Yale-NUS College - 2018-2019

Bruno Bodin

YSC2227: INTRO TO C

week 02.1.structures (auto-generated)

STRUCT

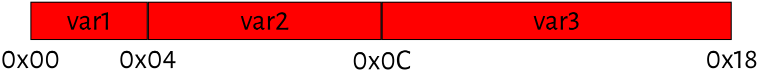
- You can define new types as a set of elements

```
struct stamp {
  int  var1;
  char var2[8];
  char var3[12];
};
```

- Then you can use this as a new type :

```
struct stamp A;
A.var1 = 0;
A.var2[5]  = ’a’;
(&A)->var2[5]  = ’a’;
```

- In memory data will be contiguous[1] :



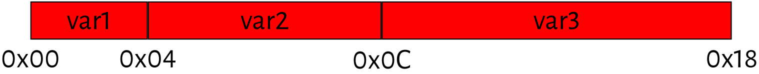
var1 var2 var3

0x00 0x04 0x0C 0x18

[1] each element is aligned to `sizeof(element)`

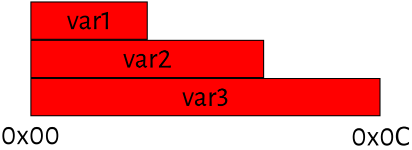
UNION

- Syntax is similar to struct, but the semantics is different!
- With struct elements are ordered in memory:


var1 var2 var3

0x00 0x04 0x0C 0x18

- with union they overlap:


var1

var2

var3

0x00 0x0C

TYPEDEF

- A different tool to define new types but only using renaming.
  `typedef <PREVIOUS_TYPE> <NEW_TYPE>;`
- For example : `typedef int entier;`
- Much more powerful than `#define` when it comes to type definition.
- Some examples that does not work with the `#define` macro.

```
typedef int *int_ptr;
typedef void (*fun_ptr)(int);
int_ptr p1, p2;
void fun (fun_ptr f);
```

- **What would happen with `#define` ?**

STORY TYPEDEF+STRUCT

- One of the biggest and most confusing missunderstanding in C online tutorials.

```
typedef struct {
  int x;
  int y;
} point;
```

- That is:

```
typedef struct something {
  int x;
  int y;
} point;
```

- And more precisely :

```
struct something {
  int x;
  int y;
};
typedef struct something point;
```

STRUCT AND UNION DEMO

- Create one of each
- Set a value to its first member
- Print the value of this member
- Set a value to its second member
- Print the value of the second member
- Print the value of the first member
- Same with pointers

What interesting things could we do with union ?

EXAMPLE WITH BIT-FIELD

```
union bit_char {
    struct {
        unsigned int bit0: 1;
        unsigned int bit1: 1;
        unsigned int bit2: 1;
        unsigned int bit3: 1;
        unsigned int bit4: 1;
        unsigned int bit5: 1;
        unsigned int bit6: 1;
        unsigned int bit7: 1;
    } bits;
    char character;
};
```

