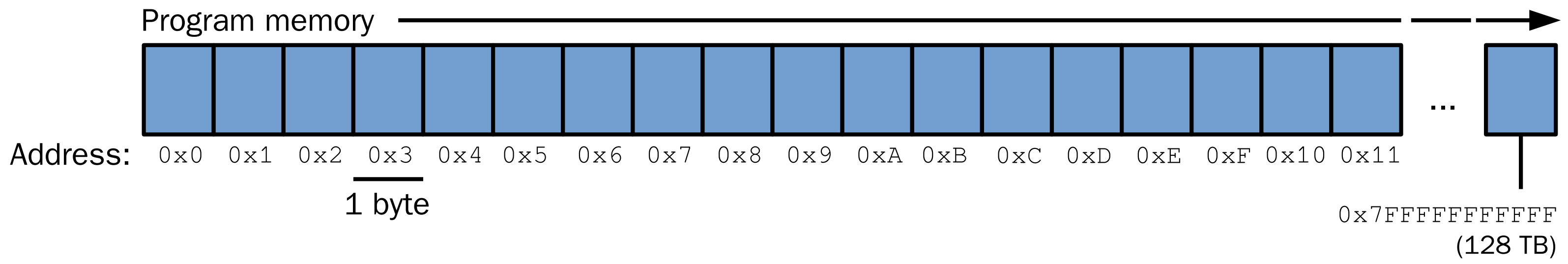
COMP26020 Programming Languages and Paradigms Part 1: C Programming

Introduction to Pointers

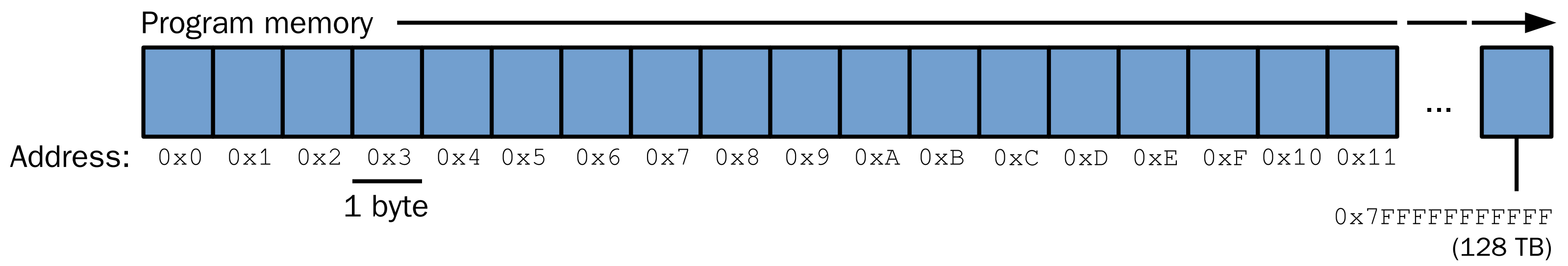
Program Memory Layout

- All the program's code and data is present somewhere in memory
- The area of memory accessible by the program is the **address space**
    - It's a large array of contiguous bytes
    - **Addresses** are indexes in that array



Program Memory Layout

- All the program's code and data is present somewhere in memory
- The area of memory accessible by the program is the **address space**
  - It's a large array of contiguous bytes
  - **Addresses** are indexes in that array



- Note that this is <u>virtual</u> memory:
  - Address space size independent from the amount of RAM
  - Each program gets its own address space

Program Memory Layout

- **Address of a variable**: the address in memory of the first byte storing that variable

