CSE320 System Fundamentals II
Variables

YoungMin Kwon

SUNY Korea
The State University of New York

Contents

- Types
  - Basic, array, pointer, composite types
- Variable Scopes
  - auto, static, extern
  - local, global

SUNY Korea

Basic Types

- Basic types
  - char, short, int, long, long long, …
  - float, double, long double, …
- sizeof operator to get the size of a type in bytes

```
#include <stdio.h>
int main() {
    int i = 10;
    printf("sizeof(int): %ld\n", sizeof(int));
    printf("sizeof(i):   %ld\n", sizeof(i));
    printf("sizeof(10):  %ld\n", sizeof(10));
    return 0;
}
```

SUNY Korea

Basic Types & Literals

```
#include <stdio.h>
int main() {
    //see stdint.h for the size of primitive types
    char     c = 'a';
    short    s = (short)0;  //equivalent to short int
    int      i = 0;
    long     l = 0L;          //equivalent to long int
    long long ll = 0LL;     //equivalent to long long int
                            //(at least as large as long)
    float    f = 0.0f;
    double   d = 0.0;
    long double  ld = 0.0L;
...
    return 0;
}
```

SUNY Korea
The State University of New York

Basic Types & Literals

```
printf("       size of char: %2d, %2d\n", sizeof(c)
                                        , sizeof((char)'a'));
printf("      size of short: %2d, %2d\n", sizeof(s)
                                        , sizeof((short)0));
printf("        size of int: %2d, %2d\n", sizeof(i), sizeof(0));
printf("       size of long: %2d, %2d\n", sizeof(l), sizeof(0L));
printf("  size of long long: %2d, %2d\n", sizeof(ll), sizeof(0LL));
printf("      size of float: %2d, %2d\n", sizeof(f), sizeof(0.0f));
printf("     size of double: %2d, %2d\n", sizeof(d), sizeof(0.0));
printf("size of long double: %2d, %2d\n", sizeof(ld), sizeof(0.0L));
```

```
       size of char:  1,  1
      size of short:  2,  2
        size of int:  4,  4
       size of long:  8,  8
  size of long long:  8,  8
      size of float:  4,  4
     size of double:  8,  8
size of long double: 16, 16
```

SUNY Korea

Array Types

- Array Types
  - Array of other types. E.g. `int a[10];`
  - Array name is a `constant` pointing to the address of the first array element
  - sizeof(a) returns the total bytes allocated to the array

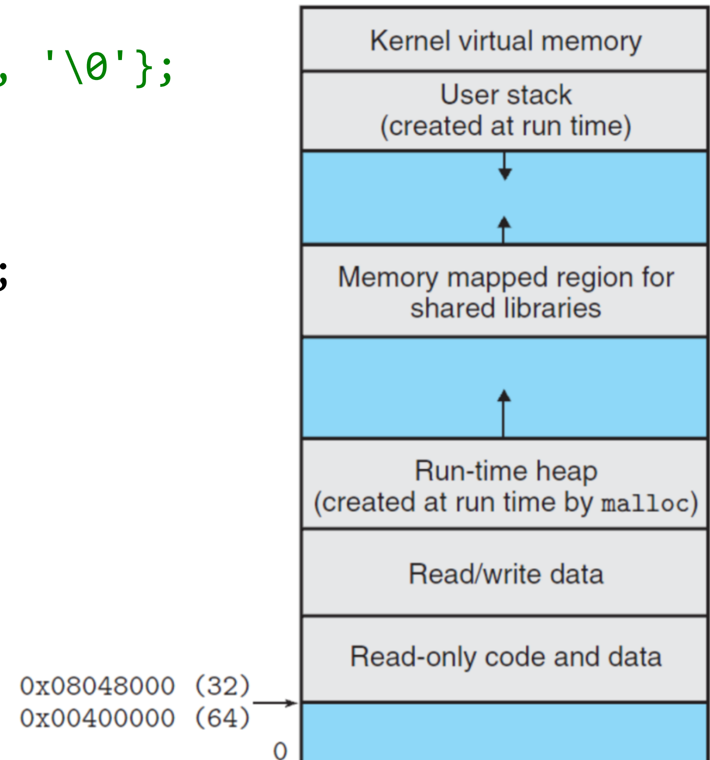
```
#include <stdio.h>
int main() {
    int a[10];
    printf("a:     %p\n", a);
    printf("&a[0]: %p\n", &a[0]);
    printf("sizeof(a):                %ld\n", sizeof(a));
    printf("sizeof(a)/sizeof(int): %ld\n", sizeof(a)/sizeof(int));
    return 0;
}
```

```
a:     0x7fffe5346530
&a[0]: 0x7fffe5346530
sizeof(a):             40
sizeof(a)/sizeof(int): 10
```

