Proposed storage allocator and garbage collection for basic 5.21 compiled to z88dk zcc. Take the following specifications and psudo code and write the full mbasic 20205 string allocator and garbage collector.

- Written in c
- For background on how garbage collection in real 5.21 works on an 8080 see help/mbasic/implementation/string-allocation-and-garbage-collection.md
- The number of strings is known at compile time:
    - Simple string variables $ at end of name or defstr
    - Cells in a string array
    - Function parameters
    - String temporaries used with a statement (all statements can share if that is ok with CSE)
    - Some number of temporaries for use inside string functions
    - The number of strings is known at compile time
    - The number of string is set in the c code generated as MB25_NUM_STRINGS for example #define MB25_NUM_STRINGS (27)
    - Per string we add a is_const flag. This is used when for code like A$="foo" in this case the string descriptor for A$ gets its is_const set to 1. The data part of the string points to the f in foo in the c constant string. Thus constant data can be converted to a basic string without using any heap spacwe.
    - Per string we add a writeable bit. In 5.21 the data of a string was only ever written once. That allowed string functions like left$ and right$ and mid$ (the non assignment mid) to point the substring into the bigger string. We take a mixed approach, when newly allocated string descriptors point to writeable data. The writeable bit set to 1. So for example of b$ points to a writeable string of say 100 characers, then you need to assing a new value to b$ that is 50 characters long and not in permanent data then the 50 chars can be copyed to the b$ data and b$ len set to 50. This will have 50 bytes un pointed to the next garbage collection will clean up. On the issue of permanent data. if you had done b$=c$ then c$ data and len would be copied over to b$ and they would share the space. The copy assignment need only be done if it the source is not long lived.
      left$, right$ and mid$(non assignment) return a shared pointer into the source string if it is long lived. They also turn off the writeable bit in the source as now the source is immutable.

What follows is pseudo code written in c style for how the new string allocation and garbage collection will work.

In real mbasic 5.21 a string looks like:

```
struct mb521_string {
  uint8 len;
 char * data;
};
```

The problem being with so little data and no space for an extra copy it uses the above described N^2 garbage collection. For mbasic2025 we add some extra info to allow a much faster garbage collection.

```
struct mb25_string_R {
 uint16
   str_id:14,   /* allow for 16K string objects */
   is_const:1,
   writeable:1;
 uint8 len;
 uint8 *data;
};

typedef struct mb25_string_R mb25_string_t, *mb25_string_pt;

struct mb25_globals_R {
 uint16  pool_size;
 uint16 allocator;
 uint8 *pool;    /* pointer to malloced string pool of all string data */
  mb25_string_t mb25_strings[MB25_NUM_STRINGS];
};
struct mb25_globals_R mb25_global;
void mb25_string_alloc(uint16 str_id,uint16 str_size, char *input_str) {
 mb25_string_pt str_loc=mb25_global.strings+str_id;
size_t len=strlen(input_str)
str_loc.is_const=1;
str_loc.len=len;
str_loc.writeable=0;
str_loc.data=(uint8 *) input_char;
}


void mb25_string_alloc(uint16 str_id,uint16 str_size) {
  uint8 try_gc=1;
 mb25_string_pt str_loc=mb25_global.strings+str_id;
  while(TRUE) {
    uint16 available=mb25_global.pool_size-mb25_global.allocator;
    if (available >= str_size) {
      str_loc.data=mb25_global.pool+mb_25.global.allocator;
     mb_25.global.allocator += str_size;
    }
   If(try_gc <1)
    fatal_error("out of string space")
  try_gc--;
```

```
 mb25_garbage_collect()
}
}

/*
       in original mbasic 5.21 strings where moved down one at a time scanning all strings
 *    each time giving N^2 performance.  We sort the string pointers table by the string pointers
*    so they can be compacted in one pass.  Then we sort them back into palce for usage
*/
void mb25_garbage_collect() {
 sort_ mb25_strings_by_data(); /* sort by data field, cast data to uint16. lowest number first*/
 mb25_string_pt current_loc=mb25_global. mb25_strings;
 for(i=0;i< MB25_NUM_STRINGS; i++) {
  /* if lowest string doesn't end at the allocator move it down to move free space above it*?
  if (!current_loc->  end of string != mb_25_global.allocator+mb25_global.pool) {

    move string down to end at top of

   mb25_global.allocator+=current_loc.len'

}

current_loc++;

 }

 sort_ mb25_strings_by_str_id();

}
```