MEMORY MANAGEMENT

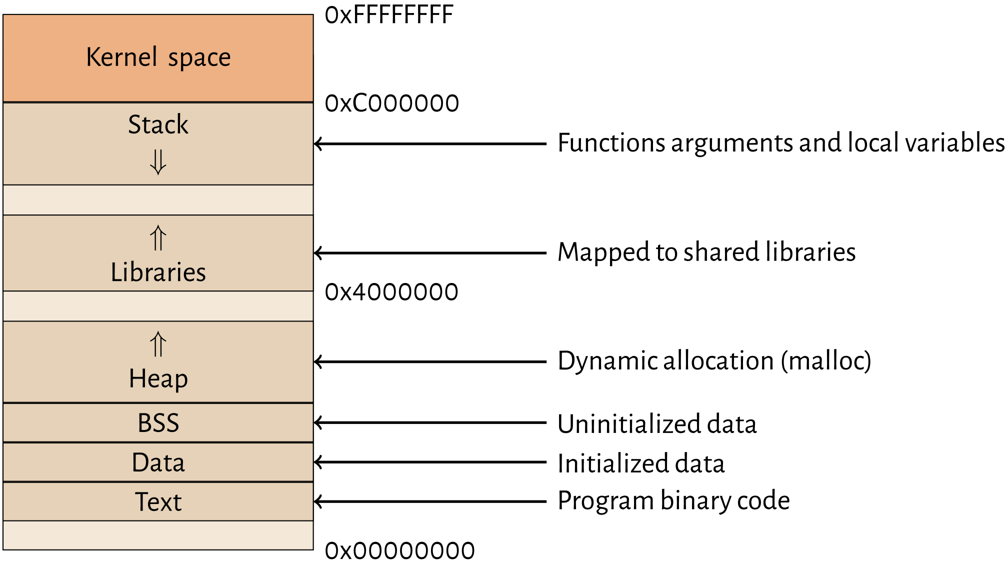
- You can allocate memory space using `malloc` that returns a pointer to a new memory location :

```
int* numbers = malloc ( 12 * sizeof (int));
for (int i = 0; i < 12 ; numbers[i++] = i);
for (int i = 0; i < 12 ; i++) printf("%d\n",i);
```

- Be careful this memory space is used until you `free` it or terminate your program.

```
free ( numbers );
```

LINUX MEMORY LAYOUT



0xFFFFFFFF

Kernel space

0xC000000

Stack ⇓ — Functions arguments and local variables

⇑ Libraries — Mapped to shared libraries

0x4000000

⇑ Heap — Dynamic allocation (malloc)

BSS — Uninitialized data

Data — Initialized data

Text — Program binary code

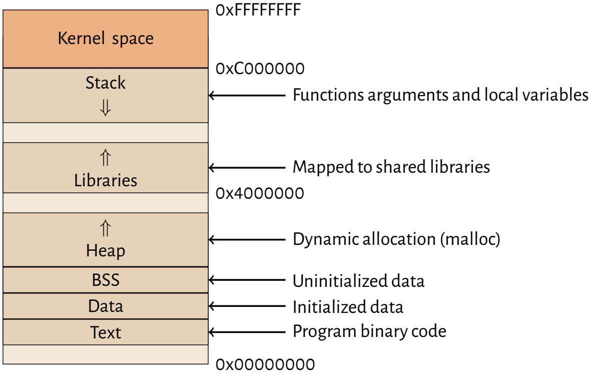
0x00000000

EXERCICE

```
#include  <stdio.h>

int A = 0;
int B;
int C = 0;
int D;
int main(int argc, char ** argv)
{
  int *E = malloc(sizeof(int)*2);
  return 0;
}
```

Where are stored A,B,C,D,E, and main?



EXERCICES

- `int permute (int num)` : permutes the digit of `num`
- `size_t strlen (const char* str)` : returns the length of `s`
- `char* strcpy (char* d, const char * s)` : copy `s` to `d`, returns `d`
- `char* strcat (char* d, const char * s)` : append a copy of `s` to `d`, returns `d`
- `int atoi(const char *s)` : Interprets an integer in a string pointed to by `s`.

TOPICS COVERED

- Variables and Assignement Operators ✓ (T. Bailey, Chapter 1 and 2)
- Numeric Data Types and Conversion ✓ (T. Bailey, Chapter 2)
- Arrays ✓ (T. Bailey, Chapter 8)
- Arithhmetic and bitwise operators ✓ (T. Bailey, Chapter 2 and 12)
- Compilation, flags, and command-line arguments ✓ (D. Harris C.10)
- Pointers ✓ (T. Bailey, Chapter 7)
- C functions ✓ (T. Bailey, Chapter 4)
- Files and I/O
- Control structures, logic operators, and loops ✓ (T. Bailey, Chapter 3)
- Scope ✓ (T. Bailey, Chapter 5)
- Structures and Unions ✓ (T. Bailey, Chapter 11 and 14)
- Memory management and segmentation ✓ (T. Bailey, Chapter 9)
- Basic libraries
- Makefile
- Debugging

KEY POINTS

REFERENCES

- cook and magician: *https://pixabay.com/en/users/graphicmama-team-2641041/*