SUNY Korea

Array Types

```
printf("\"hello world\":     %p\n", "hello world");
printf("&\"hello world\"[0]: %p\n", &"hello world"[0]);
printf("sizeof(\"hello world\"): %ld\n", sizeof("hello world"));

char arr[] = "hello world";
//char arr[12] = {'h', 'e', ..., 'l', 'd', '\0'};
printf("arr:     %p\n", arr);
printf("&arr:    %p\n", &arr);
printf("&arr[0]: %p\n", &arr[0]);
printf("sizeof(arr): %ld\n", sizeof(arr));

"hello world":     0x55ffacda8088
&"hello world"[0]: 0x55ffacda8088
sizeof("hello world"): 12

arr:     0x7ffd67db146c
&arr:    0x7ffd67db146c
&arr[0]: 0x7ffd67db146c
sizeof(arr): 12
```



SUNY Korea

Pointer Types

- Pointer Types
  - Variables holding the address of other elements:

```
int *p, q; //*p: p is an int pointer type variable
p = &a;    //p has the address of a
q = *p;    //q has the value of a
```

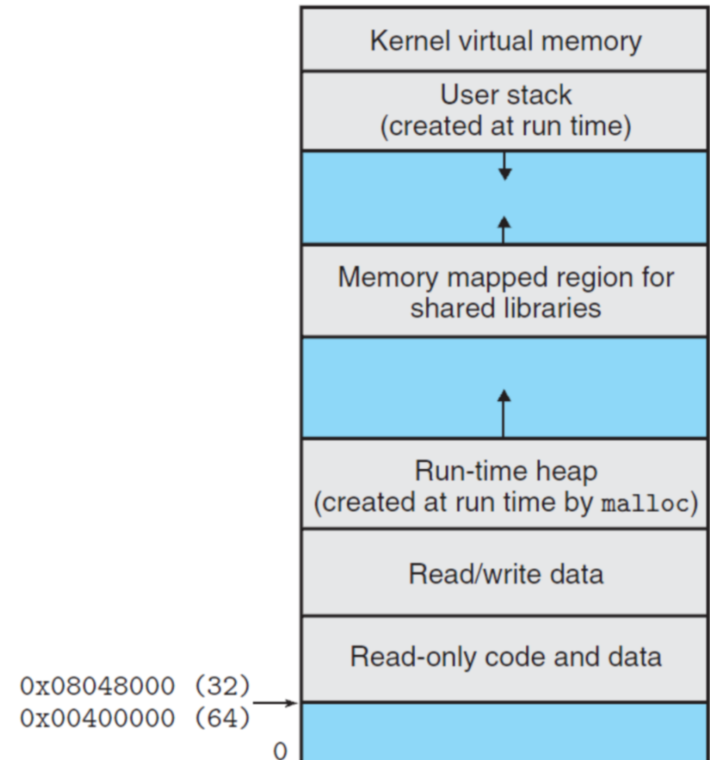
  - * at a variable definition: pointer definition
  - & operator returns the address of a variable
  - * operator returns the value stored at the address

SUNY Korea

Pointer Types

```
printf("\"hello world\":      %p\n", "hello world");
printf("&\"hello world\"[0]: %p\n", &"hello world"[0]);
printf("sizeof(\"hello world\"): %ld\n", sizeof("hello world"));

char *str = "hello world";
printf("str:     %p\n", str);
printf("&str:    %p\n", &str);
printf("&str[0]: %p\n", &str[0]);
printf("sizeof(str): %ld\n", sizeof(str));
```

```
"hello world":     0x55ffacda8088
&"hello world"[0]: 0x55ffacda8088
sizeof("hello world"): 12

str:     0x55ffacda8088
&str:    0x7ffd67db1438
&str[0]: 0x55ffacda8088
sizeof(str): 8
```



Pointer Arithmetic

- A type is associated with a pointer variable
- `+, -` operators on pointers add or subtract the size of the associated type
  - `p++, p--, p += 2, …`
  - `p – q` is the number of elements between `p` and `q`
  - `p + q, p * 2, p / 2,…` are invalid operations
- Casting
  - `(type) expr`
  - Changes the type of expr to `type`

