

Summary of Lesson 15

Functions , introduced in this lesson

```
// changing the content of binbuf
void binbuf_set (Binbuf *bin, t_symbol *msg, short argc, t_atom *argv);

// appending t_atoms to a Binbuf
void binbuf_append (Binbuf *bin, t_symbol *msg, short argc, t_atom
*argv);

// get one atom from a binbuf
short binbuf_getatom (Binbuf *bin, long
*typeOffset, long *stuffOffset, t_atom *result);

// instantiate a binbuf
Binbuf *binbuf_new (void);

// read a max file from the disk and load it on the binbuf
short binbuf_read (Binbuf *bin, char *filename, short volume, short
binaryFlag);

// write a max file from binbuf and save it on the disk
short binbuf_write (Binbuf *bin, char *filename, short volume, short
binaryFlag);

// open a open dialog for user
short open_dialog (char *filename, short *path, OSType *dstType,
SFTypelist typelist, short numtypes);

// get fqn from file and path id pair
short path_topathname(short path, char *file, char *name);

// make internal object
t_object *newinstance (t_symbol *className, short argc, t_atom *argv);

// send message to internal object
void *typedmess (void *receiver, t_symbol *message, short argc, t_atom
*argv);

// get table data
short table_get (t_symbol *tableName, long ***dstHandle, long *dstSize);

// mark dirty flag
short table_dirty (t_symbol *tableName);
```

ex1.c

#include "ext.h"

// binbuf

```

typedef struct {
    t_object b_ob;
    void *binbuf;
    void *out;
} t_ex1;

void ex1_bang(t_ex1 *x);

void *ex1_new(t_symbol *s, short argc, t_atom *argv);
void ex1_free(t_ex1 *x);

void ex1_append(t_ex1 *x, t_symbol* message, short argc, t_atom *argv);
void ex1_clear(t_ex1 *x);

void *ex1_class;

void main(void)
{
    setup((t_messlist **)&ex1_class, (method)ex1_new, (method)ex1_free,
(short)sizeof(t_ex1), 0L, 0L);
    addbang((method)ex1_bang);
    addmess((method)ex1_append, "append", A_GIMME, 0L);
    addmess((method)ex1_clear, "clear", 0L);
}

void ex1_append(t_ex1 *x, t_symbol* message, short argc, t_atom *argv)
{
    binbuf_append(x->binbuf, 0L, argc, argv);
}

void ex1_clear(t_ex1 *x)
{
    binbuf_set(x->binbuf, 0L, 0L, 0L);
}

void ex1_bang(t_ex1 *x)
{
    t_atom holder;
    long to, so;

    to = 0;
    so = 0;
    while(!binbuf_getatom(x->binbuf, &to, &so, &holder))
        postatom(&holder);
    post("");
}

void *ex1_new(t_symbol *s, short argc, t_atom *argv)
{

```

```

    t_ex1 *x;
    x = (t_ex1*)newobject(ex1_class);
    x->binbuf = binbuf_new();

    return x;
}

void ex1_free(t_ex1 *x)
{
    freeobject(x->binbuf);
}

```

ex2.c

```

#include "ext.h"

// read max file into binbuf

typedef struct {
    t_object b_ob;
    void *binbuf;
    void *out;
} t_ex2;

void ex2_bang(t_ex2 *x);

void *ex2_new(t_symbol *s, short argc, t_atom *argv);
void ex2_free(t_ex2 *x);
void ex2_open(t_ex2 *x);

void *ex2_class;

void main(void)
{
    setup((t_messlist **)&ex2_class, (method)ex2_new, (method)ex2_free,
(short)sizeof(t_ex2), 0L, 0L);
    addbang((method)ex2_bang);
    addmess((method)ex2_open, "open", 0L);
}

void ex2_open(t_ex2 *x)
{
    long filter = FOUR_CHAR_CODE('maxb');
    long dstType;
    char filename[256];
    short path;
}

