Void Pointer

Sujoy Sinha Roy

School of Computer Science

University of Birmingham

Void pointer

- A void pointer in C has no associated data type.
- It can store the address of any type of object
- ‘*Generic pointer*’
- It can be type-casted to any types.

Syntax for declaration

```
void *pointer_name;
```

Void pointer and reusability

- Most important feature of the void pointer is reusability.
- We can store the address of **any** object
- Whenever required we can typecast it to a required type

Void pointer example

```
int main()
{
   void *pv;
   int iData = 5;
   char cData = 'C';

   //Pointer to char
   pv = &cData;

   //Dereferencing void pointer with char typecasting
   printf("cData = %c\n\n",*((char*)pv));

   //Pointer to int
   pv = &iData;

   //Dereferencing void pointer with int typecasting
   printf("iData = %d\n\n",*((int *)pv));

   return 0;
}
```

The same pointer is reused for multiple data types.
Type must be specified while dereferencing.

Arithmetic on void pointer

```
#include<stdio.h>

int main()
{
  int a[4] = {1, 5, 13, 4};
  void *pv = &a[0];
  pv = pv + 1;

  printf("Value %d\n", *((int *) pv) );

  return 0;
}
```

What will be the output?

Arithmetic on void pointer

```
#include<stdio.h>

int main()
{
  int a[4] = {1, 5, 13, 4};
  void *pv = &a[0];
  pv = pv + 1;

  printf("Value %d\n", *((int *) pv) );

  return 0;
}
```

What will be the output?

It will not print 5

pv+1 does not increment pv by scale_factor=4

Arithmetic on void pointer

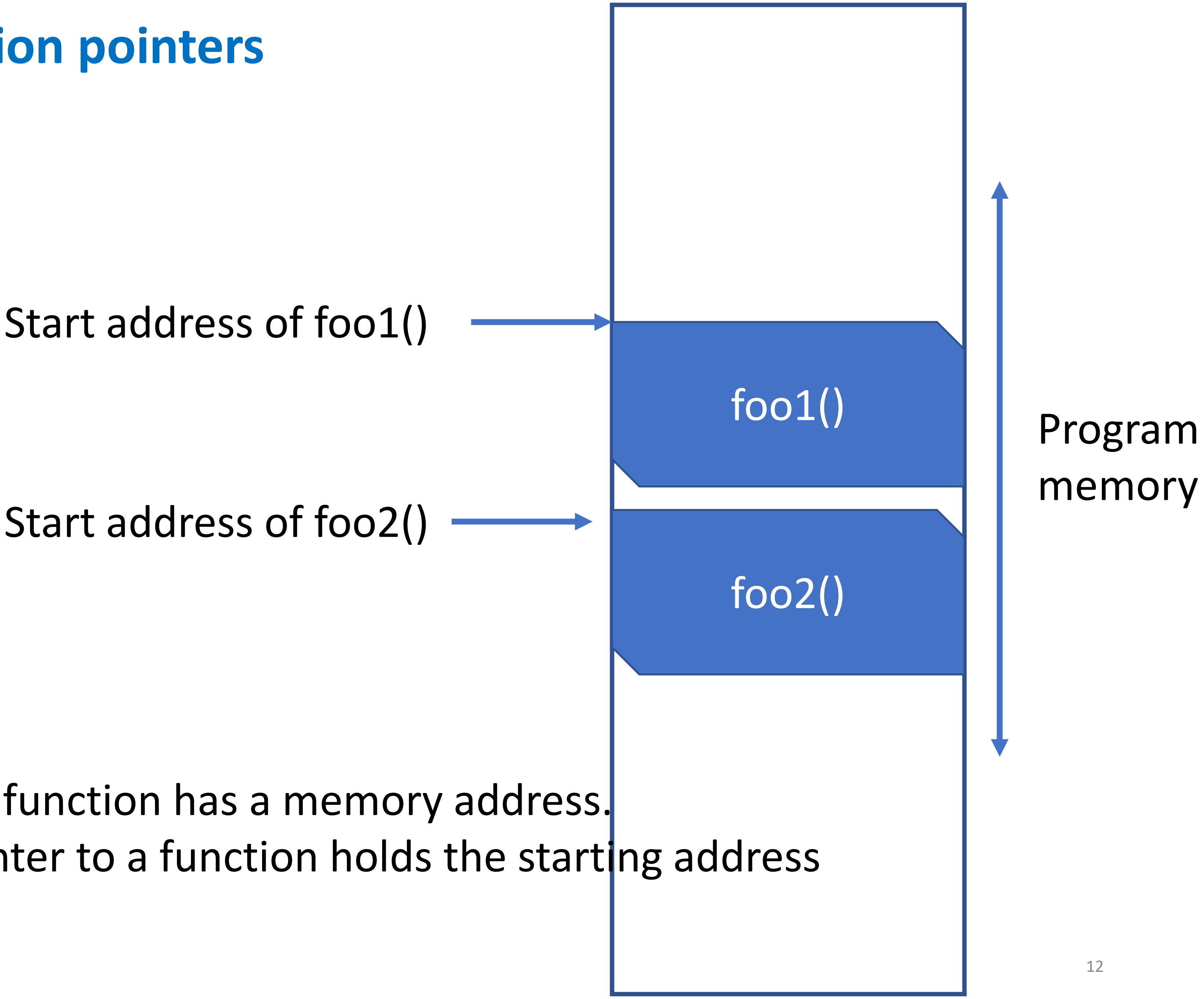
Perform proper typecasting on the void pointer before performing arithmetic operation.

```c
#include<stdio.h>

int main()
{
  int a[4] = {1, 5, 13, 4};
  void *pv = &a[0];
  pv = (int *) pv + 1;

  printf("Value %d\n", *((int *) pv) );

  return 0;
}
```