SUNY Korea

Pointer Arithmetic

```c
#include <stdio.h>
void swap(int *a, int *b) {
    int t = *a;
    *a = *b;
    *b = t;
}

void getbuf(int** p, int n) {
    static int next = 0;
    static int buf[100] =
               { 0, 1, 2, 3, };
    // TODO: boundary check
    *p = buf + next;
    next += n;
}
```

```c
int main() {
    int *a, *b;
    getbuf(&a, 4);
    swap(a+0, a+3);
    swap(a+2, a+1);
    printf("a:[0..3]: %d, %d, %d, %d\n",
            a[0], a[1], a[2], a[3]);

    getbuf(&b, 4);
    printf("a + 1: %ld, %ld, %ld\n",
           (long)a,
           (long)(a + 1),
           ((long)a) + 1);
    printf("b - a: %ld, %ld\n",
           b - a,
           (long)b - (long)a);
}
```

```
a:[0..3]: 3, 2, 1, 0
a + 1: 93978374623264, 93978374623268, 93978374623265
b - a: 4, 16
```

SUNY Korea

Composite Types

- Composite Types

```
struct spair {char a; int b;};
union  upair {char a; int b;};
```

  - struct type
    - Each element has its own space
  - union type
    - All elements share the same space
  - . : to access the elements of a composite data
  - -> : to access the elements of a composite data from a pointer

```
struct spair s   = {.a = 1, .b = 2},
             *ps = &s;
ps->a = ps->b; //equiv. to (*ps).a = (*ps).b;
```

SUNY Korea

Composite Types

```c
int main() {
    struct spair {long a; long b;};
    union  upair {long a; long b;};

    struct spair s = {.a = 1, .b = 2}, *ps = &s;
    union  upair u = {.a = 1, .b = 2}, *pu = &u;

    printf("sizeof(s): %ld, &s.a: %p, &s.b: %p\n", sizeof(s), &s.a, &s.b);
    printf("sizeof(u): %ld, &u.a: %p, &u.b: %p\n", sizeof(u), &u.a, &u.b);

    ps->b = 3; //equiv. to (*ps).b = 3;
    pu->b = 3; //equiv. to (*pu).b = 3;
    printf("s.a: %ld, s.b: %ld\n", s.a, s.b);
    printf("u.a: %ld, u.b: %ld\n", u.a, u.b);

    return 0;
}
```

```
sizeof(s): 16, &s.a: 0x7ffd2a88f000, &s.b: 0x7ffd2a88f008
sizeof(u): 8,  &u.a: 0x7ffd2a88efe8, &u.b: 0x7ffd2a88efe8
s.a: 1, s.b: 3
u.a: 3, u.b: 3
```

SUNY Korea

Composite Types

- Byte order

```c
//byteorder.c
#include <stdio.h>

union byte_order {
    unsigned long i;
    unsigned char a[8];
};

int main() {
    union byte_order u;

    u.i = 0x0123456789abcdef;

…
    printf("%lx\n", u.i);
    printf("%02x, ", u.a[0]);
    printf("%02x, ", u.a[1]);
    printf("%02x, ", u.a[2]);
    printf("%02x, ", u.a[3]);
    printf("%02x, ", u.a[4]);
    printf("%02x, ", u.a[5]);
    printf("%02x, ", u.a[6]);
    printf("%02x\n", u.a[7]);
    return 0;
}
/* output:
123456789abcdef
ef, cd, ab, 89, 67, 45, 23, 01
*/
```

SUNY Korea

Typedef

- Users can define their own types using typedef
- Syntax is like variable definitions except that they begin with typedef keyword

```
typedef char* string;
typedef int myint;
typedef float vector[3];
typedef struct person {
    char name[10];
    long id;
} person;
typedef int (*binop)(int, int); // function pointer
```