```

```

    open_dialog(filename,&path,&dstType,&filter,1);
    binbuf_read(x->binbuf,filename, path,1);

}

void ex2_bang(t_ex2 *x)
{
    t_atom holder;
    long to,so;

    to = 0;
    so = 0;
    binbuf_eval(x->binbuf,0L,0L,0L);

    while(!binbuf_getatom(x->binbuf,&to,&so,&holder))
    {
        postatom(&holder);
        if(holder.a_type == A_SEMI)
            post("");
    }
}

void *ex2_new(t_symbol *s, short argc, t_atom *argv)
{
    t_ex2 *x;
    x = (t_ex2*)newobject(ex2_class);
    x->binbuf = binbuf_new();

    return x;
}

void ex2_free(t_ex2 *x)
{
    freeobject(x->binbuf);
}

```

ex3.c

```

#include "ext.h"

// read max file into binbuf and modify it

typedef struct {
    t_object b_ob;
    void *binbuf;
    void *out;
} t_ex3;

```

```

void *ex3_new(t_symbol *s, short argc, t_atom *argv);
void ex3_free(t_ex3 *x);
void ex3_open(t_ex3 *x);

void *ex3_class;

void main(void)
{
    setup((t_messlist **)&ex3_class, (method)ex3_new, (method)ex3_free,
(short)sizeof(t_ex3), 0L, 0L);
    addmess((method)ex3_open, "open", 0L);
}

void ex3_open(t_ex3 *x)
{
    long filter = FOUR_CHAR_CODE('maxb');
    long dstType;
    char filename[256];
    char newname[256];
    short path;
    int count;
    t_atom holder;
    long to,so;
    void *resultbuf = binbuf_new();

    to = 0;
    so = 0;
    count = 0;
    open_dialog(filename,&path,&dstType,&filter,1);
    binbuf_read(x->binbuf,filename, path,1);

    while(!binbuf_getatom(x->binbuf,&to,&so,&holder))
    {
        if(holder.a_type == A_SYM)
        {
            if(!strcmp(holder.a_w.w_sym->s_name,"number"))
            {
                t_atom atom;
                SETSYM(&atom, gensym("flonum"));
                binbuf_append(resultbuf, 0L, 1,&atom );
                count++;
            }
            else
            {
                binbuf_append(resultbuf, 0L, 1, &holder);
            }
        }
        else
    }
}

```

```

        {
            binbuf_append(resultbuf, 0L, 1, &holder);
        }

    }
    post("%d replaced",count);
    sprintf(newname, "%s_converted", filename);
    binbuf_write(resultbuf,newname,path,2);

    freeobject(resultbuf);
}

```

```

void *ex3_new(t_symbol *s, short argc, t_atom *argv)
{
    t_ex3 *x;
    x = (t_ex3*)newobject(ex3_class);
    x->binbuf = binbuf_new();

    return x;
}

void ex3_free(t_ex3 *x)
{
    freeobject(x->binbuf);
}

```

ex4.c

```

#include "ext.h"

// automatic comment

typedef struct {
    t_object b_ob;
    void *binbuf;
    void *out;
} t_ex4;

void *ex4_new(t_symbol *s, short argc, t_atom *argv);
void ex4_free(t_ex4 *x);
void ex4_open(t_ex4 *x);

void *ex4_class;

void main(void)

```

```

{
    setup((t_messlist **)&ex4_class, (method)ex4_new, (method)ex4_free,
(short)sizeof(t_ex4), 0L, 0L);
    addmess((method)ex4_open, "open", 0L);
}

