```
void setup()
{
   Serial.begin(9600);
}

void loop()
{
   int x = 0, y = 0, z = 0;

   uint16_t varAddress1 = (uint16_t)&x;
   uint16_t varAddress2 = (uint16_t)&y;
   uint16_t varAddress3 = (uint16_t)&z;

   Serial.println("Addr1: Ox" + String(varAddress1, HEX));
   Serial.println("Addr2: Ox" + String(varAddress2, HEX));
   Serial.println("Addr3: Ox" + String(varAddress3, HEX));

   // Valeur fin de RAM:
   Serial.println("RAMEND: Ox" + String(RAMEND, HEX));

   delay(10000);
}


// void setup() éludée...
void loop()
{
   uint8_t* buffer1 = (uint8_t*)malloc(10);
   uint8_t* buffer2 = (uint8_t*)malloc(10);

   Serial.println("Addr1: Ox" + String((uint16_t)buffer1, HEX));
   Serial.println("Addr2: Ox" + String((uint16_t)buffer2, HEX));

   free(buffer1);
   free(buffer2);

   delay(10000);
}


void PrintVariable(uint16_t address, int len)
{
   // On place un pointeur à l'adresse:
   uint8_t* pointer = (uint8_t*)address;

   Serial.print("Ox"); // On commence l'écriture.
   for (int i = 0 ; i < len ; i++)
   {
      uint8_t val = *(pointer+i); // valeur d'1 octet.
      Serial.print((val < 16 ? "0" : "") + String(val, HEX));
   }
   Serial.println(""); // CRLF

}


void loop()
{
  long maVar = 32;
```

```
  // Obtention de l'adresse avec '&'
  // et de la taille avec sizeof():
  PrintVariable((uint16_t)&maVar, sizeof(maVar));

  delay(10000);
}

void loop()
{
  byte tableau[] = {0, 1, 2, 3, 4, 5, 6, 7, 8, 9};
  PrintVariable((uint16_t)&tableau, sizeof(tableau));

  delay(10000);
}


void loop()
{
  // On crée un buffer:
  uint8_t* buffer = (uint8_t*)malloc(10);
  // On initialise tous les octets à 0xa0:
  memset(buffer, 0xa0, 10);

  // Imaginons maintenant qu'on ne
  // connaisse pas la taille du buffer:

  // Adresse du buffer dans le tas:
  uint16_t address = (uint16_t)buffer;

  uint8_t* pointer = (uint8_t*)(address - 2);

  // Taille:
  uint16_t len = 0;
  // On copie la valeur à l'adresse -2 dans len:
  memcpy(&len, pointer, sizeof(uint16_t));

  pointer += 2;
  PrintVariable((uint16_t)pointer, len);

  free(buffer);  // On libère !

  delay(10000);
}

// Ecrit les taille de chaque section:
void PrintMemusage()
{
  extern char* __data_start;
  extern char* __data_end;
  extern char* __bss_start;
  extern char* __bss_end;
  extern char* __heap_start;
  // prochaine allocation du tas disponible:
  extern char* __brkval;

  uint16_t memReg = (uint16_t)&__data_start;
  uint16_t memData = (uint16_t)&__data_end - (uint16_t)&__data_start;
  uint16_t memBss = (uint16_t)&__bss_end - (uint16_t)&__bss_start;
  uint16_t memHeapTtl = ((uint16_t)__brkval == 0) ?  0 : (__brkval - __malloc_heap_start);
```

```
    uint16_t memStack = RAMEND - SP; // SP = pointeur de pile.
    uint16_t freeRam = SP - ((int)&__heap_start + memHeapTtl);

    Serial.println("Registres: " + String(memReg) + " octets");
    Serial.println(".Data: " + String(memData) + " octets");
    Serial.println(".Bss: " + String(memBss) + " octets");
    Serial.println("Tas1: " + String(memHeapTtl) + " octets");
    Serial.println("Pile: " + String(memStack) + " octets");
    Serial.println("Libre: " + String(freeRam) + " octets");
}

void loop()
{
    uint8_t* buff1 = (uint8_t*)malloc(10);
    uint8_t* buff2 = (uint8_t*)malloc(20);

    PrintMemusage();

    free(buff2);   // Libérée !
    free(buff1);  // Délivrée !

    delay(10000);
}


void loop()
{
    uint8_t* buff1 = (uint8_t*)malloc(10);
    uint8_t* buff2 = (uint8_t*)malloc(20);

    PrintMemusage();

    memset(buff1, 0xe4, 10);
    memset(buff2, 0xe4, 20);

    free(buff2);   // Libérée !
    free(buff1);  // Délivrée !
    delay(10000);
}
```