SUNY Korea

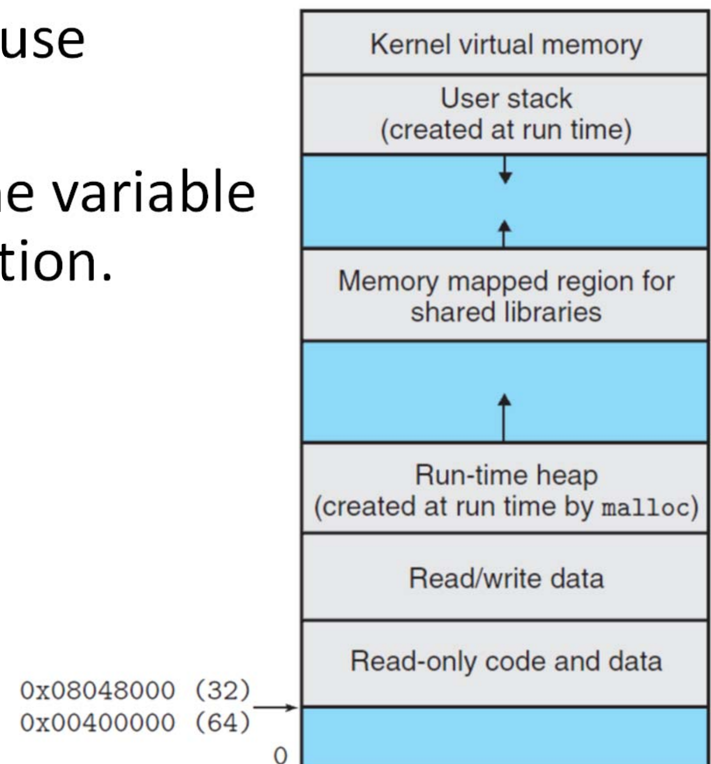
```c
#include <stdio.h>
typedef struct vector3 vector3;
struct vector3 {
   double x, y, z;
   void (*add)(vector3* self, vector3* a);
   void (*sub)(vector3* self, vector3* a);
   void (*print)(vector3* self);
};

void add(vector3* self, vector3* a) {
    self->x += a->x;
    self->y += a->y;
    self->z += a->z;
}

void sub(vector3* self, vector3* a) {
    self->x -= a->x;
    self->y -= a->y;
    self->z -= a->z;
}

void print(vector3* self) {
    printf("[%lf, %lf, %lf]\n",
        self->x, self->y, self->z);
}
```

```c
void init(vector3* self,
        double x, double y, double z)
{
    self->x = x;
    self->y = y;
    self->z = z;
    self->add = add;
    self->sub = sub;
    self->print = print;
}

int main() {
    vector3 a, b;

    init(&a, 1, 2, 3);
    init(&b, 2, 3, 4);
    a.add(&a, &b);
    a.print(&a);
    return 0
}
```

SUNY Korea

Variable Scope

- Variable Declaration vs Definition
  - Declaration: tell the compiler about the variable/function you are going to use without actually allocating space.
  - Definition: allocating a space for the variable and the machine code for the function.
- Variable Scope
  - Where the variable is visible
  - Auto, Static, Extern scopes
- Variable Lifetime
  - Global, Local



SUNY Korea

Variable Scope

- Auto Variables
  - Visible from the containing block
  - Valid while the program remains within the block (local)
  - Allocated in stack area
  - Initialized every time the program enters the block

```
int auto_var(int p) {
    int i = 0;
    while(i < 10) {
        int j = 0;
        while(j < 10) {

            {
                int k = 0;
                k++;
            }

            if(i < j) {
                int k = 0;
                k++;
            }

            j++;
        }
        i++;
    }
}
```

SUNY Korea

Variable Scope

- Static Variables
  - Visible from the containing block or from the same file when defined outside of a function.
  - Valid through out the lifetime of the program (global)
  - Allocated in read only or read/write data area
  - Initialized only once during the lifetime of the program

```
static int s = 0;

int static_var() {
    int i = 0;
    while(i < 10) {
        int j = 0;
        while(j < 10) {

            {
                static int k = 0;
                k++;
            }

            if(i < j) {
                static int k = 0;
                k++;
                s++;
            }

            j++;
        }
        i++;
    }
}
```

SUNY Korea

Variable Scope

- Extern Variables
  - Visible form other files
  - Valid through out the lifetime of the program (global)
  - Allocated in read only or read/write data area
  - Initialized only once during the lifetime of the program

```
int e = 0;     //define e
extern int f; //declare f

int extern_var() {
    int i;
    while(i < 10) {
        int j;
        while(j < 10) {

            {
                extern int k; //declare k
                k++;
            }

            if(i < j) {
                extern int k; //declare k
                k++;
            }

            j++;
        }
        i++;
    }
}
```

