Example – memory distributor

Free memory is organized in the string of blocks. Every block contains size, pointer to the next block and area of free memory.

Algorithms:

- "first fit"
- "best fit"

On each machine there exists variable type forcing strongest constraints – if objects of the same type should be stored under certain address then also objects of any other type may be stored under the same address. On some machines "double" is the most demanding type, on other it suffices to take either int or long.

Every free block contains pointer to the next free block in the string, number determining size of the block and amount of the free memory itself; such control information at the beginning of the block is called "header". To simplify the control of the position in memory all blocks have got size being a multiple of the header size, and header is placed correctly. That result has been obtained with the aid of the union containing the desired header structure and object of type forcing strongest constraints.

```
typedef long Align;
           /* position for objects of long type */
union header { /* block header */
   struct {
      union header *ptr;  /* next free block */
      unsigned size;      /* size of that block  */
   } s;
   Align x;   /* forcing the block position */
} ;

typedef union header Header;
```

Union member of type Align is nether used; it serves exclusively to force the proper position of the whole header with respect to the most demanding variable type.

```
static Header base;
/* empty string at the beginning */
static Header *freep=Null;
/* beginning of the string of empty blocks */

/*  malloc: general memory distributor */
void *malloc(unsigned nbytes)
{
   Header *p, *prevp;
   Header *morecore(unsigned);
   unsigned nunits;  /* number of required portions */

   nunits=(nbytes+sizeof(Header)-1)/sizeof(Header)+1;
   if ((prevp=freep) == NULL) {  /* empty string */
      base.s.ptr = freep = prevp = &base;
      base.s.size = 0;
   }
   for (p=prevp->s.ptr;; prevp=p, p=p->s.ptr) {
      if (p->s.size >= nunits) {
                  /* sufficiently large block */
         if (p->s.size == nunits)
            /* exactly as required */
            prev->s.ptr = p-<s.ptr;
         else {       /* too big; assign the end */
            p->s.size -= nunits;
            p += p->s.size;
            p->s.size = nunits;
         }
         freep = prevp;
         return (void *)(p+1);
      }
      if (p == freep)
         /* whole string has been searched */
         if ((p = morecore(nunits)) == NULL)
            return NULL;  /* lack of memory */
   }
}
```

Example (Application of INT 25h function to read sektor number 13 of a disquette placed in drive a: and store its conent in array Sektor):

```
/* Function reads logical sektor No. 13
   of a disquette in drive A:
   using interrupt 25h of DOS
   and stores it in array Sektor  */

#include <stdio.h>
#include <dos.h>
unsigned char far Sektor[512];
union REGS RejWej, RejWyj;
struct SREGS RejSeg;
void czytajSektorInt_25(void)
{
   RejSeg.ds = FP_SEG(Sektor); /* buffer address*/
   RejWej.x.bx = FP_OFF(Sektor);
   RejWej.h.al = 0x00;   /* drive A: */
   RejWej.x.cx = 0x01;   /*  one sector */
   RejWej.x.dx = 0x0d;   /* first sector number 13 */
   int86x(0x25,&RejWej,&RejWyj,&RejSeg);
   asm pop ax;  /* remove unnecessary word from
                   the buffer */
   if ((RejWej.x.cflag & 0x01) == 0x01)
      printf("\nError number:  %x\n",RejWyj.x.ax);
}
```

Declarations of unions and structures:

```
union REGS {
   struct WORDREGS x;
   struct BYTEREGS h;
};

struct BYTEREGS {
   unsigned char al, ah, bl, bh;
   unsigned char cl, ch, dl, dh;
};

struct WORDREGS {
   unsigned int ax, bx, cx, dx;
   unsigned int si, di, cflag, flags;
};
```