

**The COOP Cheat Sheet**

|  |
| --- |
| **MEMORY MANAGEMENT** |
| At the very beginning of your program,  should initialize the memory management system that you’d like to use throughout your program,  by calling the ***init\_global\_memory*** function.  You can choose between limited size memory, that will be allocated at the beginning of your program, (meaning, no more dynamic allocations will be done) like this:  init\_global\_memory(1024 \* 10, LIMITED\_SIZE\_MEMORY);   * this call will allocate 10 Kb on the heap.   **OR**  You can just use the ususal heap of C:  init\_global\_memory(0, HEAP\_BASED\_MEMORY);   * NOTE! When using LIMITED\_SIZE\_MEMORY the actual memory size is a little smaller than what you ask, since part of the pre-allocated buffer is used to store meta data about the memory blocks. |

|  |  |
| --- | --- |
| **CONDITIONS** | |
| The C++ way | The COOP Way |
| if (condition) {  // ...  } else if (another\_condition) {  // ...  } else if (a\_third\_condition) {  // ...  } else {  // ...  } | IF(condition) {  // ...  } ELSE\_IF(another\_condition) {  // ...  } ELSE\_IF(a\_third\_condition) {  // ...  } ELSE {  // ...  }  END\_IF; |
| for (int i = 0; i < 10; i++)  {  if (i > 2) {  continue;  }  // ...  if (i > 5) {  break;  }  } | FOR(int i = 0; i < 10; i++)  {  IF(i > 2) {  CONTINUE; // <-- note the CONTINUE  }  // ...  IF(i > 5) {  BREAK; // <-- note the BREAK  }  }END\_LOOP; |
| while (condition) {  // ...  } | WHILE(condition) {  // ... same BREAK and CONTINUE keywords are  available  }END\_LOOP; |

|  |  |
| --- | --- |
| **FUNCTIONS** | |
| The C++ way | The COOP Way |
| // ... in the header file: void calc\_area(int height, int width, int\* out\_product); | // ... in the header file: FUN\_DECL(calc\_area, int height, int width, int\* out\_product); |
| // ... in the source file: void calc\_area(int height, int width, int\* out\_product) {  \*out\_product = height \* width;  } | // ... in the source file: FUN\_IMPL(calc\_area, int height, int width, int\* out\_product) {  \*out\_product = height \* width;  } END\_FUN; |
| Call a function: | |
| int area = 0;  calc\_area(5, 6, &area); | int area = 0;  FUN(calc\_area) 5, 6, &area CALL; |

|  |  |
| --- | --- |
| **EXCEPTIONS** | |
| The C++ way | The COOP Way |
| try {  throw new std::exception("ERROR! Help!");  } catch (std::exception & e) {  std::cout << e.what() << std::endl;  } | TRY {  THROW\_MSG("ERROR! Help!");  }CATCH{  printf("%s", LAST\_EXCEPTION\_ERROR\_MSG);  }END\_TRY; |
|  | // Utilities  ASSERT(condition);  THROW\_MSG\_UNLESS(condition, msg);  ASSERT\_NOT\_NULL(x) |

|  |  |
| --- | --- |
| **CLASSES** | |
| The COOP Way Header file: | |
| DEF\_CLASS(Rectangle);  // Define member fields here. e.g:  int hight;  int width;  END\_DEF(Rectangle);  FUNCTIONS(Rectangle, /\*ctor params:\*/ int hight, int width);  MEM\_FUN\_DECL(Rectangle, /\*func. name:\*/ GetSideLength, /\*func. params:\*/int side\_idx, int\* ret\_val);  MEM\_FUN\_DECL(Rectangle, /\*func. name:\*/ GetArea, /\*func. params:\*/ int\* ret\_val);  END\_FUNCTIONS(Rectangle); | |
| The C++ Way Header file: | |
| class Rectangle  {  public:  // Define member fields here. e.g:  int hight;  int width;  Rectangle(/\*ctor params:\*/ int hight, int width);  ~Rectangle();  virtual void /\*func. name:\*/ GetSideLength(/\*func. params:\*/int side\_idx, int\* ret\_val);  virtual void /\*func. name:\*/ GetArea(/\*func. params:\*/ int\* ret\_val);  }; | |
| Create an instance on the stack, call a member function: | |
| The C++ way | The COOP Way |
| void do\_something()  {  Rectangle rect(10, 3);  int area = 0;  rect.GetArea( & area );  } | FUN\_IMPL(do\_something)  {  CREATE(Rectangle, rect), 10, 3 CALL;  int area = 0;  MFUN(&rect, GetArea), & area CALL;  }END\_FUN; |