```

```

void ex4_open(t_ex4 *x)
{
    long filter = FOUR_CHAR_CODE('maxb');
    long dstType;
    char filename[256];
    char newname[256];
    char comment[24];
    long argholder[2];
    short path, flag = 0;
    int count= 0, argcount = 0;
    t_atom holder;
    long to=0, so =0;
    void *resultbuf = binbuf_new();
    open_dialog(filename, &path, &dstType, &filter, 1);
    binbuf_read(x->binbuf, filename, path, 1);

    while(!binbuf_getatom(x->binbuf, &to, &so, &holder))
    {
        binbuf_append(resultbuf, 0L, 1, &holder);
        if(holder.a_type == A_SYM)
        {
            if(!strcmp(holder.a_w.w_sym->s_name, "number"))
            {
                flag = 1;
            }

        }
        else if(holder.a_type == A_LONG)
        {
            if(flag == 1)
            {
                if(argcount == 0)
                    argholder[argcount] = holder.a_w.w_long;
                else if(argcount == 1)
                    argholder[argcount] = holder.a_w.w_long;

                argcount++;
            }
        }
        else if(holder.a_type == A_SEMI)
        {
            if(flag == 1)
            {
                sprintf(comment, "numberbox-%d", count);
            }
        }
    }
}

```

```

        binbuf_vinsert(resultbuf, "sslllls",
gensym("#P"),gensym("comment"), argholder[0]+40, argholder[1], 120, 196617,
gensym(comment) );
        flag = 0;
        argcount = 0;
        count++;

    }

}

}
post("%d added", count);
sprintf(newname, "%s_converted", filename);
binbuf_write(resultbuf, newname, path, 2);
freeobject(resultbuf);
}

void *ex4_new(t_symbol *s, short argc, t_atom *argv)
{
    t_ex4 *x;
    x = (t_ex4*)newobject(ex4_class);
    x->binbuf = binbuf_new();

    return x;
}

void ex4_free(t_ex4 *x)
{
    freeobject(x->binbuf);
}

```

ex5.c

```

#include "ext.h"

// how to share the value between two different objects

typedef struct {
    t_object b_ob;
    t_symbol *name;
    long value;
    void *out;
} t_ex5;

void ex5_bang(t_ex5 *x);
void ex5_int(t_ex5 *x, long input);

void *ex5_new(t_symbol *s);

```



```

void ex5_free(t_ex5 *x);

void *ex5_class;

void main(void)
{
    setup((t_messlist **)&ex5_class, (method)ex5_new, 0L, (short)sizeof(t_ex5),
0L, A_SYM,0L);
    addbang((method)ex5_bang);
    addint((method)ex5_int);
}

void ex5_bang(t_ex5 *x)
{
    if(0 != strcmp(ob_sym(x->name->s_thing)->s_name,"ex5"))
    {
        error("it's not my business");
    }
    else
    {
        outlet_int(x->out,((t_ex5*)(x->name->s_thing))->value);
    }
}

void ex5_int(t_ex5 *x, long input)
{
    x->value = input;
    x->name->s_thing = (t_object *)x;
}

void *ex5_new(t_symbol *s)
{
    t_ex5 *x;
    x = (t_ex5*)newobject(ex5_class);
    x->out = intout(x);
    x->name = s;
    x->name->s_thing = (t_object *)x;

    return x;
}

```

ex6.c

```

#include "ext.h"

// assignment
// share and output list!!

```

```

typedef struct {
    t_object b_ob;
    t_symbol *name;
    t_atombuf *atoms;
    void *out;
} t_ex6;

void ex6_bang(t_ex6 *x);
void ex6_list(t_ex6 *x, t_symbol *message, short argc, t_atom *argv);

void *ex6_new(t_symbol *s);
void ex6_free(t_ex6 *x);

void *ex6_class;

void main(void)
{
    setup((t_messlist **)&ex6_class, (method)ex6_new, 0L, (short)sizeof(t_ex6),
    0L, A_SYM, 0L);
    addbang((method)ex6_bang);
    addmess((method)ex6_list, "list", A_GIMME, 0);
}

void ex6_bang(t_ex6 *x)
{
    if(0 != strcmp(ob_sym(x->name->s_thing)->s_name, "ex6"))
    {
        error("it's not my business");
    }
    else
    {
        t_atombuf* a = ((t_ex6*)(x->name->s_thing))->atoms;
        outlet_list(x->out, 0L, a->a_argc, a->a_argv);
    }
}

void ex6_list(t_ex6 *x, t_symbol *message, short argc, t_atom *argv)
{
    if(x->atoms)
        atombuf_free(x->atoms);

    x->atoms = atombuf_new(argc, argv);
    x->name->s_thing = (t_object *)x;
}

void *ex6_new(t_symbol *s)
{