SUNY Korea

Variable Allocation

```c
//var_loc.c
//gcc -c var_loc.c

#include <stdio.h>

extern int ext;
int g_ini = 3;
int g_uni;
static int s_ini = 4;
static int s_uni;

int main() {
    int a_ini = 5;
    int a_uni;
    printf("ext: %d\n", ext);
    printf("g_ini: %d, g_uni: %d\n", g_ini, g_uni);
    printf("s_ini: %d, s_uni: %d\n", s_ini, s_uni);
    printf("a_ini: %d, a_uni: %d\n", a_ini, a_uni);
}
```

SUNY Korea

Variable Allocation

```
$ readelf -s var_loc.o

Symbol table '.symtab' contains 18 entries:
   Num:    Value          Size Type    Bind   Vis      Ndx Name
     0: 0000000000000000     0 NOTYPE  LOCAL  DEFAULT  UND
     1: 0000000000000000     0 FILE    LOCAL  DEFAULT  ABS var_loc.c
…
     5: 0000000000000004     4 OBJECT  LOCAL  DEFAULT    3 s_ini
     6: 0000000000000000     4 OBJECT  LOCAL  DEFAULT    4 s_uni
…
    12: 0000000000000000     4 OBJECT  GLOBAL DEFAULT    3 g_ini
    13: 0000000000000004     4 OBJECT  GLOBAL DEFAULT  COM g_uni
    14: 0000000000000000   138 FUNC    GLOBAL DEFAULT    1 main
    15: 0000000000000000     0 NOTYPE  GLOBAL DEFAULT  UND ext
    16: 0000000000000000     0 NOTYPE  GLOBAL DEFAULT  UND _GLOBAL_OFFSET_TABLE_
    17: 0000000000000000     0 NOTYPE  GLOBAL DEFAULT  UND printf
```

Local to the file

Visible from all files

SUNY Korea

Variable Allocation

```c
//--file1.c-----------------------------------
int e1 = 100;
static int s1 = 200;

int main() {
    extern int e2;
    extern void foo(int a);

    foo(e1 + e2 + s1);
    return 0;
}

//--file2.c-----------------------------------
#include <stdio.h>
int e2;

void foo(int a) {
    extern int e1;
    static int s2;
    int b = 300;

    printf("e1: %d, e2: %d, s2: %d, a: %d, b: %d\n",
        e1, e2, s2, a, b);
}
```

SUNY Korea

Variable Allocation

```
$ readelf -s file1.o

Symbol table '.symtab' contains 15 entries:
   Num:    Value          Size Type    Bind   Vis      Ndx Name
     0: 0000000000000000     0 NOTYPE  LOCAL  DEFAULT  UND
     1: 0000000000000000     0 FILE    LOCAL  DEFAULT  ABS file1.c
     2: 0000000000000000     0 SECTION LOCAL  DEFAULT    1
     3: 0000000000000000     0 SECTION LOCAL  DEFAULT    3
     4: 0000000000000000     0 SECTION LOCAL  DEFAULT    4
     5: 0000000000000004     4 OBJECT  LOCAL  DEFAULT    3 s1
     6: 0000000000000000     0 SECTION LOCAL  DEFAULT    6
     7: 0000000000000000     0 SECTION LOCAL  DEFAULT    7
     8: 0000000000000000     0 SECTION LOCAL  DEFAULT    8
     9: 0000000000000000     0 SECTION LOCAL  DEFAULT    5
    10: 0000000000000000     4 OBJECT  GLOBAL DEFAULT    3 e1
    11: 0000000000000000    44 FUNC    GLOBAL DEFAULT    1 main
    12: 0000000000000000     0 NOTYPE  GLOBAL DEFAULT  UND e2
    13: 0000000000000000     0 NOTYPE  GLOBAL DEFAULT  UND _GLOBAL_OFFSET_TABLE_
    14: 0000000000000000     0 NOTYPE  GLOBAL DEFAULT  UND foo
```