Program Memory Layout

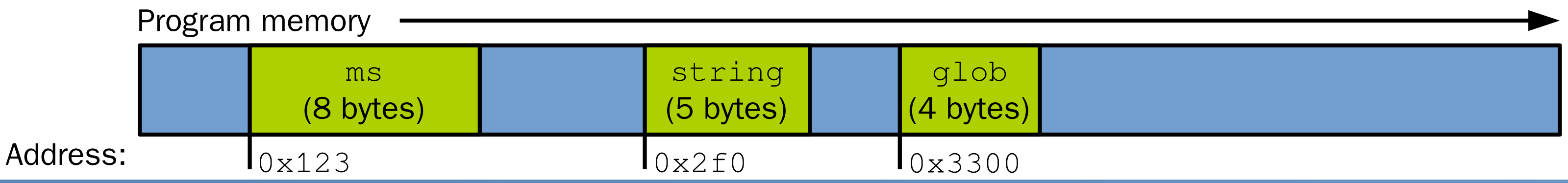
- **Address of a variable**: the address in memory of the first byte storing that variable

```
int glob = 12;
char string[] = "abcd";

typedef struct {
    int member1;
    float member2;
} mystruct;

int main(int argc, char **argv) {
    mystruct ms;
    ms.member1 = 42;
    ms.member2 = 4.2;
    printf("ms member1: %d, member2: %f\n", ms.member1, ms.member2);
    return 0;
}
```

Program Memory Layout

- **Address of a variable**: the address in memory of the first byte storing that variable

```c
int glob = 12;               // glob's address is 0x3300
char string[] = "abcd";      // string's address is 0x2f0

typedef struct {
    int member1;
    float member2;
} mystruct;

int main(int argc, char **argv) {
    mystruct ms;              // ms' address is 0x123
    ms.member1 = 42;
    ms.member2 = 4.2;
    printf("ms member1: %d, member2: %f\n", ms.member1, ms.member2);
    return 0;
}
```



Addresses

- Use the `&` operator to get the address of a variable

```c
int glob = 12;
char string[] = "abcd";

typedef struct {
    int member1;
    float member2;
} mystruct;

int main(int argc, char **argv) {
    mystruct ms = {1, 2.0};

    // With modern processors an address is a 64 bits value so we need the right format
    // specifier: "%p", which will print the address in hexadecimal prefixed by "0x",
    // for example "0x12345"
    printf("ms address: %p, glob address: %p, string address: %p\n", &ms, &glob, &string);
    return 0;
}
```

09-pointers-introduction/ampersand.c

Pointers

***Pointer*: variable that contains an address** (possibly of another variable)

Pointers

***Pointer*: variable that contains an address** (possibly of another variable)

- Declaration: `<pointed type> *<pointer name>;`

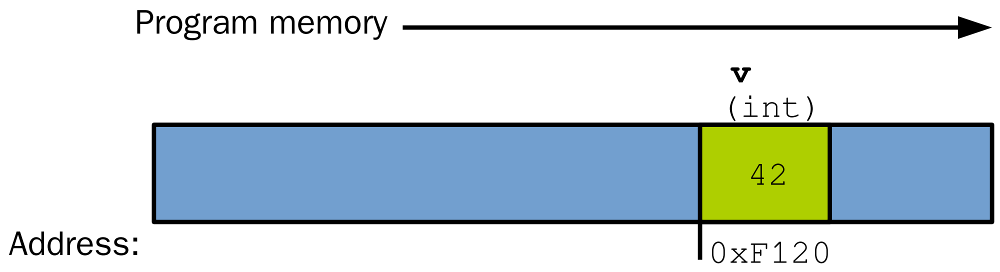
```c
int v = 42;
int *ptr = &v;
```

Pointers

***Pointer*: variable that contains an address** (possibly of another variable)

- Declaration: `<pointed type> *<pointer name>;`

```c
int v = 42;
int *ptr = &v;
```

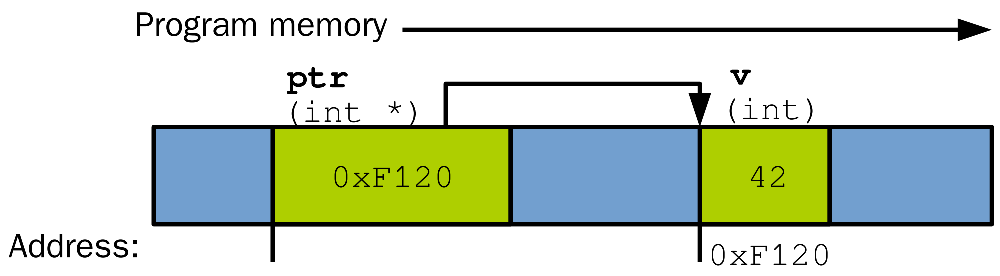


Pointers

***Pointer*: variable that contains an address** (possibly of another variable)

- Declaration: `<pointed type> *<pointer name>;`

```
int v = 42;
int *ptr = &v;
```

