the typedef should be in the same header file.

Here's an example:

```
// employee.h

#ifndef EMPLOYEE_H
#define EMPLOYEE_H

#include <iostream>

typedef int EmployeeCode;

const EmployeeCode CEO = 0;
const EmployeeCode MANAGER = 1;
const EmployeeCode FULLTIME = 2;
const EmployeeCode PARTTIME = 3;

void print(EmployeeCode);
EmployeeCode promote(EmployeeCode);
EmployeeCode demote(EmployeeCode);

#endif EMPLOYEE_H
```

```
// employee.cpp

#include <iostream>
#include "employee.h"

void print(EmployeeCode code)
{
    std::cout << code;
}

EmployeeCode promote(EmployeeCode code)
{
    return (code > CEO ? code – 1 : code);
}

EmployeeCode demote(EmployeeCode)
{
    return (code < PARTTIME ? code + 1 : code);
}
```

Exercise. Rewrite this program so that you have two more file for the array-related typedef and functions: a header file `Array.h` containing the typedef and function prototypes and C++ source file `Array.cpp` for the definition of the array-related functions. (See the previous section.)

```
#include <iostream>

//---------------------------------------------------
// Allocate memory for p
//---------------------------------------------------
void constructArray(double * p, int size)
{
    p = new double[size];
}

//---------------------------------------------------
// Deallocate memory for p
//---------------------------------------------------
void destructArray(double * p)
{
    delete [] p;
}

//---------------------------------------------------
// Randomize the array p is pointing to with random
// doubles between 0.0 and 1.0.
//---------------------------------------------------
void randArray(double * p, int size)
{
    for (int i = 0; i < size; i++)
    {
        p[i] = rand() / RANDMAX;
    }
}

//---------------------------------------------------
// Print all the element p is pointing to
//---------------------------------------------------
void printArray(double * p, int size)
{





}

int main()
{
    srand(0);

    std::cout << "How many doubles do you want?";
    std::cin >> size;
    while (size > 0)
    {
        double * arr;
        constructArray(arr, size);
        randArray(p, size);
        printArray(p, size);
        destructArray(arr);

        std::cout << "How many doubles do you want?";
        std::cin >> size;
    }
}
```