|  |
| --- |
| The COOP Way Source file: |
| // \*\*\*\*\*\*\*\*\*\*\* COOP: \*\*\*\*\*\*\*\*\*\*\*\*  #include "Rectangle.h"  DEF\_CTOR(Rectangle, int hight, int width)  {  \_this->hight = hight;  \_this->width = width;  }  END\_CTOR  DEF\_DTOR(Rectangle)  {  //Nothing to do here...  }  END\_DTOR  MEM\_FUN\_IMPL(Rectangle, GetArea, int\* ret\_val)  {  \*ret\_val = \_this->hight \* \_this->width;  }  END\_FUN;  MEM\_FUN\_IMPL(Rectangle, GetSideLength, int side\_idx, int\* ret\_val)  {  \*ret\_val = side\_idx == 0 ? \_this->hight : \_this->width;  }  END\_FUN;  INIT\_CLASS(Rectangle)  BIND(Rectangle, GetArea);  BIND(Rectangle, GetSideLength);  END\_INIT\_CLASS(Rectangle) |
| The C++ Way Source file: |
| // \*\*\*\*\*\*\*\*\*\*\* C++: \*\*\*\*\*\*\*\*\*\*\*\*  Rectangle::Rectangle(int hight, int width)  {  this->hight = hight;  this->width = width;  }  Rectangle::~Rectangle()  {  //Nothing to do here...  }  void Rectangle::GetArea(int\* ret\_val)  {  \*ret\_val = this->hight \* this->width;  }  void Rectangle::GetSideLength(int side\_idx, int\* ret\_val)  {  \*ret\_val = side\_idx == 0 ? this->hight : this->width;  } |

|  |
| --- |
| **INHERITANCE** |
| The COOP Way Header file: |
| DEF\_DERIVED\_CLASS(Cube, Rectangle);  int depth; // <-- another member variable, in addition to the inherited ones.  END\_DEF\_DERIVED(Cube);  DERIVED\_FUNCTIONS(Cube, Rectangle, int hight, int width, int depth);  FUN\_OVERRIDE(Rectangle, GetArea, int\* retVal); // <-- all methods are virtual  MEM\_FUN\_DECL(Cube, JustReturn10, int\* retVal); // <-- non inherited method  END\_DERIVED\_FUNCTIONS(Cube); |
| The C++ Way Header file: |
| class Cube : public Rectangle  {  public:  int depth; // <-- another member variable, in addition to the inherited ones.  Cube(int hight, int width, int depth);  ~Cube();  void GetArea(int\* retVal) override; // <-- overrides virtual method  void JustReturn10(int\* retVal); // <-- non inherited method  }; |

|  |
| --- |
| The COOP Way Source file: |
| DEF\_DERIVED\_CTOR(Cube, Rectangle, int hight, int width, int depth) SUPER, hight, width ME  {  \_this->depth = depth;  }  END\_DERIVED\_CTOR  DEF\_DERIVED\_DTOR(Cube, Rectangle)  **Calling the base class constructor**  {  }  END\_DERIVED\_DTOR  FUN\_OVERRIDE\_IMPL(Cube, Rectangle, GetArea, int\* retVal)  {  \*retVal = 0;  FUN\_BASE(\_this, GetArea), retVal CALL; // <-- this is how you call a function of base class !  // And this is how you access member variables of base class:  THROW\_MSG\_UNLESS(retVal == \_this->\_base.hight \* \_this->\_base.width,   "What is going on here?!");  \*retVal = (\*retVal) \* \_this->depth;  }  END\_FUN  MEM\_FUN\_IMPL(Cube, JustReturn10, int\* retVal)  {  \*retVal = 10;  }  END\_FUN  INIT\_DERIVED\_CLASS(Cube, Rectangle);  BIND\_OVERIDE(Cube, Rectangle, GetArea);  BIND(Cube, JustReturn10);  END\_INIT\_CLASS(Cube) |
| The C++ Way Source file: |
| Cube::Cube(int hight, int width, int depth) : Rectangle(hight, width)  {  this->depth = depth;  }  Cube::~Cube()  {  }  void Cube::GetArea(int\* retVal)  {  \*retVal = 0;  ((Rectangle\*)this)->GetArea(retVal);  \*retVal = (\*retVal) \* this->depth;  }  void Cube::JustReturn10(int\* retVal)  {  \*retVal = 10;  } |

|  |  |
| --- | --- |
| **CONTAINERS** | |
| The C++ way | The COOP Way |
| std::vector<char>  std::vector<int>  std::vector<float> | Vector\_char  Vector\_int  Vector\_float  Vector\_object // any type of object |
|  | // N-dimensional array:  Tensor\_char  Tensor\_int  Tensor\_float |
| std::shared\_ptr<int[]> | SharedPodPtr |