```

```

    t_ex6 *x;
    x = (t_ex6*)newobject(ex6_class);
    x->out = outlet_new(x,0L);
    x->name = s;
    x->name->s_thing = (t_object *)x;
    return x;
}

void ex6_free(t_ex6 *x)
{
    atombuf_free(x->atoms);
}

```

ex7.c

```

#include "ext.h"

// making an object internally.

typedef struct {
    t_object b_ob;
    t_object *a_print;
} t_ex7;

void ex7_anything(t_ex7 *x, t_symbol *message, short argc, t_atom *argv);
void *ex7_new(void);
void ex7_free(t_ex7 *x);
void *ex7_class;

void main(void)
{
    setup((t_messlist **)&ex7_class, (method)ex7_new, 0L, (short)sizeof(t_ex7),
    0L,0L);
    addmess((method)ex7_anything, "anything", A_GIMME, 0L);
}

void ex7_anything(t_ex7 *x, t_symbol *message, short argc, t_atom *argv)
{
    // utilize internal print object
    typedmess(x->a_print, message, argc, argv);
}

void *ex7_new(void)
{
    t_ex7 *x;
    t_atom atom;
    x = (t_ex7*)newobject(ex7_class);
    SETSYM(&atom, gensym("test"));
    x->a_print = newinstance(gensym("print"), 1, &atom);
}

```

```

    return x;
}

void ex7_free(t_ex7 *x)
{
    if(x->a_print)
        freeobject(x->a_print);
}

```

ex8.c

```

#include "ext.h"

// table access

typedef struct {
    t_object b_ob;
    Symbol *tablename;
    void* out;
} t_ex8;

void ex8_bang(t_ex8 *x);
void *ex8_new(t_symbol *s);
void *ex8_class;

void main(void)
{
    setup((t_messlist **)&ex8_class, (method)ex8_new, 0L, (short)sizeof(t_ex8),
    0L, A_SYM, 0L);
    addbang((method)ex8_bang);
}

void ex8_bang(t_ex8 *x)
{
    long dstSize;
    long **dstHandle;
    long i;
    table_get (x->tablename, &dstHandle, &dstSize);
    for(i = 0; i < dstSize; i++)
    {
        post("%d", *((*dstHandle)+i) );
    }
}

void *ex8_new(t_symbol *s)
{
    t_ex8 *x;
    x = (t_ex8*)newobject(ex8_class);
}

```

```

    x->tablename = s;
    return x;
}

```

ex9.c

```

#include "ext.h"

// table smoother

typedef struct {
    t_object b_ob;
    Symbol *tablename;
    void* out;
} t_ex9;

void ex9_bang(t_ex9 *x);
void *ex9_new(t_symbol *s);
void *ex9_class;

void main(void)
{
    setup((t_messlist **)&ex9_class, (method)ex9_new, 0L, (short)sizeof(t_ex9),
    0L, A_SYM, 0L);
    addbang((method)ex9_bang);
}

void ex9_bang(t_ex9 *x)
{
    long accum = 0;
    long dstSize, average, dif;
    long **dstHandle;
    long i;
    table_get (x->tablename, &dstHandle, &dstSize);
    for(i = 0; i < dstSize; i++)
    {
        accum += ((*dstHandle)+i);
    }
    average = accum / dstSize;
    // smoother
    for(i = 0; i < dstSize; i++)
    {
        dif = ((*dstHandle)+i) - average;
        dif = dif / 2;
        ((*dstHandle)+i) = average + dif;
    }
    table_dirty(x->tablename);
}

```

```
void *ex9_new(t_symbol *s)
{
    t_ex9 *x;
    x = (t_ex9*)newobject(ex9_class);
    x->tablename = s;
    return x;
}
```