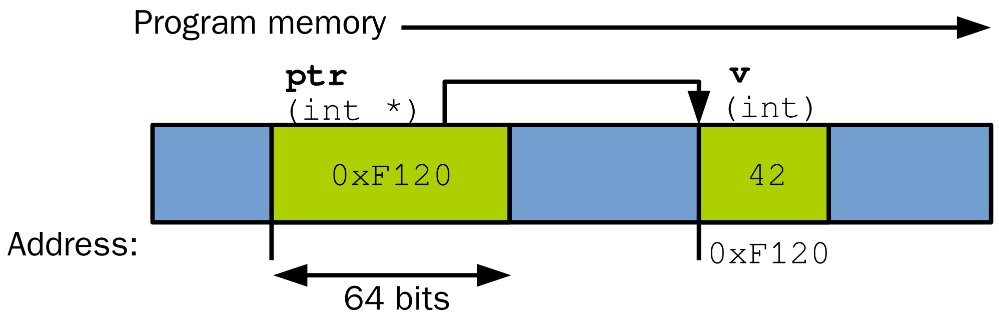


Pointers

***Pointer*: variable that contains an address** (possibly of another variable)

- Declaration: `<pointed type> *<pointer name>;`

```c
int v = 42;
int *ptr = &v;
```

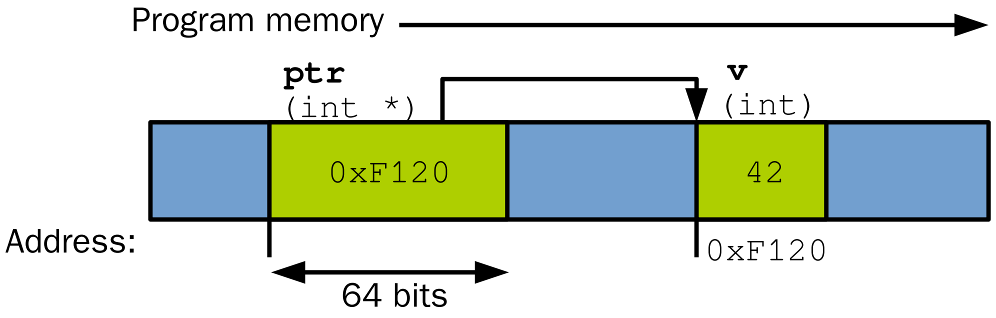


Pointers

***Pointer*: variable that contains an address** (possibly of another variable)

- Declaration: `<pointed type> *<pointer name>;`

```c
int v = 42;
int *ptr = &v;
```



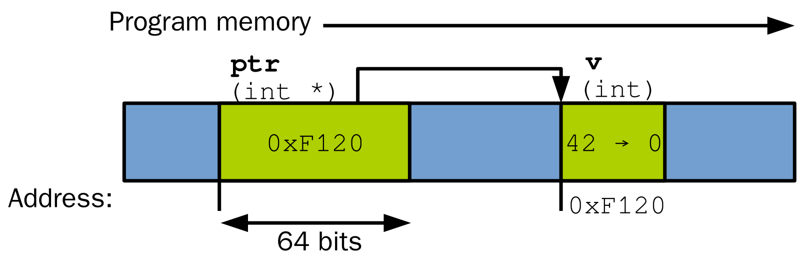
- We say that `ptr` ***points to*** `v`

Pointers

- **Through a pointer it is possible to access the variable it points**
- This is called **dereferencing**
- It is achieved with the operator `*`

```c
int v = 42;
int *ptr = &v;

printf("value pointed by ptr:   %d\n", *ptr);     // 42
printf("v's value:              %d\n", v);        // 42

*ptr = 0;

printf("value pointed by ptr:   %d\n", *ptr);     // 0
printf("v's value:              %d\n", v);        // 0
```

