Lecture 13: C leftovers

- Type aliases
- Function pointers

Type Aliases: typedef

```
typedef signed char   sint1;
typedef unsigned char uint1;
typedef signed int    sint4;
typedef float real;
// typedef double real; // alternative!
sint4 i;  // signed four-byte integer
uint1 c;  // unsigned one-byte integer
real  r;  // float or double

typedef const char *ccptr;
int strlen(ccptr s) { ... }
```

- Type aliases are new type names for existing types
- Syntax: **typedef** <variable-declaration>**;**
- Variable identifier is treated as type name
- Increases readability and portability
- Can simplify complex type expressions

Function Pointers

```
//pointer to function without parameter returning int
typedef int (*X)(void);

//pointer to function with 2 int params returning nothing
typedef void (*Y)(int,int);
```

- In C, there is no function data type
- But it is possible to declare pointers to functions which point to the first byte of the code
- These pointers can be stored like any other types (e.g. in arrays) or used as parameters
- **These pointers can be used to call functions**
- Declaration: use typedef + function definition. Function name prefixed by * and enclosed in ( )

Calling Functions Via Pointers

- Syntax: **(***<function-pointer>**)(**<parameters>**)**
- Semantics:
  - Evaluate parameter expressions
  - Put values on the stack
  - Call the function the pointer is pointing to
  - Returns the value to the calling environment

Function Pointer Example (1)

```
#include <iostream>
// Binary integer operator: (int,int) -> int
typedef int (*BinIntOp)(int, int);

int plus (int x, int y) { return x+y; }
int minus(int x, int y) { return x-y; }
int mult (int x, int y) { return x*y; }
int divi (int x, int y) { return x/y; }

int main() {
  const int N = 4;
  // f stores N function pointers
  BinIntOp f[N] = { plus, minus, mult, divi };

  for (int i=0; i < N; ++i)
    std::cout << (*f[i])(7,3) << " ";
  return 0;
}

Output: 10 4 21 2
```