Now it prints 5

During pv+1 compiler increments pv by scale_factor=4

Function pointers



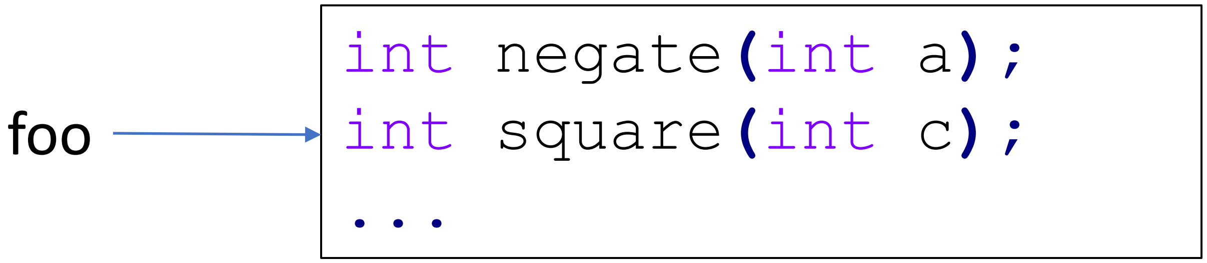
- Every function has a memory address.
- A pointer to a function holds the starting address

Function pointer syntax

Syntax for declaration

```
int  (*foo)(int);
```

- foo is a pointer to a function
- Where function takes one int argument and returns int.



```
int negate(int a);
int square(int c);
...
```

Foo can point to any of these functions

Function pointer syntax: careful

Function pointer declaration

```
int (*foo)(int);
```

Here function returns pointer of type int

```
int *foo(int);
```

To declare a function pointer ( ) must be used

Function pointer syntax

What is the meaning of this syntax?

```
int *(*foo)(int);
```

- foo is a pointer to a function
- Where function takes one int argument and **returns pointer to int**.

foo $\rightarrow$

```
int *negate(int a);
int *square(int c);
...
```

Foo can point to any of these functions

Initialization of function pointer

```
void int_func(int a)
{
  printf("%d\n", a);
}

int main()
{
  void (*foo)(int);

     // & is optional
  foo = &int_func;

  return 0;
}
```

Calling function using function pointer

```c
void int_func(int a)
{
  printf("%d\n", a);
}
int main()
{
  void (*foo)(int);

  // & is optional
  foo = &int_func;
  // two ways to call
  foo(2);
  (*foo)(3);
  return 0;
}
```