09-pointers-introduction/pointer-update.c



Pointers

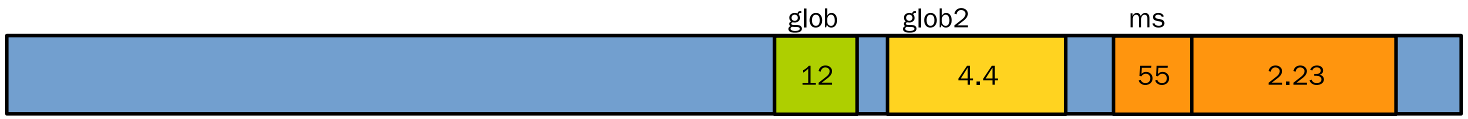
```c
int glob = 12;
double glob2 = 4.4;

typedef struct { int member1; double member2; } mystruct;

int main(int argc, char **argv) {
    mystruct ms = {55, 2.23};

    int *ptr1 = &glob;
    double *ptr2 = &glob2;
    mystruct *ptr3 = &ms;

    /* Print each pointer's value (pointed address), and pointed value (pointed variable) */
    printf("ptr1 = %p, *ptr1 = %d\n", ptr1, *ptr1);
    printf("ptr2 = %p, *ptr2 = %f\n", ptr2, *ptr2);
    printf("ptr3 = %p, *(ptr3).member1 = %d, *(ptr3).member2 = %d\n",
            ptr3, *(ptr3).member1, *(ptr3).member2);
}
```

09-pointers-introduction/pointer.c

Pointers

```c
int glob = 12;
double glob2 = 4.4;

typedef struct { int member1; double member2; } mystruct;

int main(int argc, char **argv) {
    mystruct ms = {55, 2.23};

    int *ptr1 = &glob;
    double *ptr2 = &glob2;
    mystruct *ptr3 = &ms;

    /* Print each pointer's value (pointed address), and pointed value (pointed variable) */
    printf("ptr1 = %p, *ptr1 = %d\n", ptr1, *ptr1);
    printf("ptr2 = %p, *ptr2 = %f\n", ptr2, *ptr2);
    printf("ptr3 = %p, *(ptr3).member1 = %d, *(ptr3).member2 = %d\n",
            ptr3, *(ptr3).member1, *(ptr3).member2);
}
```

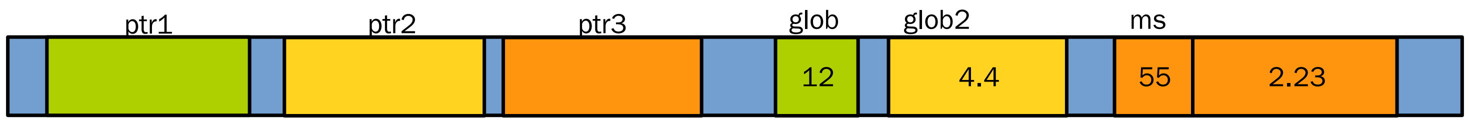
09-pointers-introduction/pointer.c



Pointers

```c
int glob = 12;
double glob2 = 4.4;

typedef struct { int member1; double member2; } mystruct;

int main(int argc, char **argv) {
    mystruct ms = {55, 2.23};

    int *ptr1 = &glob;
    double *ptr2 = &glob2;
    mystruct *ptr3 = &ms;

    /* Print each pointer's value (pointed address), and pointed value (pointed variable) */
    printf("ptr1 = %p, *ptr1 = %d\n", ptr1, *ptr1);
    printf("ptr2 = %p, *ptr2 = %f\n", ptr2, *ptr2);
    printf("ptr3 = %p, *(ptr3).member1 = %d, *(ptr3).member2 = %d\n",
            ptr3, *(ptr3).member1, *(ptr3).member2);
}
```

