# Code generation

```
#
#
#
#
#
#
#
```

```
from casadi import *
```

Let's build a trivial symbolic SX graph

```
x = SX.sym("x")
y = SX.sym("y")
z = x*y+2*y
z += 4*z
```

A Function is needed to inspect the graph

```
f = Function("f", [x,y],[z])
```

The default representation is just the name of the function

```
print f.__repr__()
```

```
f
```

A print statement will call __str__() The result will look like a node-by-node tree evaluation

```
print f
```

```
 Number of inputs: 2
  Input 0 ("i0"): 1-by-1 (dense)
  Input 1 ("i1"): 1-by-1 (dense)
 Number of outputs: 1
  Output 0 ("o0"): 1-by-1 (dense)
@0 = input[0][0];
@1 = input[1][0];
@0 = (@0*@1);
@2 = 2;
@2 = (@2*@1);
@0 = (@0+@2);
@2 = 4;
@2 = (@2*@0);
@0 = (@0+@2);
output[0][0] = @0;
```

The generate method will insert this node-by-node evaluation in exported C code

```
f.generate("f_generated")
```

This is how the exported code looks like:

```
print file('f_generated.c').read()
```

```
/* This function was automatically generated by CasADi */
#ifdef __cplusplus
extern "C" {
#endif

#ifdef CODEGEN_PREFIX
  #define NAMESPACE_CONCAT(NS, ID) _NAMESPACE_CONCAT(NS, ID)
  #define _NAMESPACE_CONCAT(NS, ID) NS ## ID
  #define CASADI_PREFIX(ID) NAMESPACE_CONCAT(CODEGEN_PREFIX, ID)
#else /* CODEGEN_PREFIX */
  #define CASADI_PREFIX(ID) f_generated_ ## ID
#endif /* CODEGEN_PREFIX */

#include <math.h>

#ifndef real_t
#define real_t double
#define to_double(x) (double) x
#define to_int(x) (int) x
#endif /* real_t */

/* Pre-c99 compatibility */
#if __STDC_VERSION__ < 199901L
real_t CASADI_PREFIX(fmin)(real_t x, real_t y) { return x<y ? x : y;}
#define fmin(x,y) CASADI_PREFIX(fmin)(x,y)
real_t CASADI_PREFIX(fmax)(real_t x, real_t y) { return x>y ? x : y;}
#define fmax(x,y) CASADI_PREFIX(fmax)(x,y)
#endif

#define PRINTF printf
real_t CASADI_PREFIX(sq)(real_t x) { return x*x;}
#define sq(x) CASADI_PREFIX(sq)(x)

real_t CASADI_PREFIX(sign)(real_t x) { return x<0 ? -1 : x>0 ? 1 : x;}
#define sign(x) CASADI_PREFIX(sign)(x)

static const int CASADI_PREFIX(s0)[] = {1, 1, 0, 1, 0};
#define s0 CASADI_PREFIX(s0)
/* f */
int f(const real_t** arg, real_t** res, int* iw, real_t* w, int mem) {
  real_t a0=arg[0] ? arg[0][0] : 0;
  real_t a1=arg[1] ? arg[1][0] : 0;
  a0=(a0*a1);
  real_t a2=2.;
  a2=(a2*a1);
  a0=(a0+a2);
  a2=4.;
  a2=(a2*a0);
  a0=(a0+a2);
  if (res[0]!=0) res[0][0]=a0;
  return 0;
}

void f_incref(void) {
}

void f_decref(void) {
}

int f_n_in(void) { return 2;}

int f_n_out(void) { return 1;}
```

```c
const char* f_name_in(int i){
  switch (i) {
  case 0: return "i0";
  case 1: return "i1";
  default: return 0;
  }
}

const char* f_name_out(int i){
  switch (i) {
  case 0: return "o0";
  default: return 0;
  }
}

const int* f_sparsity_in(int i) {
  switch (i) {
  case 0: return s0;
  case 1: return s0;
  default: return 0;
  }
}

const int* f_sparsity_out(int i) {
  switch (i) {
  case 0: return s0;
  default: return 0;
  }
}

int f_work(int *sz_arg, int* sz_res, int *sz_iw, int *sz_w) {
  if (sz_arg) *sz_arg = 2;
  if (sz_res) *sz_res = 1;
  if (sz_iw) *sz_iw = 0;
  if (sz_w) *sz_w = 3;
  return 0;
}


#ifdef __cplusplus
} /* extern "C" */
#endif
```