Not in this file

SUNY Korea

Variable Allocation

```
$ readelf -s file2.o

Symbol table '.symtab' contains 16 entries:
   Num:    Value          Size Type    Bind   Vis      Ndx Name
     0: 0000000000000000     0 NOTYPE  LOCAL  DEFAULT  UND
     1: 0000000000000000     0 FILE    LOCAL  DEFAULT  ABS file2.c
     2: 0000000000000000     0 SECTION LOCAL  DEFAULT    1
     3: 0000000000000000     0 SECTION LOCAL  DEFAULT    3
     4: 0000000000000000     0 SECTION LOCAL  DEFAULT    4
     5: 0000000000000000     0 SECTION LOCAL  DEFAULT    5
     6: 0000000000000000     4 OBJECT  LOCAL  DEFAULT    4 s2.2318
     7: 0000000000000000     0 SECTION LOCAL  DEFAULT    7
     8: 0000000000000000     0 SECTION LOCAL  DEFAULT    8
     9: 0000000000000000     0 SECTION LOCAL  DEFAULT    9
    10: 0000000000000000     0 SECTION LOCAL  DEFAULT    6
    11: 0000000000000004     4 OBJECT  GLOBAL DEFAULT  COM e2
    12: 0000000000000000    74 FUNC    GLOBAL DEFAULT    1 foo
    13: 0000000000000000     0 NOTYPE  GLOBAL DEFAULT  UND e1
    14: 0000000000000000     0 NOTYPE  GLOBAL DEFAULT  UND _GLOBAL_OFFSET_TABLE_
    15: 0000000000000000     0 NOTYPE  GLOBAL DEFAULT  UND printf
```

e2 is in this file

SUNY Korea

Variable Allocation

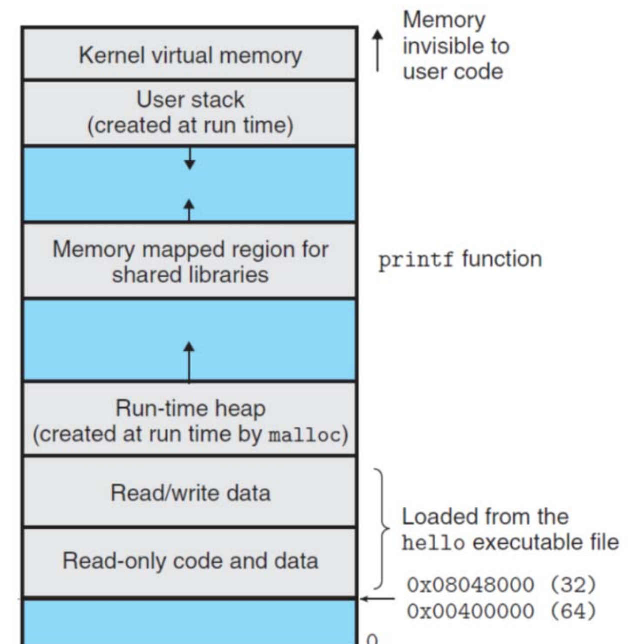
```
$ readelf -s a.out

Symbol table '.symtab' contains 71 entries:
   Num:    Value          Size Type    Bind   Vis      Ndx Name
...
    36: 0000000000000000     0 FILE    LOCAL  DEFAULT  ABS file1.c
    37: 0000000000004014     4 OBJECT  LOCAL  DEFAULT   25 s1
    38: 0000000000000000     0 FILE    LOCAL  DEFAULT  ABS file2.c
    39: 000000000000401c     4 OBJECT  LOCAL  DEFAULT   26 s2.2318
...
    54: 0000000000000000     0 FUNC    GLOBAL DEFAULT  UND printf@@GLIBC_2.2.5
    55: 0000000000004020     4 OBJECT  GLOBAL DEFAULT   26 e2
...
    62: 0000000000001175    74 FUNC    GLOBAL DEFAULT   16 foo
...
    66: 0000000000001149    44 FUNC    GLOBAL DEFAULT   16 main
...
    68: 0000000000004010     4 OBJECT  GLOBAL DEFAULT   25 e1
...
```

Load at run time

SUNY Korea

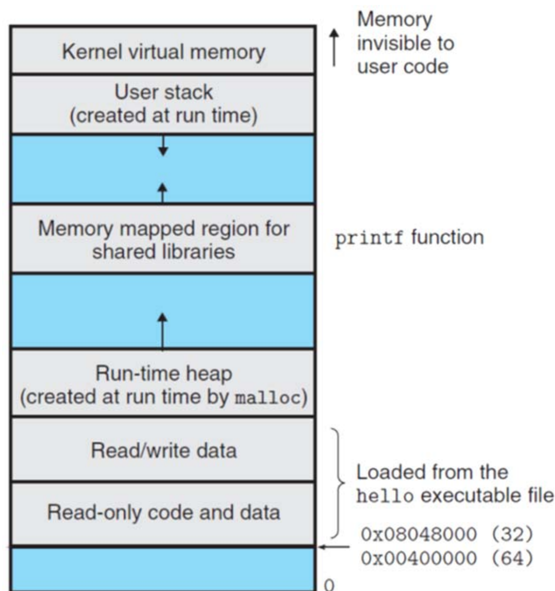
Variable Allocation in Assembler

- .data
  - Readable, writable, initialized, global variables
- .rodata
  - Readonly, initialized, global variables
- .comm
  - Uninitialized global variables
- .text
  - Instruction codes
- Stack
  - Local variables, parameters
- Heap
  - Dynamic allocation (malloc)
- Shared libraries
  - Library functions like printf