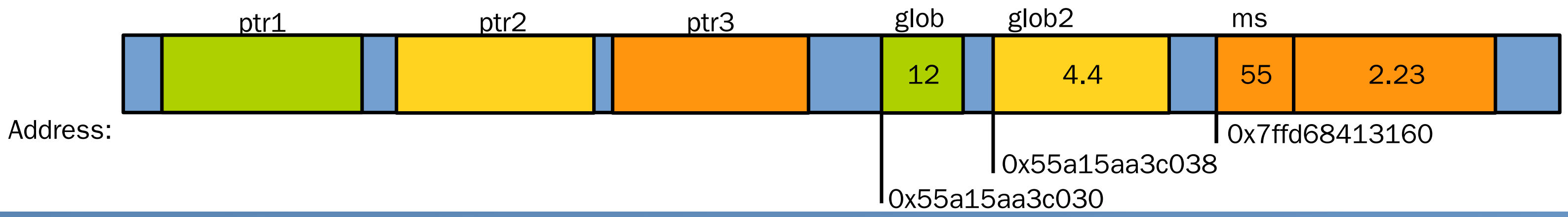
09-pointers-introduction/pointer.c



Pointers

```c
int glob = 12;
double glob2 = 4.4;

typedef struct { int member1; double member2; } mystruct;

int main(int argc, char **argv) {
    mystruct ms = {55, 2.23};

    int *ptr1 = &glob;
    double *ptr2 = &glob2;
    mystruct *ptr3 = &ms;

    /* Print each pointer's value (pointed address), and pointed value (pointed variable) */
    printf("ptr1 = %p, *ptr1 = %d\n", ptr1, *ptr1);
    printf("ptr2 = %p, *ptr2 = %f\n", ptr2, *ptr2);
    printf("ptr3 = %p, *(ptr3).member1 = %d, *(ptr3).member2 = %d\n",
            ptr3, *(ptr3).member1, *(ptr3).member2);
}
```

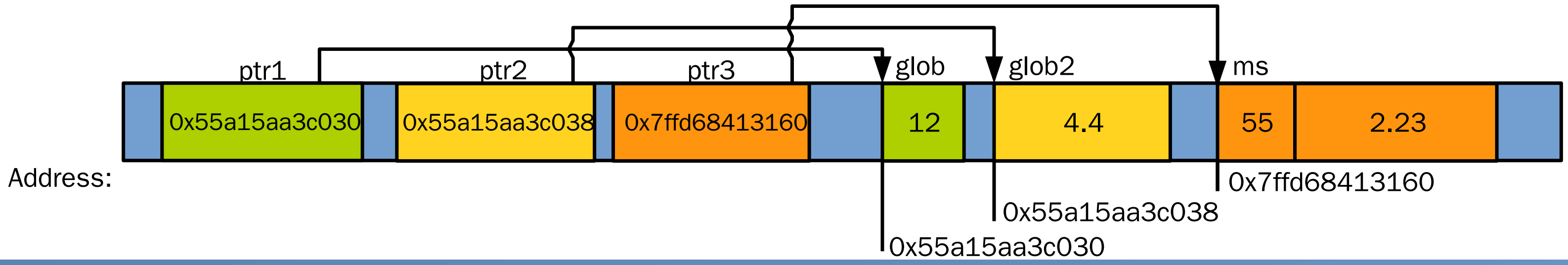
09-pointers-introduction/pointer.c



Pointers

```c
int glob = 12;
double glob2 = 4.4;

typedef struct { int member1; double member2; } mystruct;

int main(int argc, char **argv) {
    mystruct ms = {55, 2.23};

    int *ptr1 = &glob;
    double *ptr2 = &glob2;
    mystruct *ptr3 = &ms;

    /* Print each pointer's value (pointed address), and pointed value (pointed variable) */
    printf("ptr1 = %p, *ptr1 = %d\n", ptr1, *ptr1);
    printf("ptr2 = %p, *ptr2 = %f\n", ptr2, *ptr2);
    printf("ptr3 = %p, *(ptr3).member1 = %d, *(ptr3).member2 = %d\n",
            ptr3, *(ptr3).member1, *(ptr3).member2);
}
```

09-pointers-introduction/pointer.c



Summary

- A pointer: variable that stores an address corresponding to a memory location
- Can access that location through the pointer

Feedback form: https://bit.ly/3fKzIzr