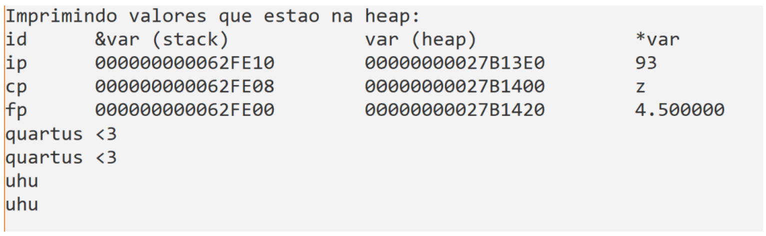
Resumo P2 ICC

```
// RESUMO P2
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

int* alocaVetorA(int N) {
      return (int*) malloc(N * sizeof(int));
}

void alocaVetorB(int** p, int N) {
      *p = (int*) malloc(N * sizeof(int));
}

char* alocaString(int N) {
      return (char*) malloc(N * sizeof(char));
}

void desalocaVetor(int* vet) {
      free(vet);
}

void desalocaString(char* str) {
      free(str);
}

int** alocaMatriz(int row, int col) {
      int** m = NULL;

      m = (int**) calloc(row, sizeof(int
*)); // aloca N linhas como vetores de int*
      for (int i = 0; i < row; i++) {
            m[i] = (int *) calloc(col,
sizeof(int)); // para cada linha, M colunas de
tipo int
      }
      return m;
}

void alocarMatriz2(int ***m, int row, int col)
{
      int i;
      *m = (int**) malloc(row * sizeof(int*));

      for (i = 0; i < row; i++) {
            (*m)[i] = (int*) malloc(col *
sizeof(int));
      }
}

void desalocaMatriz(int row, int col, int** m)
{
      int i;
      for (i = 0; i < row; i++) { // cada vetor
(coluna) dentro da matriz tem que ser liberado
na heap
            free(m[i]);
      }
      free(m); // libera o vetor de linhas na
heap
}

//////////////////////
void AulaHeap() {
      int* ip; // stack (alocacao automatica)
      ip = (int*) malloc(1 * sizeof(int)); //
alocacao dinamica (heap)
      *ip = 93;

      char* cp; // stack
      cp = (char*) malloc(1 * sizeof(char)); //
alocacao dinamica (heap)
      *cp = 'z';

      float* fp; // stack
      fp = (float*) malloc(1 *
sizeof(float)); // alocacao dinamica (heap)
      *fp = 4.5;

      printf("Imprimindo valores que estao na
heap:\n");
      printf("id\t&var (stack)\t\tvar (heap)\t\
t*var\n");

      printf("ip\t%p\t%p\t%d\n", &ip, ip, *ip);
      printf("cp\t%p\t%p\t%c\n", &cp, cp, *cp);
      printf("fp\t%p\t%p\t%f\n", &fp, fp, *fp);

      free(ip);
      free(cp);
      free(fp);

      int* vetor = calloc(50, sizeof(int));
//aloca e manda 0
      free(vetor);
}

void AulaString() {
      char* string1 = alocaString(500);

      string1[0] = 'q';
      string1[1] = 'u';
      string1[2] = 'a';
      string1[3] = 'r';
      string1[4] = 't';
      string1[5] = 'u';
      string1[6] = 's';
      string1[7] = ' ';
      string1[8] = '<';
      string1[9] = '3';
      string1[10] = '\0';
      printf("%s\n", string1);
      strcpy(string1, "quartus <3");
      printf("%s\n", string1);
      gets(string1);
      printf("%s\n", string1);
      free(string1);
}

int main(int argc, char* argv[]) {
      AulaHeap();
      AulaString();

      return 0;
}
```

