CGC Proposal

Lieyang Chen, Tianze Huang, Zhuoxuan Li, Fanhao Zeng
{lc3548, th2887, zl2890, fz2320}@columbia.edu

January 23, 2021

1 Overview

The CGC programming language is a language built upon C language, but comes with many handy syntax and features, and most importantly added garbage collection feature. In addition to the primitive data types and pointer similar to C, we provides a new built-in object type `Array` that could be used to wrap all primitive data types.

- Garbage collection

  The CGC language provides a garbage collector which will finds unused objects on the heap and deletes them to free up memory without the user knowing it. However the user can still manually empty free up memory similar to C.

- Handy syntax/features

  The CGC language will provide built-in methods for `Array` type, such as `len`, `push`, `pop`.The CGC may also support some keyword like `auto`, which is capable of deducing the type of a declared variable. The CGC also support control flows like `For` Loops

2 Language Details

2.1 Data Types and Operations

Data types:

- int, char, float, Array
- pointer

  The primitive data types of CGC are `int, char, float`. Besides them, CGC provides a new built-in object type `Array` that could be used to wrap

all primitive data types as well as itself (e.g., `c` below). `Array` keyword will evaluate the size and check whether the size is legal(size>0) at run time. If the size is not explicitly written(e.g., `Array<int>d` in the example), the size will be determined by counting the number of elements it is assigned. In addition, if the `Array` object is successfully created, it will implicitly contain the value of array size inside its data structure. `Array` object is initialized by copying each element of the right operand, and the `Array` is type mutable. For example, the statement `b[1] = '3'` is legal in CGC. Beyond primitive data types and Array, CGC also allows user to access the address of a variable through a pointer. A pointer can be created using this way, `int* x = &y`.

Array examples:

```
Array<int>[4] a = {1,2,3,4}; // Array of ints
Array<char>[4] b = "1234"; // Array of chars
b[1] = '3'; // Array indexing
Array<Array<int>>[2] c = {{1,2},{1,2,3}}; //Array of arrays
Array<int> d = {1,2}; //Array without explicite size
```

Operations:

- int: +, -, *, /, ==, ++, –, +=, -=, <, >, =<, >=, =, !=, &&, ||, !, *(dereference), &(reference)
- char: +, -, *, /, ==, ++, -, +=, -=, <, >, =<, >=, =, !=, &&, ||, !, *(dereference), &(reference)
- float: +, -, *, /, -, +=, -=, <, >, =<, >=, =, ==, *(dereference), &(reference)
- Array: +(concatenate), =, ==, *(dereference), &(reference), [ ]

  Examples:

```
Array<int>[4] a = {1,2,3,4};
Array<int>[1] b = {5};
return a + b //returns {1,2,3,4,5}}
```

2.2 Keywords

- `while, for, if, else, elseif, return, int, float, char, Array, void, const, len, resize, push, pop, auto, new, delete, class, static, printf`

2.3 Control Flow

2.3.1 For Loops

There are two ways to write a `For` Loop in CGC:

```
//first format
Array<int>[3] array = {1,2,3};
for (auto& x in array) {
    ++x;
}

//second format
for (int i = 0; i < len(x); i++) {
    ++x[i];
}
```

2.3.2 While Loops

CGC provide a more convenient and concise way to write a `while` loop:

```
//first format
int k = 0;
while(k < 10) {
        --k;
}
//Second format
while(int i = 0; i < 10) {
    // can choose to declare i inside while expression if i is not
        defined yet
    --i;
}
```

2.4 Functions

1. C-like function declaration

```
int fun(Array<int>& x, int y) {
  return x[len(x)-1] + y;
}

//support recursion
void rec(int i) {
    if (i != 0) {
        i--;
        rec(i);
```

```
    }
}

//class
class foo {
    void get_bar() {
        return bar;
    }
    int bar;
}
```

2.5 Comments

Single-line comments use double backslashes (//). Multi-line comments are denoted with a /* */ notation. For example:

```
int x = 3 // This is a single-line comment
/*
And this is a multi-line comment
*/
```

2.6 Memory

The CGC language will by defalut "pass by value" when calling a function. An argument could be passed by reference by using &. (e.g., ...). The CGC language will use the `new` keyword to allocate memory on heap. A garbage collector will be implemented to take care of the memory allocated on the heap.

3 Declarable Storage Class

The CGC language maintains a keyword `static`, which is a declarable storage class. A variable without a storage class declaration in a block, will be treated as a local variable of that block, and it will be discarded once the end of the block has been executed.In contrast, a static local variable is also local in a block, but it retains its value independently and will not be discarded on exit of the block.

```
int Example1()
{
    int a = 0;
    a++;
    return a;
    //default local variable's scope only within the block
}

int Example2()
```

```
{
    static int a = 0;
    a++;
    return a;
    //static local variable persists until end of the program
}

int main()
{
    while(int i = 0; i < 3)
    {
        printf("Round %d\n", i + 1);
        printf("auto variable = %d\n", Example1());
        printf("static variable = %d\n", Example2());
        i++;
    }
//By default, a local variable initialized every iteration
//static variable contains previous value and increment
    return 0;
}
```

4 Type Inference

CGC has the capability of type inference, by using `auto` keyword. Any variable declared to be `auto` with valid initialization expression, its type will be automatically deducted.

```
int autoExample()
{
    //type inference
    auto a, b, c = 1, 2, 3; //int
    auto i, j, k = 2.0, 3.4, 5.5; //float
    auto x, y, z = 'X', 'Y', 'Z'; //char
    auto *d = new auto(i); //pointer
}
```

5 Bound checking

CGC will do bound checking so that a function will not unintentionally modify unrelated data. For example, the following code will results in a compile error.

```
Array<int>[5] x = {1,2,3,4,5}; //Some initialization
x[6] = -1; // Compile error
```

6 Examples

```
//garbage collector example:
int example1()
{
    int *example = new int(1);
    *example = 10;
    return *example;
    //garbage collector will free the memory later
}

// A simple binary search algorithm in CGC
class example2
{
  int binarySearch(int target)
  {
    if (array[0] == target) return 0;
    // binary search
    while (int left, right = len[array] - 1; left < right)
    {
      int mid = left + (right - left) / 2;
      if (array[mid] == target) return mid;
      else if (array[mid] < target) left = mid + 1;
      else right = mid;
    }
    return -1;
  }

  int setArray(Array<int> a)
  {
    array = a;
  }

  Array<int> array;

};
```