SUNY Korea

```c
int ei = 100;
static int si = 200;

int eu;
static int su;

const int ec = 300;
static const int sc = 400;

void foo(int p1, int p2) {
    extern void bar(char*);
    int ai = 500;
    int au;
    bar("Hello world");
}
```




```
        .globl   ei
        .data
ei:
        .long    100
si:
        .long    200
        .comm    eu,4,4   # to bss
        .local   su
        .comm    su,4,4   # equiv to .lcomm su
        .globl   ec
        .section         .rodata
ec:
        .long    300
sc:
        .long    400
.LC0:
        .string "Hello world"
        .text
        .globl   foo
foo:
        pushq    %rbp       # caller's stack frame
        movq     %rsp, %rbp   # foo's stack frame
        subq     $32, %rsp    # alloc mem in stack
        movl     %edi, -20(%rbp) # p1, p1
        movl     %esi, -24(%rbp) # p2, p2
        movl     $500, -4(%rbp)  # int ai = 500;
…
        ret
```

SUNY Korea

Practice Problems

- Write a function "`int length(char* str)`" that returns the length of the string `str`.

- Write a function "`void reverse(char* str)`" that takes a string and reverses it.
  - E.g. if `str="string"`, after `reverse(str), str` becomes `"gnirts"`.

- Write a function "`void sort(char* str)`" that takes a string and sorts the characters in `str`.
  - E.g. if `str="asdf"`, after `sort(str), str` becomes `"adfs"`.

SUNY Korea

Programming Assignment 2

- Finish implementing the programs in the following slides
  - Circularly doubly linked list
  - String reverse function
- Use Blackboard to submit the assignment.
- Due date: 9/15/2022

```
A sample output:
$ ./a.out
enter a word: string
reverse: gnirts
```

SUNY Korea

```c
// linkedlist.c

#include <stdio.h>
#include <stdlib.h>

struct List {
    struct List *prev, *next;
    char data;
};

void init_head(struct List *head) {
    head->next = head->prev = head;
}

int is_empty(struct List *head) {
    return head->next == head;
}
```

SUNY Korea
The State University of New York

```c
void add_to_last(struct List *head, struct List *list) {
    list->next = head;
    list->prev =       //TODO
    head->prev->next = //TODO
    head->prev =       //TODO
}

void add_to_first(struct List *head, struct List *list) {
    //TODO
}

struct List* remove_last(struct List *head) {
    struct List *list = head->prev;
    head->prev->prev->next = //TODO
    head->prev =             //TODO
    list->next = list->prev = NULL;
    return list;
}

struct List* remove_first(struct List *head) {
    //TODO
}
```

SUNY Korea

```c
struct List free_list[500];
struct List free_head;

void init_free_list() {
    int i;
    init_head(&free_head);
    for(i = 0; i < 500; i++)          //link the 500 free lists
        add_to_last(&free_head, free_list + i);
}

struct List *new_list() {             //get a new List struct
    if(is_empty(&free_head)) {
        printf("out of list\n");
        exit(0);
    }
    return remove_first(&free_head); //queue
}

void del_list(struct List *list) {   //free a List struct
    add_to_last(&free_head, list);   //queue
}
```

SUNY Korea

```c
char *reverse(char *str) {
    struct List stack;
    int i;

    init_head(&stack);
    for(i = 0; str[i]; i++) {
        struct List *list = new_list();
        //TODO: update list's data with str[i] and push list to stack
    }

    for(i = 0; !is_empty(&stack); i++) {
        struct List *list;
        //TODO: pop list from stack and update str[i]
        del_list(list);
    }
    return str;
}
int main() {
    char str[100];
    init_free_list();
    printf("enter a word: ");
    scanf("%99s", str);
    printf("reverse: %s\n", reverse(str));
}
```

SUNY Korea