///output



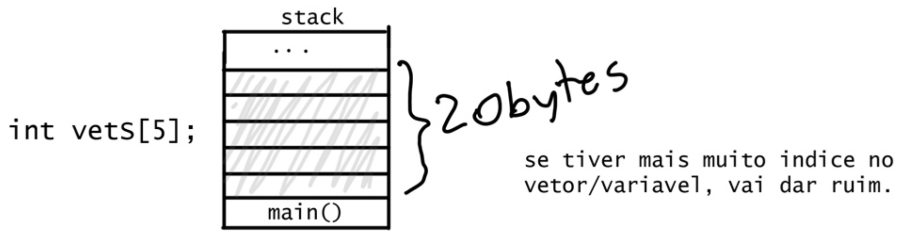
```
Imprimindo valores que estao na heap:
id      &var (stack)            var (heap)              *var
ip      000000000062FE10        00000000027B13E0        93
cp      000000000062FE08        00000000027B1400        z
fp      000000000062FE00        00000000027B1420        4.500000
quartus <3
quartus <3
uhu
uhu
```

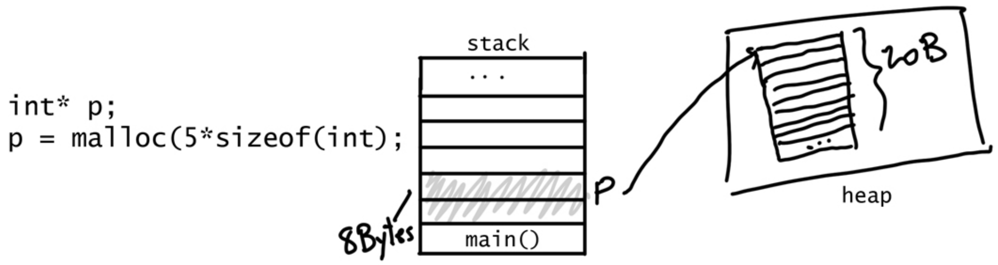
“The heap is a large pool of memory that can be used dynamically – it is also known as the “free store”. This is memory that is not automatically managed – you have to explicitly allocate (using functions such as malloc), and deallocate (e.g. free) the memory. Failure to free the memory when you are finished with it will result in what is known as a memory leak – memory that is still “being used”, and not available to other processes. Unlike the stack, there are generally no restrictions on the size of the heap (or the variables it creates), other than the physical size of memory in the machine. Variables created on the heap are accessible anywhere in the program.

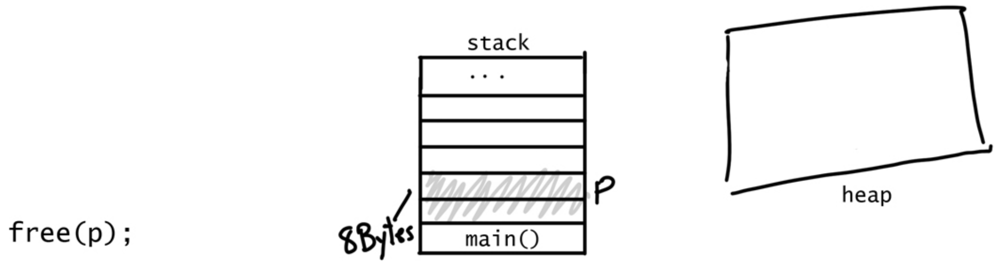
Oh, and heap memory requires you to use pointers.

A summary of the heap:

the heap is managed by the programmer, the ability to modify it is somewhat boundless
in C, variables are allocated and freed using functions like malloc() and free()
the heap is large, and is usually limited by the physical memory available
the heap requires pointers to access it”







```
void atribuiMatriz(int **m, int row, int col) {
        int i, j;
        for (i = 0; i < row; i++) {
                for (j = 0; j < col; j++) {
                        m[i][j] = (i+1)*(j+2);
                }
        }
}
```

```
void imprimeMatriz(int **m, int row, int col) {
        int i, j;
        for (i = 0; i < row; i++) {
                for (j = 0; j < col; j++) {
                        printf("%d ", m[i][j]);
                }
                printf("\n");
        }
}
```